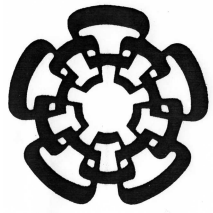

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS
AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE COMPUTACIÓN

"Transición del trabajo individual al colaborativo mediante técnicas de remodelación y redistribución plásticas"

T E S I S

Que presenta

MARCO ANTONIO CASTRO HERNÁNDEZ

Para obtener el grado de

MAESTRO EN CIENCIAS

EN LA ESPECIALIDAD DE

COMPUTACIÓN

Directora de la Tesis:

DRA. SONIA G. MENDOZA CHAPA

México, D.F. Diciembre, 2014

Agradecimientos

Al CONACyT, por el apoyo económico brindado sin el cual no hubiera podido culminar mis estudios de maestría.

A la Dra. Sonia G. Mendoza Chapa, por permitirme llevar a cabo este proyecto bajo su supervisión, pero sobre todo, por su paciencia, por las minuciosas y extraordinarias revisiones, por todas las horas que invirtió en mí, por su esfuerzo y sus valiosos consejos. Me siento muy afortunado por haber tenido esta oportunidad.

A los Dres. Amilcar Meneses Viveros y José Guadalupe Rodríguez García, por aceptar ser revisores del presente trabajo y por su valioso tiempo invertido.

A mis padres, por su apoyo incondicional para ayudarme a salir adelante y permitirme seguir mis sueños. Son mi mayor inspiración y mi modelo a seguir, sin ellos nunca habría llegado tan lejos.

A mis amistades y compañeros de clase, por su valiosa ayuda durante las fases iniciales y finales de este proyecto. Sus comentarios fueron una importante contribución.

A mi novia Ivonne Yanely, por sus palabras de ánimo y paciencia, por alentarme para seguir adelante y apoyarme cuando lo he necesitado.

Al CINVESTAV, por darme la oportunidad de continuar con mis estudios y proporcionarme los recursos para adquirir los conocimientos técnicos y científicos necesarios que me permitieron seguir desarrollándome profesionalmente.

Índice general

Índice de figuras

Índice de tablas

Índice de Algoritmos

Resumen

El término de **plasticidad** está inspirado en la propiedad de ciertos materiales que les permite expandirse y contraerse bajo ciertas restricciones naturales sin romperse, preservando así su usabilidad. En el ámbito de la HCI (*Human-Computer Interaction*), se puede definir como la capacidad que tiene un sistema interactivo para adaptarse a cambios de contexto de uso, preservando ciertas propiedades tales como la usabilidad y la continuidad de interacción.

En la actualidad, la plasticidad de interfaces de usuario es un área de estudio con bastante potencial, debido a la extensa variedad de dispositivos móviles que existen hoy en día y a las facilidades y opciones técnologicas que éstos poseen de forma inherente, cuya evolución es constante y perceptible con el paso del tiempo.

Hoy en dia, es difícil encontrar aplicaciones para dispositivos móviles que permitan a usuarios situados cara a cara interactuar de forma colaborativa para explotar las capacidades tecnológicas y de comunicación que poseen sus propios teléfonos inteligentes y tabletas. Así mismo, existe una gran variedad de aplicaciones que dan soporte al trabajo individual así como al colaborativo, sin embargo, estas aplicaciones no brindan un soporte adecuado que permita realizar una transición entre ambos tipos de trabajo, lo cual implica que aplicaciones diseñadas para el trabajo individual no puedan ser utilizadas para colaborar y viceversa.

A lo largo de la presente tesis, describimos un soporte multi-usuario y multi-plataforma para dispositivos móviles (teléfonos inteligentes y tabletas), que facilita de forma flexible transiciones del trabajo individual al trabajo colaborativo y viceversa, mediante técnicas de remodelación y redistribución plásticas. La contribución principal consiste en el diseño e implementación de un soporte (a modo de framework) que cumple con el propósito ya descrito. El diseño se realizó de tal forma, que pueda ser migrado a otras plataformas para poder crear una implementación que sea nativa en otros sistemas operativos. Adicionalmente, la implementación se entrega como un proyecto de tipo biblioteca (*project library*) para Android. Dicho proyecto cuenta con la documentación necesaria para conocer el funcionamiento de cada una de las funciones y clases que lo componen.

Abstract

The term **plasticity** is inspired on the property of certain materials that allows them to expand and contract under certain natural restrictions without breaking, thus preserving its usability. In the field of HCI (*Human-Computer Interaction*), it can be defined as the ability of an interactive system to adapt to changes on context of use while preserving certain properties such as usability and continuity of interaction.

At the present time, the plasticity of user interfaces is an area of research with great potential, due to the wide variety of mobile devices available and the communicaction and technology facilities that they possess inherently, whose evolution is constant and noticeable over time.

Nowadays, it is difficult to find applications for mobile devices that allow co-located users to interact collaboratively, in order to take advantage of the technological and communication features of their own smartphones and tablets. Likewise, there exists a wide variety of applications that support individual work as well as collaborative, however, these applications do not provide an adequate support to enable transitions between these two types of work, which means that applications designed for individual work, can not be used to collaborate and vice versa.

Throughout the present work, we describe a multi-user and multi-platform support for mobile devices (smartphones and tablets), which facilitates in a flexible way, to carry out transitions between individual and collaborative work modes, by relying on plastic remodeling and redistribution techniques. Our main contribution is the design and implementation of a support (given as a framework) that fulfills the purpose we just described. The design was done in such a way that it can be migrated to other platforms in order to create an implementation that is native in other operating systems. Additionally, the implementation is delivered as a project *project library* for Android. This project has the necessary documentation so that it is easy to understand how each of the functions and classes provided work and can be used.

Capítulo 1

Introducción

1.1. Contexto de investigación

Existe una intersección entre dos áreas de investigación: por un lado, las ciencias sociales y de comportamiento y, por el otro, las computadoras y tecnologías de la información. La Interacción Hombre-Máquina (mejor conocida como HCI por sus siglas en inglés) es una de las áreas que se puede ubicar en dicha intersección, la cual tiene como objetivo tratar de entender cómo es que la gente hace uso de dispositivos y sistemas capaces de llevar a cabo tareas de computación y cómo éstos pueden ser más útiles [Carroll, 2003]. Si bien la HCI es solamente una perspectiva del Trabajo Cooperativo Asistido por Computadora, es también un área de investigación sobre la cual está inscrito el soporte propuesto.

Actualmente, el paradigma de interacción con dispositivos (móviles o fijos) que poseen capacidades computacionales, está enfocado típicamente en interfaces gráficas de usuario (GUI por sus siglas en inglés) y comandos enviados mediante una variedad de configuraciones, la gran mayoría compuestas por dispositivos de entrada y salida tales como teclados, ratones y pantallas táctiles [Krumm, 2009]. Independientemente del tipo de dispositivos, la información que almacenan y procesan suele ser de carácter personal y requieren que el usuario esté completamente enfocado en las tareas que está realizando con dicha información. Las ideas que propone Mark Weiser son en sí un cambio de paradigma desde el punto de vista de interacción con dichos dispositivos computacionales, siendo una de las más representativas:

> *Las tecnologías más profundas son aquellas que desaparecen. Se entrelazan entre sí en la fábrica de la vida diaria hasta que son indistinguibles.*

El Cómputo Ubicuo, otra de las perspectivas del Trabajo Cooperativo Asistido por Computadora, tiene sus bases en esas ideas y busca desarrollar sistemas heterogéneos compuestos por una gran variedad de dispositivos de bajo y alto nivel (móviles y fijos), i.e., dispositivos que pueden o no requerir entradas directas de los usuarios. Idealmente, dichos dispositivos deberían estar embebidos en el ambiente que nos rodea permaneciendo indetectables, presentando información de retroalimentación únicamente cuando se les solicite. De esta forma, a pesar de que exista un gran número de dispositivos a nuestro alrededor, se evita que sean intrusivos. En este contexto, existe un tipo de interfaces conocidas como interfaces de usuario ubicuas (UUI por sus siglas en

inglés), las cuales son las encargadas de recolectar información acerca del contexto de uso del sistema, todo esto a través de hardware, software y servicios especiales (e.g., cámaras, micrófonos, acelerómetros o reconocimiento de patrones) [Krumm, 2009]. De esta forma, es posible identificar dos puntos clave:

- Con la tecnología actual, el desarrollo de sistemas que cumplan con las características anteriores sigue siendo un reto, debido a que se requiere hardware dedicado y a lo complicado que resulta desarrollar la capa de *middleware* para cada uno de los dispositivos que conformen el sistema (diferentes sistemas operativos y protocolos de comunicación).
- En la parte que nos compete, es posible tomar parte del concepto de lo que representa el Cómputo Ubicuo para desarrollar interfaces mixtas, i.e., una combinación de UUI y GUI tradicionales, desarrolladas con base en los sensores con los que cuenta la mayoría de los dispositivos móviles en la actualidad.

El objetivo de utilizar interfaces de tipo UUI en el Cómputo Ubicuo consiste en hacer desaparecer los dispositivos fijos en el ambiente y evitar que los usuarios se sientan abrumados por la cantidad de atención requerida para atender cada una de las alertas, mensajes y configuraciones que podrían solicitar dichos dispositivos en algún momento. Debido a la capacidad que tienen estas interfaces para identificar el contexto de uso del sistema al cual pertenecen y a que los dispositivos móviles ya mencionados cuentan con los sensores requeridos para llevar a cabo dicha tarea, es posible combinar ambas tecnologías para atacar otro tipo de problemas.

Finalmente, el Cómputo Móvil es otro de los campos de investigación sobre los cuales se inscribe el soporte de la presente tesis, el cual se puede definir básicamente como un entorno de computación en movimiento. Su objetivo consiste en permitir a un usuario acceder a datos, información u otro tipo de objetos lógicos desde cualquier dispositivo mientras se encuentra en movimiento. El principio que persigue es "información en donde sea y cuando sea" [Talukder et al., 2010]. Debido a que el soporte propuesto está enfocado a los dispositivos móviles, es indispensable tomar en cuenta los beneficios y las desventajas que éstos acarrean (e.g., duración de la batería, tamaño de la pantalla, alcance y velocidad de transmisión de datos), los cuales ya han sido ampliamente estudiados.

1.2. Motivación

En un principio, los teléfonos móviles fueron diseñados para permitir a un usuario comunicarse en cualquier lugar y en cualquier momento con otras personas. Con el paso del tiempo, nuevas tecnologías contribuyeron a la rápida evolución de estos dispositivos para dar paso a una nueva generación de teléfonos conocidos como teléfonos inteligentes (*smartphones*). Comparados con sus predecesores, los teléfonos inteligentes son capaces de llevar a cabo procesamientos computacionales y tareas que van más allá de una simple llamada telefónica, con la ventaja de que, al ser aún más pequeños y ligeros, brindan un mayor factor de movilidad. Adicionalmente, dichos dispositivos cuentan con diversos sensores y hardware dedicado (e.g., pantallas táctiles, acelerómetros, giroscopios, sensores de luz, sensores de proximidad o soporte para redes inalámbricas). Así mismo, otros dispositivos integrados con especificaciones similares, pero con pantallas de mayor tamaño y mayor poder de procesamiento (conocidos como tabletas) fueron

diseñados y son, al igual que los teléfonos inteligentes, ampliamente usados en todo el mundo.

Originalmente, estos dispositivos fueron pensados para su uso personal, con un grado de interacción social limitado y principalmente a distancia, sin embargo, avances en las tecnologías de comunicación han dado paso a nuevas formas de interaccionar, ya sea en entornos distribuidos o cara a cara. Por otro lado, no solo las formas en las que se comunican los dispositivos móviles están cambiando con el tiempo, sino también el abanico de aplicaciones de cada uno de sus componentes, i.e., eventualmente se están encontrado nuevas formas de uso para dicho hardware.

1.3. Planteamiento del problema

Hoy en dia, es difícil encontrar aplicaciones para dispositivos móviles que permitan a usuarios situados cara a cara interactuar de forma colaborativa para explotar las capacidades tecnológicas y de comunicación que poseen sus propios teléfonos inteligentes y tabletas. Así mismo, existe una gran variedad de aplicaciones que dan soporte al trabajo individual así como al colaborativo, sin embargo, estas aplicaciones no brindan un soporte adecuado que permita realizar una transición flexible entre ambos tipos de trabajo, lo cual implica que aplicaciones diseñadas para el trabajo individual no puedan ser utilizadas para colaborar y viceversa.

En particular, existen algunos prototipos que permiten habilitar un espacio compartido de trabajo, tales como: Mindmap [Lucero et al., 2010], Pass-Them-Around [Lucero et al., 2011] y ConnecTables [Tandler et al., 2001], sin embargo, su soporte está limitado en ciertos aspectos, tales como no permitir realizar operaciones concurrentes, ausencia de soporte para dispositivos heterogéneos, permitir una configuración regular únicamente dentro del arreglo de dispositivos, así como con un número limitado de los mismos, entre otros más. Adicionalmente, cabe mencionar que ninguno de los prototipos provee las herramientas necesarias para poder reproducir sus funcionalidades y resultados en otro tipo de aplicaciones, i.e., que las transiciones que permiten están limitadas dentro de sus respectivos modelos de tareas y no pueden ser extendidos o sustituidos.

La extensa variedad de dispositivos móviles que existe hoy en día, junto con las tecnologías que conforman a cada uno de sus componentes (tanto software como hardware), crean áreas de oportunidad en diversos campos de investigación (e.g., Cómputo Ubicuo, Cómputo Móvil o Trabajo Cooperativo Asistido por Computadora) que podrían ser explotadas para permitir a los usuarios verse envueltos en interacciones más dinámicas e intuitivas.

1.4. Objetivos

El objetivo general de la presente tesis consiste en desarrollar un soporte multi-usuario y multiplataforma para dispositivos móviles (teléfonos inteligentes y tabletas, específicamente) mediante técnicas de remodelación y redistribución plásticas, que facilite de forma flexible transiciones del trabajo individual al trabajo colaborativo y viceversa en entornos cara a cara.

A continuación se enumeran los objetivos específicos sobre los cuales está basado el desarrollo del soporte propuesto:

1. Diseñar e implementar un espacio de trabajo que pueda ser soportado tanto por un solo dispositivo (e.g., un teléfono inteligente) como por un arreglo irregular de dispositivos heterogéneos (e.g., tabletas y teléfonos inteligentes), limitado de forma física únicamente por la configuración del mismo y no por el número de elementos que lo componen. La formación del espacio de trabajo se basará en gestos intuitivos y flexibles.

2. Desarrollar un mecanismo de recuperación que permita cambiar dinámicamente la configuración del arreglo de dispositivos que soportan el espacio de trabajo, sin comprometer la sesión actual.

3. Diseñar e implementar una aplicación colaborativa que permita validar la factibilidad y utilidad del soporte propuesto.

4. Realizar pruebas con usuarios finales con el fin de evaluar la interfaz de usuario, así como los criterios de calidad hedónicos y pragmáticos de dicho espacio de trabajo.

Mediante el uso de técnicas de remodelación y redistribución plásticas en dispositivos móviles, es posible facilitar las trancisiones del trabajo individual al trabajo colaborativo y viceversa en entornos cara a cara, multi-usuario y multi-plataforma.

1.5. Organización de la tesis

El documento está dividido en ocho capítulos (ver figura 1.1). En el capítulo 2 se describe la teoría requerida para desarrollar el soporte propuesto, así como las técnicas mediante las cuales es posible desencadenar un acoplamiento de dispositivos. Así mismo, se propone una categorización de los diferentes tipos de gestos de acoplamiento y se describen algunos prototipos encontrados en la literatura que forman parte del trabajo relacionado y son representativos en este contexto. Posteriormente, en el capítulo 3 se presenta el análisis y diseño del soporte desarrollado, los cuales contemplan los aspectos de diseño sobre los cuales está basada toda la arquitectura propuesta, los problemas encontrados durante fases iniciales e intermedias de desarrollo, así como sus respectivas soluciones. Los algoritmos utilizados por cada uno de los servicios proporcionados y las configuraciones necesarias, son descritos en el capítulo 4. Una vez definida la implementación del soporte, en el capítulo 5 se presenta el diseño e implementación de una aplicación de ejemplo a modo de validación. Las pruebas con usuarios finales, junto con las herramientas utilizadas y los resultados obtenidos se presentan en el capítulo 6. Finalmente, se da una breve recapitulación del problema tratado, las conclusiones y se discute el trabajo a futuro en el capítulo 7.

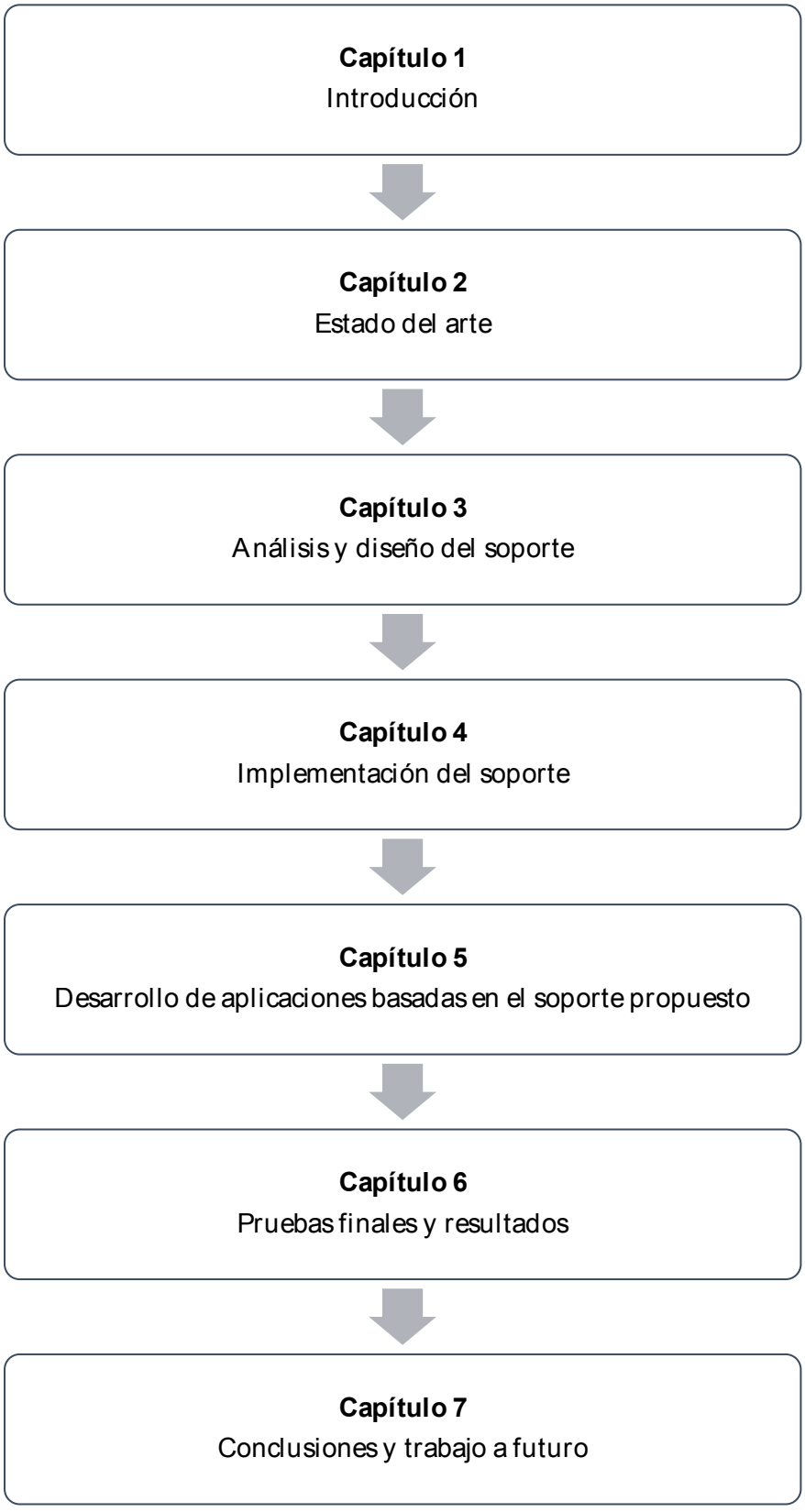

Figura 1.1: Organización de la tesis

Capítulo 2

Estado del arte

En este capítulo se describen los conceptos teóricos sobre los cuales está basada la presente tesis, así como una recopilación de métodos de acoplamiento y aplicaciones de ejemplo. En la sección 2.1 se define el concepto de plasticidad aplicado a las interfaces de usuario y se describen las variables contextuales que afectan a todo sistema interactivo. En la sección 2.2 se detalla en qué consiste un proceso de adaptación, así como los medios utilizados para ello y las variables que intervienen en este proceso. Posteriormente en las secciones 2.3, 2.4 y 2.5, se discute la importancia y las consideraciones que se deben tener si se desean utilizar métodos de acoplamiento, además, se propone una clasificación de los distintos métodos de acoplamiento identificados en la literatura. Finalmente, en la sección 2.6 se presenta un listado de algunos prototipos que resultan ser representativos dentro del estado del arte.

2.1. Plasticidad y Contexto de uso

El término de **plasticidad** está inspirado en la propiedad de ciertos materiales que les permite expandirse y contraerse bajo ciertas restricciones naturales sin romperse, preservando así su usabilidad. En el ámbito de la HCI (*Human-Computer Interaction*), se puede definir como la capacidad que tiene un sistema interactivo para adaptarse a cambios de contexto de uso, preservando ciertas propiedades tales como la usabilidad y la continuidad de interacción [Calvary et al., 2001b].

Antes de continuar, es indispensable definir la **usabilidad** en términos de la HCI. Este término, aunque puede ser aplicado en diferentes ámbitos, inicialmente se acotó en la norma ISO / IEC 9126 para abarcar únicamente las interfaces de usuario. Dicha norma fue desarrollada por separado como un estándar de ingeniería de software, la cual fue sustituida un año después por la norma ISO / IEC 9126-1. La definición que se proporciona es la siguiente:

> *Usabilidad: es la capacidad que tiene el Software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo ciertas condiciones específicas (i.e., contexto de uso)* [Bevan, 2001].

Adicionalmente, en esta norma se proponen dos métricas en donde se establecen diversos criterios de usabilidad:

- **Métricas externas**: son aquellas que se pueden utilizar para especificar o evaluar el comportamiento de un sistema cuando es operado por un usuario (e.g.: determinar cuánto tiempo le lleva a un usuario aprender a utilizar alguna funcionalidad del sistema, si los usuarios pueden deshacer acciones, entre otras más).

- **Métricas internas**: son aquellas que se pueden utilizar para crear requerimientos que describan propiedades estáticas de las interfaces que puedan ser evaluadas sin la necesidad de probar el sistema (e.g.: ¿Cuál es el porcentaje de funciones que estan documentadas?, ¿Cuál es el porcentaje de funciones que admiten la opción de deshacer?, etc.).

Las variables contextuales de un sistema interactivo definen tres espacios o fuentes de información [Calvary et al., 2007] [Coutaz and Calvary, 2012], los cuales se definen a continuación:

- **El modelo del usuario**: se refiere a la persona arquetípica que hará uso del sistema en algun momento. Este modelo puede abarcar ciertas funciones o atributos, tales como: preferencias de usuario básicas, perfiles, predicción de actividades dependiendo del uso de las mismas a lo largo del tiempo, entre otras.

- **El modelo de la plataforma**: hardware y software que conforman a un dispositivo (i.e., componentes tangibles y conceptuales). Extendiendo este concepto, se puede considerar como plataforma al conjunto de recursos: de cómputo, sensado, comunicación por red y de interacción, los cuales enlazan el mundo físico con el virtual. En la actualidad, esta fuente de información no está representada solamente por dispositivos individuales, sino también por diferentes agrupaciones de los recursos heterogéneos ya mencionados.

- **El modelo del ambiente físico**: es aquel en el cual se lleva a cabo la interación con el sistema interactivo, conformado por un grupo de entidades circundantes al conjunto de tareas que el sistema puede realizar en un momento determinado bajo un estado específico (estas entidades pueden ser objetos, personas o eventos). Debido a la generalización de la definición de ambiente físico en este contexto, es posible acotar este elemento como el conjunto de entidades que pertenecen al dominio de la aplicación y que, en algún momento dado, podrían afectar o requerir cambios en el estado del sistema [Calvary et al., 2001a].

2.2. Adaptación

Es posible definir una adaptación como una reacción a cambios de contexto específicos [Coutaz and Calvary, 2012]. Como se mencionó anteriormente, un sistema interactivo debe tener la capacidad de adaptarse a los cambios de contexto de uso; seguir esta pauta implica conservar dos propiedades importantes: que se tenga la **capacidad de adaptación** y que se cuente con un alto grado de **adaptabilidad** [Thevenin and Coutaz, 1999]. Existe una diferencia específica entre ambos términos, los cuales se describen a continuación:

- La capacidad de adaptación de un sistema es aquella que le permite ser personalizado mediante solicitudes explícitas del usuario, de acuerdo a un conjunto de parámetros predefinidos.

- La adaptabilidad es la capacidad de un sistema que le permite realizar adaptaciones sobre sí mismo sin previa acción del usuario.

En este contexto, un proceso de adaptación se puede llevar a cabo a través de los siguientes medios [Coutaz and Calvary, 2012]:

- **Redistribución**: se refiere a la división y dispersión de la interfaz de usuario entre varios dispositivos, lo cual implica que actúa específicamente en la capa de Hardware.
- **Remodelación**: es cuando se adecúa la representación del dominio de la aplicación y los componentes que forman parte de la tareas del usuario. Actúa en la capa de Software.

Los elementos del sistema afectados por la adaptación, dependerán de la naturaleza del cambio sujeto a un contexto en particular. De acuerdo a Thevenin et. al., independientemente de que la adaptación haya sido solicitada por el usuario o sea de forma automática, este espacio de diseño se puede separar formalmente en tres ejes ortogonales: el **objetivo**, las **formas** y el **tiempo** [Thevenin and Coutaz, 1999].

- El objetivo de la adaptación: denota la entidad objetivo de la adaptación (el usuario, el ambiente o las características físicas del sistema, en caso de que apliquen).
- Las formas de la adaptación: señala los componentes (software) afectados por la adaptación (el modelo de tareas del sistema, las técnicas de representación de la información o de la interfaz de usuario y los subsistemas de ayuda).
- La dimensión temporal de la adaptación: puede ser estática o dinámica. Las unidades de tiempo generalmente son lógicas, i.e., puede estar marcadas por eventos, sesiones o cualquier tipo de acción referente al contexto del sistema.

Para que un sistema interactivo pueda realizar adaptaciones plásticas adecuadas, el perfil general de los usuarios finales debe ser tomado en cuenta durante las fases tempranas de desarrollo del sistema. De este modo, las interfaces desarrolladas se ajustarán a las necesidades y preferencias de los usuarios, brindando así continuidad y correctitud en las interacciones con dicho sistema.

2.3. Acoplamiento de dispositivos

El grado de usabilidad del soporte propuesto en la presente tesis, está definido no únicamente por un buen diseño de interfaces gráficas de usuario (de las aplicaciones en las cuales estará integrado), sino por la combinación de éstas junto con métodos de interacción sólidos que permitan hacer uso de cada una de las funcionalidades del sistema, alcanzables a través de la GUI o la UUI.

Mediante el acoplamiento de dispositivos se pretende:

- Permitir transiciones entre el trabajo colaborativo (específicamente, para interacciones cara a cara) y el individual sobre aplicaciones que inicialmente estaban destinadas a ser únicamente de tipo colaborativo o individual.

- Aprovechar los recursos fisicos que ofrecen los dispositivos móviles (sensores y pantallas) para elaborar métodos de interacción más intuitivos y desplegar interfaces de usuario que serían difíciles de utilizar en un solo dispositivo.
- Ajustar automáticamente el modelo de tareas del sistema dependiendo del modo de trabajo seleccionado.

Existen ciertas consideraciones a tomar en cuenta, al momento de elegir o diseñar un método acoplamiento de dispositivos:

- Las condiciones físicas del entorno en el cual se hará uso del soporte (contexto de uso), i.e., si las interacciones colaborativas pueden o no suceder en exteriores y no solamente dentro de habitaciones, oficinas o laboratorios. Cabe mencionar que dichas condiciones físicas pueden ser ambientales (humedad, iluminación, etc.), geográficas (ubicación) o tecnológicas (conectividad a internet, disponibilidad de redes inalámbricas de corto o largo alcance, etc.).
- Las características de los dispositivos en términos de hardware (e.g., sensores, tamaño de la pantalla, procesador, medios de comunicación, etc.) y software (e.g., versión del sistema operativo, resolución de la pantalla, dependencia de otras aplicaciones o servicios, etc.) para dar el soporte adecuado y métodos de acoplamiento alternativos cuando sea necesario.
- Los requerimientos de la aplicación que hará uso del acoplamiento de dispositivos, e.g., si se debe conocer la orientación o ubicación relativa o absoluta de los dispositivos, etc.
- El estado final de los dispositivos posterior al acoplamiento, i.e., si éstos se colocarán sobre alguna superficie o si estarán en las manos de los colaboradores todo el tiempo.

Así mismo, es posible utilizar métodos de acoplamiento mucho más sofisticados que los tradicionales (e.g., escaneos que muestran listas con dispositivos disponibles para conexiones mediante Bluetooth o WiFi). Lo que se busca es que los usuarios puedan seleccionar mediante gestos o interacciones físicas con los dispositivos (conocidos como **gestos de acoplamiento**), aquellas entidades participantes en el proceso de acoplamiento. Utilizando dichos medios de seleccion que puedan ser entendidos y traducidos de forma correcta por los dispositivos, es posible eliminar la necesidad de conocer con anterioridad datos técnicos de los mismos (e.g., dirección MAC, dirección IP, nombre de una red WiFi, nombre del dispositivo, etc.). Al facilitar la identificación de dispositivos, se les brinda a los usuarios la capacidad de expresar acciones o ejecutar comandos sobre éstos de una forma más natural y al sistema la capacidad de entenderlos [Peng et al., 2009].

2.4. Métodos de acoplamiento

En realidad, existe una variedad de métodos de acoplamiento que cumplen con un propósito similar al que buscamos. El problema radica en que muchos de ellos se adaptan a contextos distintos o están contenidos en sistemas más complejos, cuyos objetivos dependen específicamente del dominio de sus respectivas aplicaciones (e.g., autenticación mutua, intercambio de archivos, etc.). Es posible reproducir la mayoría de estos métodos aislando su funcionamiento del modelo de tareas de los sistemas en los cuales están contenidos. Cabe mencionar que dichos métodos de acoplamiento forman parte de la UUI de los sistemas en los cuales se utiliza esta técnica.

A continuación se proponen algunas categorías y se describen a modo de ejemplo, algunos sistemas que hacen uso de estos métodos. Como nota adicional, aunque en las descripciones de las categorías se ejemplifican los medios de interacción con recursos que generalmente están integrados de fábrica en los dispositivos, es posible utilizar hardware dedicado embebido para incrementar su funcionalidad y proveer al sistema de una mayor cantidad de información sobre el contexto de uso.

2.4.1. Basados en el uso de interfaces y sensores conscientes de contexto

En esta categoría se encuentran aquellos métodos que hacen uso de adaptadores de red (integrados en los dispositivos), cámaras o de uno o más sensores (e.g., acelerómetro, giroscopio, sensor de proximidad, de iluminación, magnetómetro, sensor de temperatura y de humedad), con el objetivo de determinar el contexto de uso del sistema en el cual está integrado el método de acoplamiento. Dichos sensores y dispositivos son capaces de brindar información relevante del entorno que tenga efecto directamente sobre el teléfono inteligente o tableta o el estado en el que estos se encuentran.

Mediante este tipo de métodos, es posible definir un comportamiento específico en un sistema dependiendo del contexto identificado y los sensores o adaptadores utilizados, desencadenando como resultado, la actualización de las interfaces disponibles (GUI y/o UUI) y el ajuste del modelo de tareas disponibles para los usuarios. Un ejemplo sería el siguiente: un sistema diseñado para dar soporte a interacciones cara a cara y al acoplamiento de dispositivos, podría diferenciar si un intento de acoplamiento tiene como objetivo extender el espacio de trabajo (añadir dispositivos para formar un arreglo de éstos) o entablar una sesión privada con otro usuario para el intercambio de información (como podría ser el intercambio de objetos virtuales personales entre dos personas situadas una enfrente de la otra, cada una interactuando con su propio dispositivo).

Shake well before use [Mayrhofer and Gellersen, 2009] es un mecanismo de autenticación y acoplamiento seguro para dispositivos móviles. Para realizar acoplamientos seguros, hace uso de movimientos físicos similares entre ambos dispositivos. Dichos movimientos son configurables y, debido a su naturaleza, es posible realizar acoplamientos sin la necesidad de interfaces gráficas. Adicionalmente, se pueden crear gestos de un solo uso, lo cual lo hace más seguro. Al no requerir de pantallas para el desplegado de información, se utilizan formas de retroalimentación alternativas (e.g. vibraciones o sonidos) para la confirmación de una conexión exitosa o fallida.

Amigo [Varshavsky et al., 2007] es en realidad un método de autenticación basado en proximidad. Su objetivo es autenticar correctamente a dos dispositivos de forma segura, basándose en las condiciones del entorno en el cual se intenta realizar el acoplamiento entre ambos dispositivos, e.g., intensidad en la conectividad de la señal WiFi actual, características y estado de las redes inalámbricas disponibles (celulares, bluetooth, WiFi), entre otras más. Este método está basado en la premisa de que, aunque se conozcan ciertos aspectos del ambiente en el cual se intentará realizar un acoplamiento, es muy difícil predecir las condiciones exactas de dicho espacio físico, dado un lugar y un tiempo en específico. Este método consta de cuatro fases principales, desde el inicio de la comunicación hasta el resultado de la autenticación: alineamiento de paquetes iniciales, división de los paquetes en rebanadas, extracción de características y clasificación

(mediante aproximaciones).

Por otro lado, **Hinckley** [Hinckley, 2003] propone el acoplamiento de tabletas (aumentadas con acelerómetros de dos ejes) golpeando ligeramente una tableta contra la otra por la orilla. Este gesto indica de forma implícita no solamente los lados por los cuales se desea unir los dispositivos, sino también una jerarquía que distingue el dispositivo cuya pantalla será utilizada para ampliar el espacio de trabajo y el dispositivo cuyo espacio de trabajo será expandido.

2.4.2. Basados en el uso de gestos sobre la pantalla táctil

Actualmente, las pantallas táctiles son un recurso imprescindible en los teléfonos inteligentes y las tabletas, pues la mayor parte de las interacciones con los sistemas y aplicaciones se llevan a cabo a través de estas interfaces. Dichas interacciones suelen ser a través de gestos sobre la pantalla táctil (ya sea con uno o varios dedos), los cuales inherentemente permiten a un sistema determinar ciertas características del mismo, tales como la dirección, el tamaño, la velocidad, ubicación, entre otras más. El punto clave radica en que es posible contextualizar un gesto a través de estas características para reaccionar de forma apropiada, e.g., un gesto en línea recta sobre la pantalla de un dispositivo nos brinda la siguiente información: punto de inicio y fin del gesto, dirección, velocidad, duración y número de dedos utilizados; con estos datos, el sistema podría deducir la posición relativa de cada uno de los dispositivos y parametrizar las otras características para usarlas como medio de comparación. De esta forma es posible proceder con el acoplamiento si dichos parámetros concuerdan entre sí o si tienen una relación lógica dentro del dominio de la aplicación.

Mindmap [Lucero et al., 2010], **Pass-Them-Around** [Lucero et al., 2011] y **MobiComics** [Lucero et al., 2012a] son parte de los prototipos basados en la plataforma SSI de Nokia, los cuales utilizan un mecanismo de acoplamiento basado en un gesto táctil concurrente sobre las pantallas de los dispositivos que se desean unir. Las direcciones de ambos gestos indican específicamente la orientación de éstos dentro del arreglo a ensamblar, haciendo una analogía a cómo una cinta adhesiva pegaría dos objetos, uno al lado del otro.

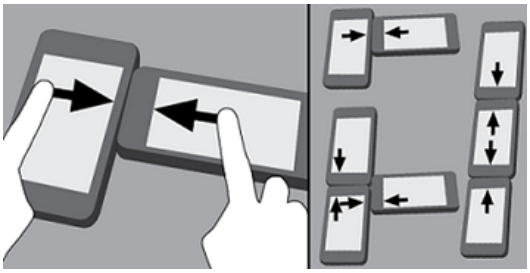

Figura 2.1: Combinaciones del gesto de acoplamiento utilizado por la plataforma SSI.

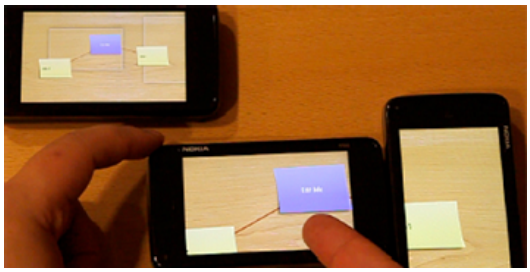

Figura 2.2: Vista general de Mindmap después de acoplar dos dispositivos.

Stitching [Hinckley et al., 2004] es una técnica de interacción basada en gestos síncronos sobre las pantallas de dos dispositivos. El gesto consiste en tocar la primera pantalla con una pluma en

una cierta posición y arrastrarla hasta la orilla del dispositivo, extendiendo esta acción hasta la segunda pantalla. De este modo, el camino formado al arrastrar la pluma indicará el contexto del gesto dentro del sistema, i.e., de forma inherente es posible conocer cuáles son los dispositivos que participarán en la interacción colaborativa, la jerarquía a adquirir en caso de que alguna tarea del sistema lo permita o requiera, la geometría del ecosistema colaborativo formado y el punto de partida y arribo en el mismo. Aunque esta técnica originalmente estaba diseñada para utilizarse con plumas, evidentemente es posible extender su uso a los dispositivos móviles que existen hoy en día.

Stitching calcula el ángulo de la línea que conforma el gesto en ambos dispositivos para determinar si es un intento de acoplamiento, ya que de acuerdo a sus resultados existe un espacio en la orilla de las pantallas en el cual la tasa de recolección de datos es más baja, por lo tanto el uso de coordenadas cerca de esta área conduciría a resultados poco fiables (restricción que ya no aplica en teléfonos inteligentes y tabletas).

Un punto importante a notar es el hecho de que, a pesar de que las técnicas mencionadas hacen uso de gestos que asemejan un movimiento rectilíneo, este no es un requerimiento estricto ya que el contexto de la aplicación, el modelo de tareas y el nivel de usabilidad del sistema son algunas de las características que se deben tomar en cuenta si se desea definir un gesto de interacción intuitivo.

2.4.3. Basados en el uso de interfaces de comunicación alternativas

Finalmente, se encuentran aquellos métodos que hacen uso de interfaces de comunicación alternativas, las cuales no estaban planeadas para transmitir datos a otros dispositivos, sino a los usuarios de los dispositivos. Como ejemplo de estas interfaces podrían ser los micrófonos y las bocinas de los teléfonos inteligentes y las tabletas. A través de dichas interfaces se pueden reproducir sonidos que puedan ser captados y procesados por otros dispositivos para determinar ciertas variables contextuales, tales como la posición relativa o la identificación de participantes en el intento de acoplamiento.

Un ejemplo de esta categoría es **Beep** [Mandal et al., 2005], el cual es un sistema de localización 3D basado en el uso de sonidos audibles para el posicionamiento de otros dispositivos. Está basado en solicitudes directas por el usuario, i.e., no es un sistema que se active de forma independiente (lo cual implica que, sin importar el contexto de uso, no se activará), ya que al utilizar sonidos audibles se pierde discreción y privacidad. Cabe recalcar que los sonidos utilizados son producidos por los mismos dispositivos, ya que son mucho más sencillos de controlar y procesar.

Beep utiliza la técnica de triangulación mediante tiempos estimados de arribo (ETA por sus siglas en inglés), i.e., el tiempo que tarda un sonido en llegar a un receptor desde el instante en el cual fue emitido.

Dentro de esta categoría también se encuentra **Point&Connect** [Peng et al., 2009], sistema cuyo objetivo es el emparejamiento de dispositivos mediante sonidos audibles (al igual que Beep). En realidad, este sistema establece su propio protocolo para cubrir las diferentes fases requeridas por un intento de acoplamiento seguro. La idea es simple: para acoplar un dispositivo con otro en un

ambiente en donde existen más de dos dispositivos, basta con sujetar uno de ellos y desplazarlo una cierta distancia con dirección hacia donde se encuentra el otro dispositivo.

Para la fase de selección, este sistema hace uso de un par de sonidos como señal de solicitud de acoplamiento. Estos sonidos son captados por los dispositivos cercanos, los cuales a su vez responderán emitiendo otro par de sonidos para que el receptor pueda medir la distancia relativa a cada uno de ellos. Este procedimiento se lleva a cabo dos veces, ya que teóricamente al mover al emisor de lugar y volver a realizar las mediciones, el dispositivo que presente un mayor cambio en la distancia medida será el que participará en el proceso de acoplamiento.

2.5. Trabajos relacionados

A continuación se presenta una recopilación del estado del arte encontrado en la literatura, relacionado con el tema de la tesis desarrollado.

2.5.1. ConnecTable

ConnecTable [Tandler et al., 2001] es un dispositivo consciente de contexto, el cual brinda soporte para la transición del trabajo individual al colaborativo, i.e., permite expandir el espacio de trabajo con el propósito de compartir el uso del mismo, superando de esa forma las restricciones físicas que conlleva el uso de una sola pantalla para interacciones colaborativas. ConnecTable hace uso de sensores dedicados (ya instalados) para el rastreo y acoplamiento con otras ConnecTables, por lo cual requiere de hardware especial. Así mismo, debido a su composición solamente es posible unir ConnecTables en arreglos de 2 elementos. Adicionalmente, no cuenta con soporte para el cambio de posiciones físicas de las mismas, ya que sólo permite el acoplamiento de los dispositivos tocando sus respectivas partes superiores.

Figura 2.3: Ejemplo de algunas configuraciones posibles con las ConnecTables

2.5.2. Mindmap

Mindmap [Lucero et al., 2010] es una de las posibles aplicaciones que se pueden implementar mediante la plataforma SSI (*Social and Spatial Interactions*) propuesta por un grupo de investigadores de Nokia. En sí, es una herramienta para dispositivos móviles que da soporte a las lluvias de ideas, permitiendo a un grupo de trabajo: crear, editar y visualizar notas virtuales. La idea original consiste en permitir a un número pequeño de personas realizar las acciones mencionadas de manera colaborativa sobre cualquier superficie plana, teniendo cada usuario su propio

dispositivo y la posibilidad de poder utilizarlos y compartirlos de forma indistinta. Mindmap da soporte al espacio de trabajo en múltiples dispositivos a través de un gesto en las pantallas que se desean unir.

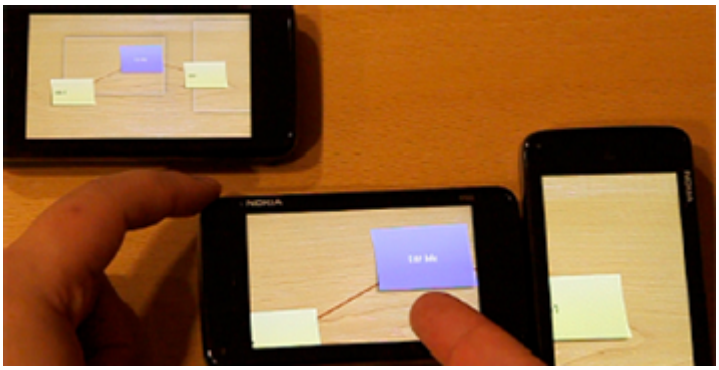

Figura 2.4: Vista general de Mindmap

2.5.3. Pass-Them-Around

Pass-Them-Around [Lucero et al., 2011] es otra de las aplicaciones que se pueden implementar mediante SSI. Su objetivo es permitir a un grupo muy pequeño de personas (dispuestas de forma inmediata alrededor de una mesa) intercambiar fotos, utilizando la metáfora del intercambio de fotografías impresas pasando una por una mano por mano. Para el rastreo de las posiciones relativas sobre una superficie plana, se utilizaron dispositivos embebidos con tecnología de rastreo por radio (sensores personalizados no incluidos de fábrica). Pass-Them-Around también da soporte a espacios de trabajo compartidos, sin embargo no permite arreglos irregulares de dispositivos y, como ya se ha mencionado, requiere de hardware externo para su funcionamiento.

Figura 2.5: Vista general de Pass-Them-Around

2.5.4. EasyGroups

EasyGroups [Lucero et al., 2012b] es un método de enlazado de dispositivos, el cual permite a usuarios ubicados en el mismo espacio físico establecer un ecosistema listo para poder interactuar,

sin la necesidad de otro tipo de hardware (como fue el caso de la tecnología de rastreo necesaria para Pass-Them-Around). La consideración más relevante para EasyGroups es el hecho de que los usuarios deben estar situados en círculo (como si estuvieran alrededor de una mesa), ya que ese es el orden en el cual se acoplarán los dispositivos, lo cual le quita un poco flexibilidad a este método de enlazado.

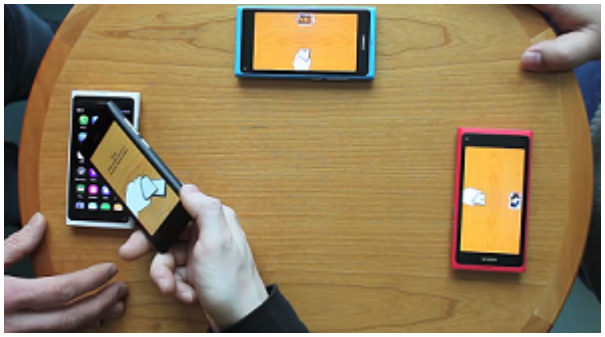

Figura 2.6: Vista del inicio del armado de un grupo con EasyGroups

2.5.5. MobiComics

MobiComics [Lucero et al., 2012a] es una aplicación que permite a un número limitado de usuarios crear tiras cómicas con sus dispositivos móviles, haciendo una separación directa entre los espacios públicos y privados (siendo los privados los mismos dispositivos de los usuarios y los públicos pantallas externas que puede usar cualquier usuario que lo desee). La aplicación permite a los usuarios compartir las tiras cómicas desde espacios privados a otros espacios privados o públicos. Al igual que Pass-Them-Around, MobiComics requiere de hardware externo para la detección de las posiciones relativas de los otros usuarios. A diferencia de las tres aplicaciones anteriores, MobiComics soporta operaciones concurrentes de grano grueso sobre el mismo objeto, i.e., operaciones sobre la misma tira cómica pero no sobre la misma burbuja de diálogo. Adicionalmente, debido al mismo diseño de la aplicación, únicamente se pueden unir los espacios de trabajo de dos dispositivos sin la posibilidad de cambiar sus respectivas posiciones de forma libre, ya que cada alineación implica una acción distinta.

Figura 2.7: Ejemplo de una tira creada mediante MobiComics

2.5.6. MobIES

MobIES [Schneider et al., 2012] es un sistema capaz de extender el espacio de trabajo de un dispositivo móvil a pantallas externas, aprovechando así el espacio extra proporcionado por la misma. Para acoplar el dispositivo móvil con la pantalla, simplemente se debe acercar a uno de los bordes de la pantalla. La conexión se realiza a través de tags NFC y códigos grabados en éstos, los cuales están situados de forma ordenada en las orillas de la pantalla. Para su funcionamiento, requiere de un servidor conectado a la pantalla, el cual estará a cargo de manipular el contenido de la misma.

Figura 2.8: Vista general de un prototipo de ejemplo utilizando MobIES

2.6. Análisis comparativo

Como se pudo observar en la sección anterior, existe una variedad de prototipos que cumplen con un propósito similar al que buscamos. Como punto de partida, es indispensable identificar las limitaciones que presentan dichos prototipos para que el soporte desarrollado sea mucho más robusto. Como se puede apreciar en las tablas 2.1 y 2.2, mediante el soporte desarrollado se pretende solventar las limitaciones de cada uno de los prototipos para mejorar las experiencias colaborativas con los sistemas en los cuales se utilice. Actualmente no existe una biblioteca reutilizable que permita realizar las transiciones entre el trabajo colaborativo y viceversa.

	Heterogeneidad de dispositivos	**Tamaño del arreglo**	**Tipo de arreglo**	**Requiere HW externo**
ConnecTables	No	1 - 2	Regular	No
Mindmap	No	-	Regular	No
Pass-them-around	No	1 - 4	Regular	Sí
MobiComics	No	1 - 3	Regular	Sí

Tabla 2.1: Comparación entre el soporte propuesto y los prototipos descritos (parte 1).

	Proporciona biblioteca(s) para reutilizarse	**Fases del acoplamiento configurables**
ConnecTables	No	No
Mindmap	No	-
Pass-them-around	No	-
MobiComics	No	-

Tabla 2.2: Comparación entre el soporte propuesto y los prototipos descritos (parte 2).

Capítulo 3

Análisis y diseño del soporte

En el presente capítulo se describe el análisis y diseño previos a la implementación del soporte propuesto. La solución obtenida provee a los desarrolladores las herramientas necesarias para habilitar transiciones entre los tipos de trabajo individual y colaborativo, independientemente del modelo de tareas de la aplicación final.

En cada una de las consecuentes secciones, se irá remarcando la relación que guarda cada uno de los conceptos definidos en el capítulo 2 con el correspondiente punto de análisis o diseño planteado. En primer lugar, en la sección 3.1 se analizan algunos aspectos identificados y que consideramos clave para la etapa de diseño. En la sección 3.2 se detalla la arquitectura general utilizada como base para el soporte, así como la división de tareas en cada uno de los servicios que lo conforman. Posteriormente, en la sección 3.3 se describen algunos de los problemas presentados durante las fases iniciales del ciclo de desarrollo del soporte y los ajustes que se tuvieron que llevar a cabo para solventarlos. Finalmente, en la sección 3.4 se muestra mediante una serie de diagramas (paquetes, clases y casos de uso), cómo están compuestos los servicios y cómo están distribuidos cada uno de sus componentes.

3.1. Análisis de los aspectos clave del soporte propuesto

Como ya se ha explicado anteriormente, buscamos habilitar transiciones flexibles entre un modo de trabajo y otro. En la presente tesis, definimos una **sesión colaborativa** como la representación tangible del modo de trabajo colaborativo, i.e., la utilización del trabajo colaborativo mediante la extensión del modelo de tareas de una aplicación inicialmente ejecutándose en el modo de trabajo individual. Dichas sesiones ofrecen a todos los colaboradores, un espacio de trabajo compartido único (y por consiguiente, de los objetos que éste posee), el cual abarca físicamente cada una de las pantallas de los dispositivos participantes en la sesión actual. Por otro lado, el modo de trabajo individual implica la utilización de una aplicación en un solo dispositivo sin la compartición de objetos ni el aprovechamiento de otros recursos (e.g., pantallas o sensores adicionales de otros dispositivos).

Para cumplir con este esquema, se tomaron cuatro aspectos de diseño como base para la definición y diseño del soporte propuesto:

- **Conectividad**: contar con la posibilidad de iniciar o terminar sesiones colaborativas, así co-

mo con la capacidad de comunicarse en cualquier momento con cada uno de los dispositivos integrantes (o potenciales integrantes) de una sesión, independientemente de los medios (interfaces) utilizados con este fin. La comunicación consiste en el envío y recepción de mensajes que pueden ser procesados y entendidos por los receptores, dado que dichos mensajes corresponden al modelo de tareas de la aplicación cuando ésta se encuentra operando bajo el modo de trabajo colaborativo.

- **Geometría**: la determinación de la posición relativa de uno o varios dispositivos en aplicaciones conscientes de la ubicación sin la necesidad de hardware especial, es aún un problema ampliamente estudiado. El soporte debe proveer la funcionalidad para determinar la relación especial que existe entre dispositivos utilizados cara a cara, sin requerir de dicho hardware.

- **Acoplamiento**: en la sección 2.3 se mencionaron algunas ventajas y características de esta técnica. La activación de los intentos de acoplamiento debe ser fácil e intuitivo para los usuarios de una aplicación, sin embargo, independientemente de la categoría a la que pertenezca(n) (sección 2.4), los gestos de acoplamiento no deben interferir con los gestos utilizados para interactuar con los objetos desplegados en el espacio de trabajo. Adicionalmente, se requiere retroalimentación multimodal mediante sonidos o háptica, producto del uso de interfaces ubicuas. Con ello se asegura que el usuario pueda ser consciente del momento en que un comando ha sido enviado (producto de una interacción con una UUI o GUI) y si está siendo procesado por el sistema.

- **Consciencia de grupo**: cuando una aplicación se encuentre en el modo de trabajo colaborativo, debe ser posible identificar las entidades que son partícipes de una sesión.

Es posible utilizar las soluciones en conjunto de cada uno de los aspectos de diseño ya mencionados, a pesar de que no existe una relación directa o dependencia funcional entre estos. Para ejemplificar la utilización de dichos aspectos, presentamos el siguiente escenario: suponiendo que se tiene una aplicación utilizada para desplegar mapas, con la capacidad para mostrar una sección del mapa en el espacio de trabajo de cada dispositivo (una sola sección en el modo de trabajo individual y secciones contiguas en el modo de trabajo colaborativo) se requiere conectividad constante entre cada teléfono o tableta acoplada, para comunicar cualquier evento que detecte alguno de los participantes (e.g., abrir, cerrar o cambiar mapa, colocar marcadores, agregar notas o cualquier otro evento que afecte el contenido del espacio de trabajo compartido). Además, para actualizar correctamente la sección del mapa que le corresponde a cada dispositivo, es indispensable que todos conozcan las posiciones relativas de al menos uno de los miembros de la sesión colaborativa actual (de esta forma, los dispositivos podrán determinar qué sección del mapa desplegar de acuerdo a dichas posiciones).

3.2. Arquitectura general del soporte

Se identificaron tres tareas principales que podrían permitir que una aplicación inicialmente de uso individual, permita realizar transiciones al trabajo colaborativo (y viceversa) de forma transparente. Dichas tareas hacen uso de los aspectos de diseño mencionados en la sección 3.1:

1. Reconocimiento e identificación de dispositivos compatibles, i.e., de dispositivos que estén ejecutando una aplicación con los mismos servicios (idealmente, que estén ejecutando la misma aplicación) y cuya implementación haga uso del soporte propuesto.

2. Acoplamiento de dos o más dispositivos. El acoplamiento involucra determinar el posicionamiento relativo de cada dispositivo para la posterior actualización del espacio de trabajo, así como la comunicación de las características de los dispositivos involucrados. De esta forma, se le permite a cada uno de los integrantes de una sesión colaborativa tener conciencia sobre las capacidades tecnológicas que posee cada entidad acoplada.

3. Comunicación de cada uno de los eventos directos e indirectos que afecten alguno de los objetos existentes en el espacio de trabajo compartido o al espacio de trabajo mismo.

Para cumplir con dichas tareas, el soporte está compuesto por tres servicios principales pensados y diseñados bajo una arquitectura P2P (figura 3.1), i.e., operan de forma independiente entre un dispositivo y otro, por lo tanto, aplicaciones que implementen el soporte y se encuentren ejecutándose en diferentes dispositivos pueden hacer uso de estos servicios cuando sean requeridos, sin depender de alguna entidad externa. En este contexto, cada dispositivo se puede considerar como una unidad funcional integrada por dichos servicios, la cual es independiente de otras unidades funcionales, i.e., una aplicación estará operando sin restricciones en el modo de trabajo individual, a pesar de que no se utilicen los tres servicios disponibles que forman parte del soporte. Cada unidad funcional entonces alcanza su máximo nivel de efectividad cuando es utilizada en el modo de trabajo colaborativo, i.e., cuando se utiliza en conjunto con otras unidades funcionales al explotar los servicios proporcionados. A continuación se describen dichos servicios.

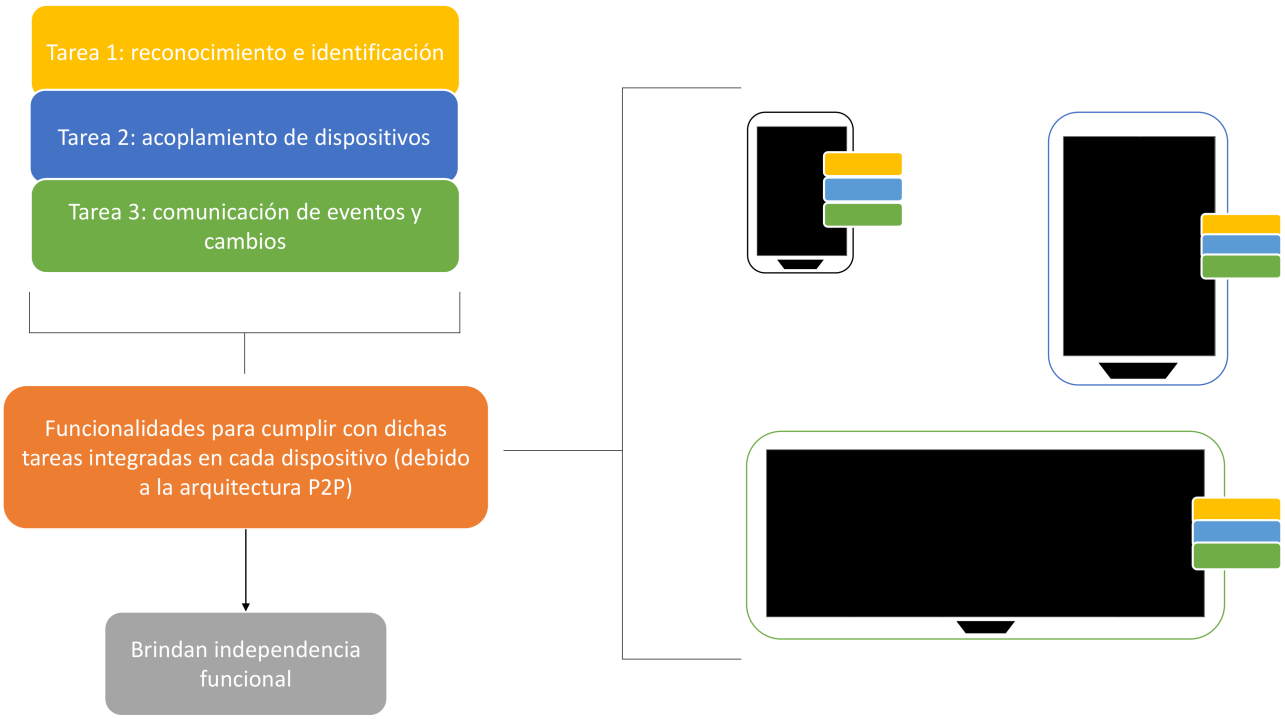

Figura 3.1: Independencia funcional de cada dispositivo

3.2.1. Servicio de descubrimiento de dispositivos

Concretamente, este servicio se encarga de recolectar los datos de identificación de otros dispositivos presentes en la misma red, así como de distribuir los propios. Así mismo, se encarga de verificar que se encuentren disponibles aquellos dispositivos que ya han sido encontrados e identificados. Esta verificación es necesaria debido al carácter volátil de ecosistemas tecnológicos cuya conectividad depende por completo de una o varias redes inalámbricas. Debido a su comportamiento, este servicio se puede considerar como un administrador de listas de presencia.

3.2.2. Servicio de control de acoplamiento

Este servicio tiene como propósito la creación de sesiones colaborativas al combinar dos o más espacios de trabajo individuales en un único espacio de trabajo compartido a través de gestos de acoplamiento. Debido a la arquitectura utilizada, es necesario que cada uno de los integrantes de una sesión colaborativa almacene localmente una copia de dicho espacio (figura 3.2). Posterior a un gesto de acoplamiento exitoso (y por consiguiente, de la creación de una sesión colaborativa), este servicio se encarga de enviar y recibir las características de las pantallas de los dispositivos, así como de los sensores, adaptadores de red y funcionalidades del sistema operativo de cada dispositivo integrante de la nueva sesión a (y desde) cada una de las entidades acopladas.

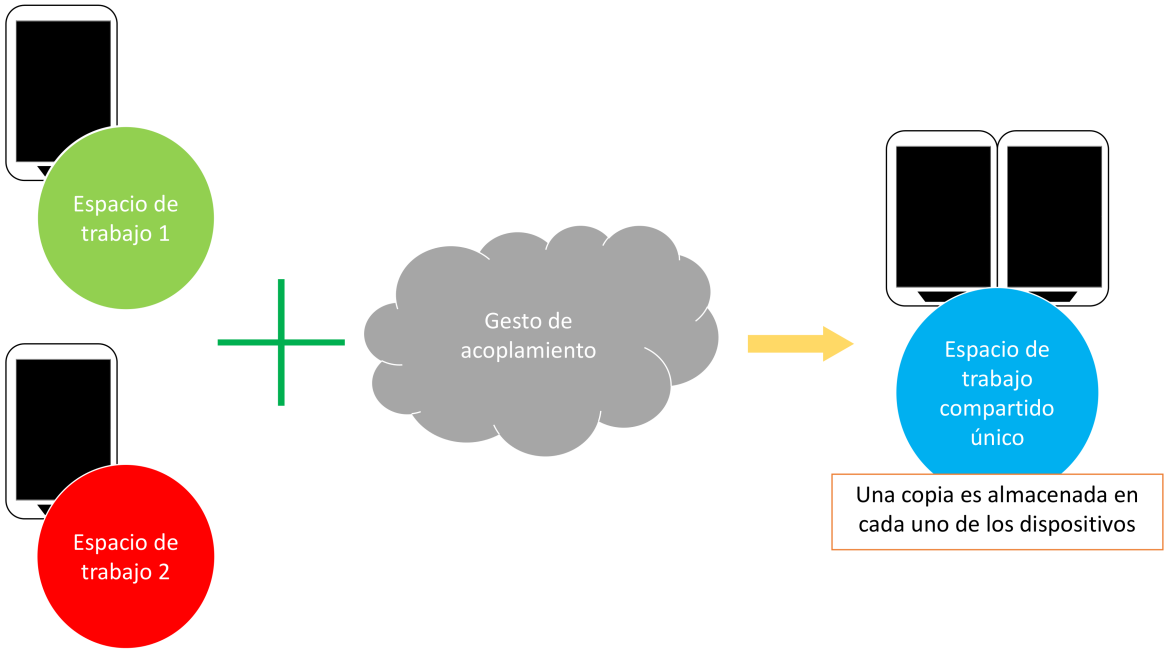

Figura 3.2: Formación de sesiones colaborativas con espacios compartidos de trabajo a través de gestos de acoplamiento

Como se mencionó en la sección 2.4.2, es posible determinar el lado por el cual se desea acoplar un dispositivo con otro a través de gestos de acoplamiento que puedan ser contextualizados. El

soporte desarrollado proporciona dos gestos que cumplen con esta característica y cuya implementación les permite ser utilizados inmediatamente:

1. **Gesto de arrastre concurrente sobre las pantallas de ambos dispositivos** (perteneciente a la categoría definida en la sección 2.4.2); consiste básicamente en arrastrar un dedo sobre cada una de las pantallas. Los gestos de arrastre deben ser en direcciones opuestas, de tal forma que se denoten los lados por los cuales se desea unir los espacios de trabajo (figura 3.3).

2. **Gesto de inclinación de los dispositivos en direcciones opuestas** (perteneciente a la categoría definida en la sección 2.4.1); se deben inclinar al menos en un cierto ángulo durante un tiempo determinado, de tal forma que los lados por los cuales se acoplarán los dispositivos permanezcan sobre la superficie (figura 3.4).

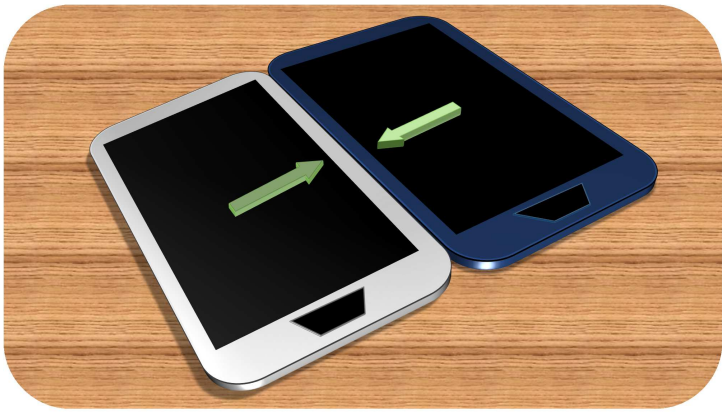

Figura 3.3: Gesto de arrastre

Figura 3.4: Gesto de inclinación

Dichos gestos de acoplamiento son parte de las interfaces ubicuas, las cuales actúan como percutores de todo el proceso de **adaptación** (ver sección 2.2) que la aplicación final lleva a cabo como parte de la transición entre un modo de trabajo y otro. Como se mencionó anteriormente,

una de las ventajas que conlleva la utilización de gestos para el acoplamiento de dispositivos, es el aprovechamiento de los recursos que dichos dispositivos ofrecen. Las pantallas son uno de los recursos más importantes y representan un medio tangible sobre el cual actúa el proceso de **redistribución**, e.g., después de acoplar dos o más dispositivos, se podrían mostrar diferentes secciones del espacio de trabajo compartido en cada una de las pantallas de los dispositivos acoplados, lo cual implica que la nueva distribución habilita indirectamente varias interfaces de usuario en varios dispositivos para poder interactuar con el mismo espacio de trabajo.

Debido a la heterogeneidad de los dispositivos soportados, es posible que las características de algunos dispositivos no sean las más adecuadas para que puedan hacer uso de ambos gestos de acoplamiento. Es por ello que el soporte desarrollado es lo suficientemente flexible como para permitir que los dispositivos procesen estos gestos de forma indistinta (figura 3.5), i.e., los usuarios son libres de utilizar el gesto de acoplamiento que les resulte más cómodo o intuitivo, dependiendo de las condiciones en las cuales se esté llevando a cabo la interacción o las características de los dispositivos. Adicionalmente, se hace uso de háptica y sonidos audibles cuando un intento de acoplamiento ha comenzado y cuando éste ha sido exitoso. Finalmente, los desarrolladores tienen la posibilidad de definir e implementar sus propios gestos de acoplamiento.

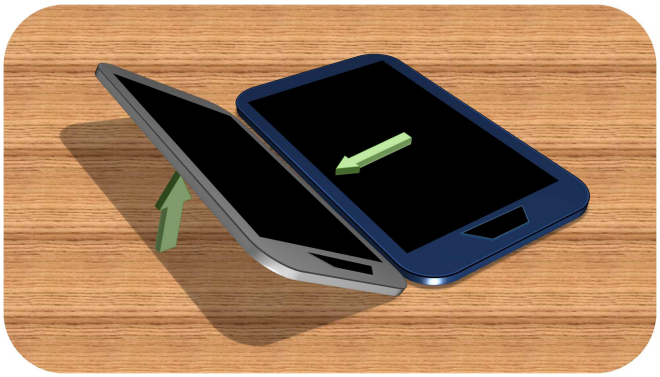

Figura 3.5: Gestos combinados

Un dispositivo (concretamente, la aplicación que está siendo ejecutada por dicho dispositivo) puede operar únicamente bajo un solo modo de trabajo en un determinado momento. Dependiendo de los modos de trabajo bajo los cuales se encontraban operando dos dispositivos recién acoplados, la sesión colaborativa resultante puede:

- **Ser creada**: para el caso en el que ninguno de los dispositivos involucrados fuera integrante de una sesión colaborativa antes del acoplamiento, i.e., ninguno de los dispositivos había sido acoplado anteriormente.
- **Ser combinada**: sucede cuando ambos dispositivos ya pertenecían a una sesión colaborativa distinta, resultando en la combinación de dichas sesiones, i.e., ambos dispositivos ya habían sido acoplados anteriormente.
- **Ser aumentada**: cuando se une un espacio de trabajo individual al espacio de trabajo compartido de una sesión colaborativa existente, i.e., solamente uno de los dispositivos ya había sido acoplado anteriormente.

3.2.3. Servicio de distribución de actualizaciones

Finalmente, el servicio de distribución de actualizaciones se encarga de enviar y recibir (hacia y desde cada uno de los dispositivos miembros de una sesión colaborativa) aquellos mensajes que no estén relacionados con las tareas de comunicación que llevan a cabo los otros dos servicios. Esto quiere decir que dichos mensajes pueden ser tan simples o complejos como el modelo de tareas de una aplicación que se encuentra operando bajo el modo de trabajo colaborativo lo requiera, e.g., cambios en el contenido del espacio de trabajo, cambios en el espacio de trabajo mismo o respuestas a algún tipo de petición. Este servicio puede ser activado de dos formas distintas (figura 3.6):

1. **Explícita**: cuando se envía o recibe algún mensaje de algún dispositivo miembro de la sesión colaborativa o cuando se crea, modifica o elimina uno o más objetos virtuales pertenecientes al espacio de trabajo compartido.

2. **Implícita**: sucede cuando la activación del servicio se da como resultado de un acoplamiento de dispositivos, e.g., difusión del nuevo conjunto de objetos virtuales después de acoplar dos dispositivos cuyos espacios de trabajo ya contenían uno o más objetos. El servicio de control de acoplamiento será quien le comunique dicho evento.

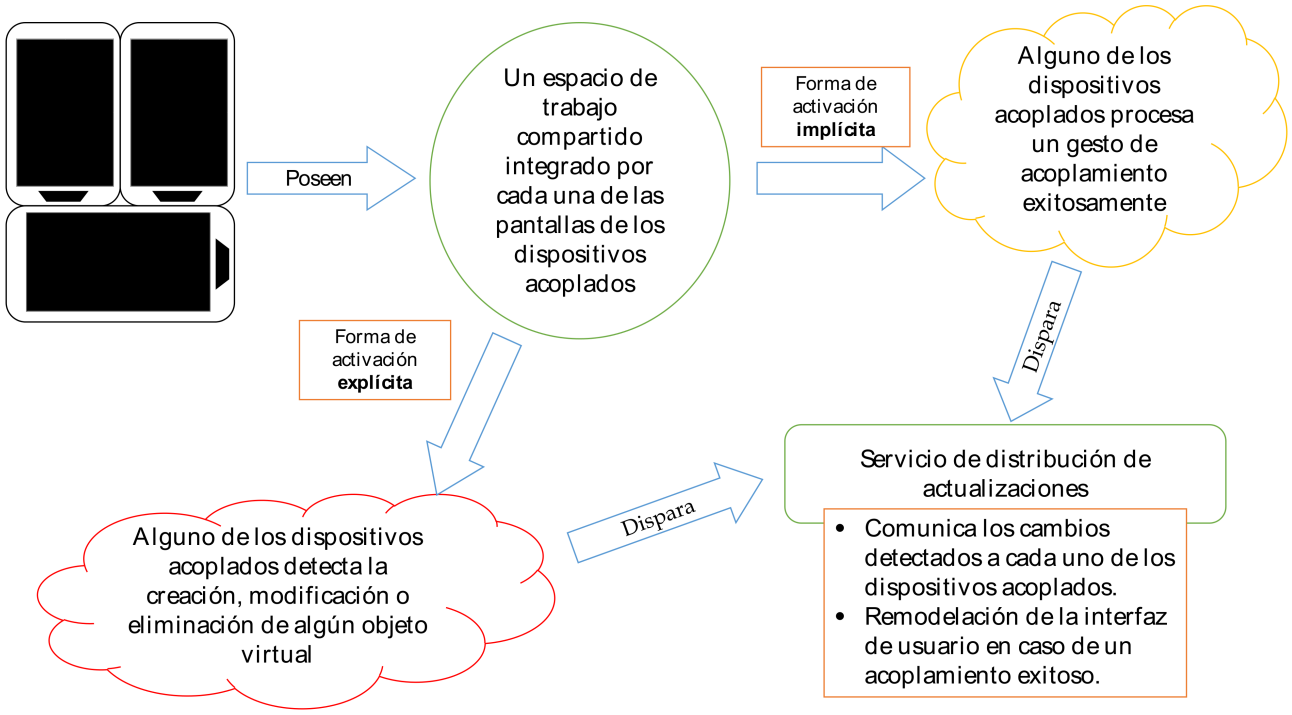

Figura 3.6: Formas de activación del servicio de distribución de actualizaciones

Es posible considerar a este servicio como el responsable de la **remodelación** de las interfaces gráficas de usuario, ya que es quien recibe actualizaciones sobre los objetos contenidos en el espacio de trabajo compartido y, por consiguiente, quien adecúa la representación visual de acuerdo al modelo de tareas de la aplicación final.

3.2.4. Relaciones entre los servicios y una aplicación

A pesar de que la división de tareas (mencionada en el inicio de la sección 3.2) permite a cada uno de los servicios comportarse de forma totalmente independiente, la idea principal consiste en utilizarlos de forma conjunta para lograr las transiciones deseadas entre un tipo de trabajo y otro. En la figura 3.7 se muestran gráficamente las relaciones que existen entre los servicios descritos y ciertos eventos relevantes dentro de un modelo de tareas típico de una aplicación móvil.

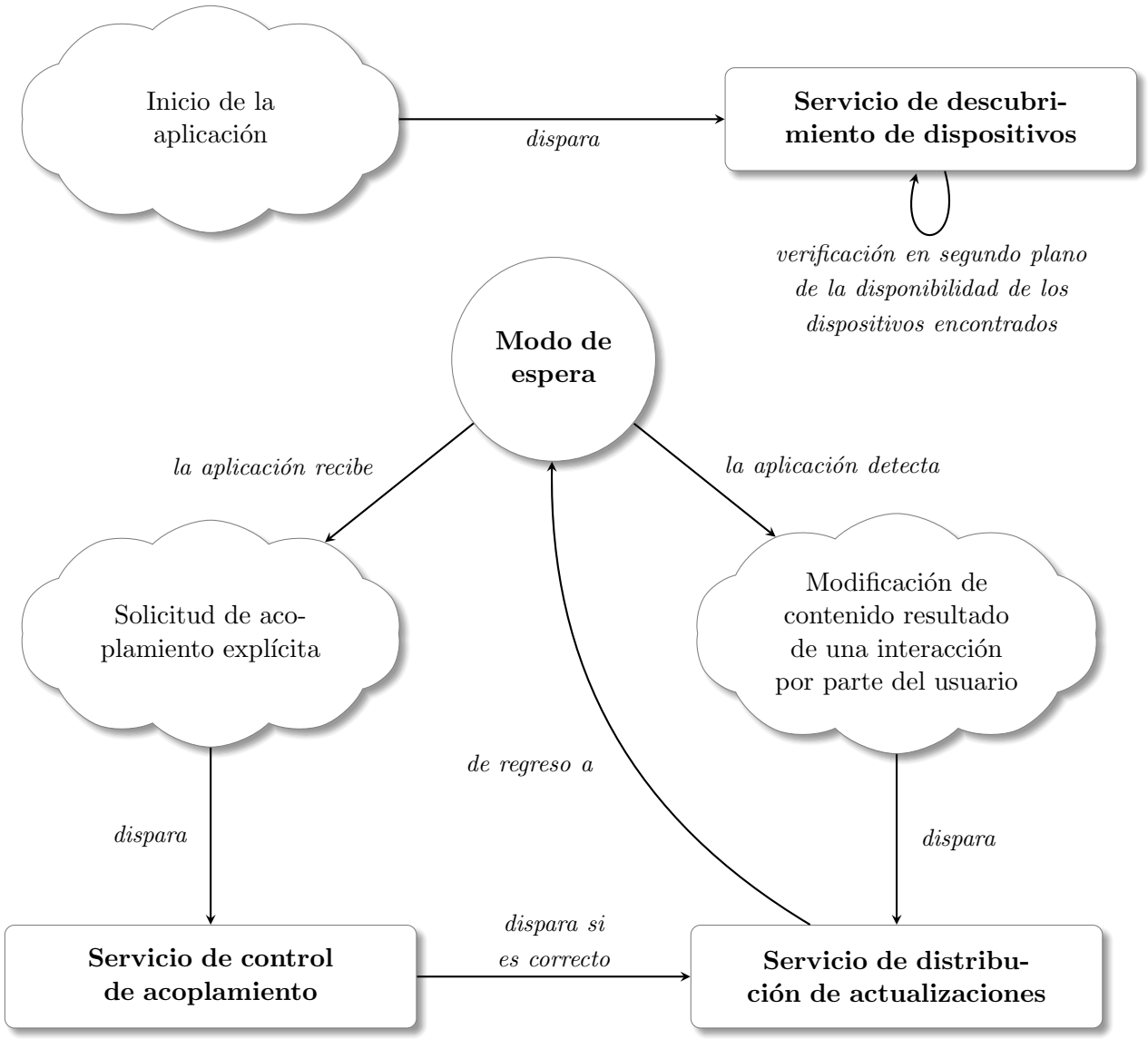

Figura 3.7: Vista general de las relaciones que guardan los servicios implementados

El primer servicio en ejecutarse siempre será el servicio de descubrimiento de dispositivos. Dicho servicio es disparado inmediatamente después de que el usuario abre la aplicación. De esta forma se brinda al dispositivo la posibilidad de determinar parte del **modelo del ambiente físico** en

donde se llevarán a cabo las interacciones colaborativas (una de las tres variables contextuales mencionadas en la sección 2.1). Por otro lado, los servicios de control de acoplamiento y distribución de actualizaciones se encuentran disponibles y en modo de espera hasta que la aplicación detecte algún evento que los requiera.

El primer punto de contacto entre el modo de trabajo colaborativo y el soporte propuesto es el servicio de control de acoplamiento, ya que éste es quien atiende la intención de colaborar por parte de dos o más usuarios. Posterior a un acoplamiento exitoso, el servicio de control de acoplamiento vuelve a estar disponible para futuros intentos y comunica al servicio de distribución de actualizaciones que debe enviar el nuevo espacio de trabajo compartido (concretamente, los objetos que contiene) a los otros integrantes de la sesión. Esto implica que la relación que guardan estos servicios es permanente, ya que el primero siempre dependerá del segundo para llevar a cabo dichas tareas de actualización.

La información referente al **modelo de la plataforma** es extraída y comunicada, a través del servicio de control de acoplamiento, a los dispositivos después de un intento de acoplamiento. Finalmente, la idea de utilizar una arquitectura P2P radica en que cualquier usuario pueda utilizar cualquier dispositivo acoplado de forma indistinta y en cualquier momento (siempre que los aspectos sociales lo permitan). Es por ello que no es necesario determinar el **modelo del usuario**.

3.3. Problemas de diseño

A lo largo de las fases iniciales de análisis de la presente tesis, se abordaron diversos problemas de diseño en múltiples ámbitos. A continuación se describen aquellos problemas que tuvieron un impacto directo en el diseño del soporte.

3.3.1. Gestos de acoplamiento propuestos inicialmente

En un principio se estableció como uno de los objetivos específicos de la presente tesis, que la formación del espacio de trabajo compartido debería basarse en gestos intuitivos y flexibles. Debido a que es indispensable hacer partícipes a los usuarios finales en etapas tempranas del desarrollo de un sistema de este tipo, se aplicó una encuesta en la cual se les solicitó su opinión para decidir cuáles son los gestos de acoplamiento que les parecen más prácticos o intuitivos (ver sección 6.1). En la encuesta se agregaron dos gestos, sin embargo, los encuestados eran libres de dar su opinión al respecto o proponer gestos alternativos. Los resultados arrojaron que la mayoría prefiere que los dos gestos propuestos en la encuesta estén disponibles en el soporte. Dichos gestos se describen a continuación:

1. Desplazar ambos dispositivos en direcciones opuestas con un movimiento rectilíneo hasta que dos de su lados hagan contacto.

2. Con un dedo sobre la pantalla de cada dispositivo, realizar un gesto concurrente en línea recta que indique cuáles serán los lados por los cuales se acoplarán los dispositivos.

La primera opción requiere la utilización de acelerómetros para poder obtener las mediciones de cada uno de los sensores en los dispositivos y así procesarlas para determinar en qué dirección se

desplazaron los mismos. Después de realizar algunas pruebas iniciales, recolectar datos de distintos acelerómetros y analizarlos, se determinó que el primer gesto de acoplamiento en realidad no es una buena opción si se desea cumplir con el objetivo específico mencionado, por las siguientes razones:

1. Dado que se pretende dar soporte a una gama heterogénea de dispositivos, las pruebas con modelos que no son tan recientes (i.e., aquellos que no soportan una versión del sistema operativo Android superior a la 2.3 y que se lanzaron al mercado aproximadamente en el 2010) arrojaron que los datos sensados por los acelerómetros no eran tan precisos, comparados con sensores de fábrica en modelos recientes. Esto se debe a que el movimiento no es lo suficientemente representativo como para que sea detectado por sensores menos sensibles.

2. Como consecuencia del punto anterior, se requeriría arrastrar algunos dispositivos con una mayor fuerza que otros, lo que eventualmente podría conllevar a una pérdida significativa de flexibilidad e incluso daños en las partes posteriores y laterales de los dispositivos. Podríamos tomar como ejemplo un intento de acoplamiento entre dos dispositivos, cuyos acelerómetros tengan sensibilidades distintas: aquel cuyo sensor requiera que se imprima mayor fuerza al momento de realizar un desplazamiento (para poder obtener lecturas significativas) provocará automáticamente una falta de precisión y consistencia en dicho intento de acoplamiento, debido a que ambos dispositivos tendrían que moverse a velocidades distintas. Esta situación podría confundir a los usuarios y conllevaría a fallos constantes durante los intentos de acoplamiento.

Como una alternativa a estas desventajas, se propuso cambiar el gesto de acoplamiento descrito por otro que perteneciera a la misma categoría (ver capítulo 2, sección 2.4.1), pero cuyo gesto sea significativo y más flexible (ver sección 3.2). Debido a que este gesto depende del estado físico de los dispositivos en vez de movimientos que pueden variar constantemente entre un intento y otro (muy difíciles de repetir), las lecturas del acelerómetro son más constantes y confiables, por lo cual se aumenta en gran medida la taza de éxito de los intentos de acoplamiento.

3.3.2. Tamaño del grupo de usuarios

Los resultados obtenidos en [Lucero et al., 2010] plantean que el uso de espacios de trabajo compartidos en arreglos de dispositivos móviles colocados sobre una superficie (todos en reposo), carga consigo una desventaja física que podría limitar la usabilidad de este tipo de sistemas. La razón se debe a que la mayoría de los usuarios utiliza su mano dominante para interactuar con su dispositivo y la no-dominante para sostenerlo, bloqueando así la vista de los otros usuarios que se encuentren, ya sea observando o interactuando con el sistema. Para evitar este tipo de inconvenientes, se propone limitar el grupo colaborativo a cuatro usuarios, sin restringir el número de dispositivos que se pueden utilizar para desplegar el espacio de trabajo.

3.3.3. Conectividad

Cumpliendo con el aspecto de diseño de **conectividad** descrito en la sección 3.1, es indispensable que el mecanismo de comunicación entre dispositivos sea lo suficientemente robusto como

para habilitar el envío de información entre múltiples dispositivos en un momento determinado y cuando los usuarios lo requieran. La transmisión de dicha información se lleva a cabo a través de redes WiFi únicamente, ya que Bluetooth cuenta con una velocidad de transmisión de datos inferior a la que ofrece la especificación de WiFi [IEEE, 2014]. Adicionalmente, Bluetooth restringe el número de dispositivos conectados a un grupo de aproximadamente siete usuarios [Bluetooth-SIG, 2014] (dependiendo de las especificaciones de los adaptadores instalados en los dispositivos, aunque en la práctica este número puede disminuir a cuatro). De esta forma, utilizar redes WiFi implica que el número máximo de dispositivos que podrán estar acoplados en un momento determinado (si es que no está limitado por el modelo de tareas de la aplicación final) dependerá del número de clientes que puedan estar conectados a una misma red al mismo tiempo.

Por otro lado, la congruencia de la información en cada uno de los dispositivos y la velocidad de transmisión de los mensajes son algunas de las características más importantes que se buscaron conservar, resultando en un factor fundamental para decidir el tipo de protocolos de comunicación a utilizar. Debido a las características de los servicios descritos en la sección 3.2, resultó indispensable hacer uso de una combinación de conexiones TCP y UDP para aprovechar las ventajas que ofrecen ambos protocolos de comunicación [IETF, 1981]. Específicamente, el protocolo UDP se utiliza para difusiones sin remitentes (ver capítulo 4, secciones 4.2.1 y 4.2.2) y TCP para la transmisión segura de información (e.g., informar a otros dispositivos acerca de algún evento en particular, como la modificación o creación de objetos virtuales).

3.3.4. Geometría

Para solucionar el aspecto de diseño de **geometría** espacial entre los dispositivos, tuvimos la necesidad de determinar la posición relativa de uno o varios dispositivos sin la utilización de hardware adicional. Es por ello que se optó por hacer uso de gestos de acoplamiento, independientemente de la categoría a la que pertenecieran. Se realizó una investigación con el objetivo de encontrar gestos que cumplieran con los siguientes requisitos:

1. Que no requieran hardware adicional.
2. Que sean consistentes, i.e., que siempre que sean contextualizados, se obtengan los mismos resultados (o lo más similares posibles).
3. Que permitan señalar o denotar de alguna forma, uno y solamente uno de los lados del dispositivo, independientemente de su orientación.
4. Que sean fácilmente reproducibles.
5. Que sean fáciles de utilizar.

Para cumplir con el 5to requisito, realizamos una encuesta a un grupo de usuarios finales con las siguientes opciones: la utilización de los gestos de acoplamiento descritos en la sección 3.3.1 de forma individual y en conjunto. Adicionalmente, se les dio la posibilidad de sugerir algún otro gesto que les pareciera interesante o intuitivo. Los resultados arrojaron que la mayoría prefirió que estuvieran disponibles ambos gestos de acoplamiento, razón por la cual se decidió implementar dos gestos que no interfirieran el uno con el otro.

3.3.5. Detección de un gesto de acoplamiento

Otro de los problemas presentados en fases iniciales de desarrollo fue diferenciar un gesto de acoplamiento (específicamente, alguno que perteneciera a la categoría descrita en el capítulo 2, sección 2.4.2) de gestos regulares sobre las pantallas que pudieran utilizarse para tareas comunes, tales como el desplazamiento de contenido (*scroll*) o la interacción con objetos que requieran algún tipo de arrastre sobre la pantalla. Es sumamente importante que los gestos se puedan llevar a cabo con total libertad, pero sin resultar intrusivos para el modelo de tareas de las aplicaciones que hagan uso del soporte. Para solucionar este problema, se implementó un detector de clicks largos, el cual puede ser adherido a cualquier objeto que tenga la capacidad de procesar eventos de tipo *touch* (e.g. *views* o *surfaceviews*). De esta forma, basta con realizar un click largo sobre el objeto deseado para indicarle al dispositivo que estamos a punto de llevar a cabo un gesto de acoplamiento.

3.4. Estructura y comportamiento del soporte

En esta sección se describe la estructura interna del soporte y la organización de cada uno de sus componentes, todo esto a través de una serie de diagramas cuyo propósito es ilustrar de forma más específica, la arquitectura y funcionalidades descritas en las secciones anteriores.

3.4.1. Diagrama de paquetes

En la figura 3.8 se muestra el diagrama de paquetes del soporte. El paquete principal, *gerbera*, está compuesto por otros cuatro paquetes lógicamente relacionados, cuyo contenido engloba cada uno de los elementos que permiten el funcionamiento de los servicios que ofrece el soporte. Dicha organización se describe a continuación:

- *services*: contiene la lógica para configurar, iniciar, utilizar y terminar los servicios de descubrimiento de dispositivos, control de acoplamiento y distribución de actualizaciones. Está compuesto por los paquetes *devicediscovery*, *couplingcontrol* y *updatedistribution*.

- *gestures*: contiene las configuraciones y algoritmos que utilizan los gestos de acoplamiento (descritos en la sección 3.2.2) cuyas definiciones se encuentran en los paquetes *pinchgesture* y *tiltgesture*. Es importante recalcar que quienes hagan uso del soporte, deberían definir e implementar sus gestos (si es que lo requieren) dentro de este paquete. Aunque no es un requerimiento estricto, sería deseable respetar la organización propuesta para mantener una mejor claridad y escalabilidad del soporte, ya que de esta forma los desarrolladores pueden ubicar fácilmente los nuevos gestos para consecuentes modificaciones.

- *global*: en este paquete están agrupados todos los recursos compartidos por cada uno de los tres servicios, tales como el contenedor de objetos virtuales que representa el espacio de trabajo y las listas de control de los servicios de descubrimiento de dispositivos y control de acoplamiento. Así mismo, en este paquete se encuentran definidas las funcionalidades necesarias para evitar problemas de concurrencia en dichos recursos.

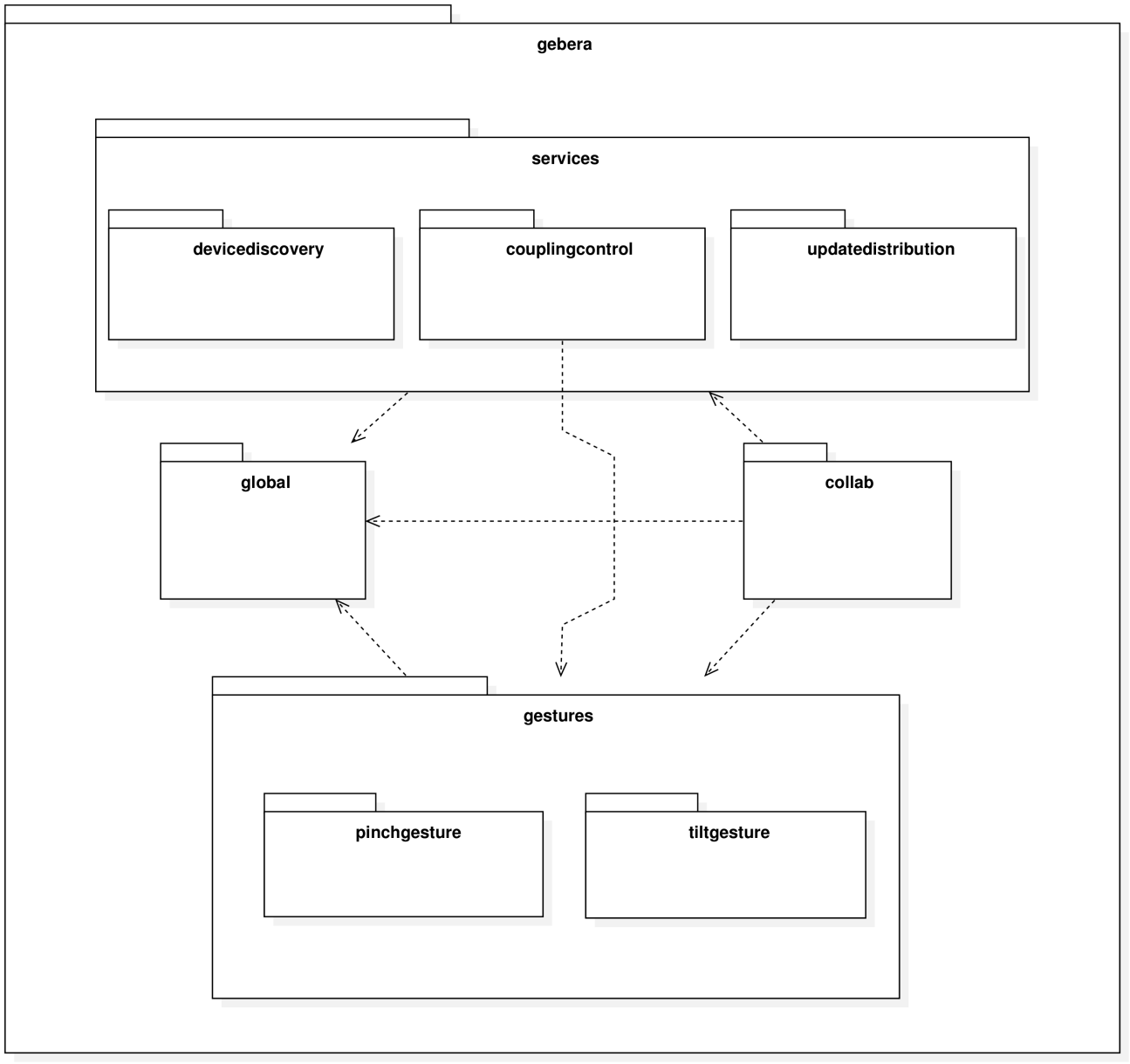

Figura 3.8: Diagrama de paquetes del soporte.

- *collab*: finalmente, en este paquete se proporcionan dos clases auxiliares, mediante las cuales se pretende facilitar la implementación del soporte propuesto, al brindar un ejemplo funcional y una clase que demuestra la configuración básica del mismo. Aunque dicha clase proporciona una configuración específica para cada uno de los servicios, es posible ajustar cada uno de los parámetros para que se adecúen al modelo de tareas de una aplicación.

3.4.2. Diagramas de clases

A continuación, se presentan los diagramas de cada una de las clases que conforman los paquetes mencionados en la sección 3.4.1, así como las relaciones que guardan entre sí.

En primer lugar se muestra el contenido del paquete *services.devicediscovery* en la figura 3.9. Dicho paquete contiene la implementación del servicio de descubrimiento de dispositivos y está compuesto por los siguientes elementos:

- Clase **DiscoveryPublishService**: publica periódicamente los siguientes datos de identificación básica: IP, puertos TCP y UDP, así como el estado actual del dispositivo, i.e. si se encuentra disponible o si está a punto de abandonar la aplicación. Cuando el dispositivo está por abandonar la aplicación, esta clase difunde una serie de mensajes para alertar a las otras entidades que ya no estará disponible.

- Clase **DiscoveryCollectService**: recolecta la información difundida a través de la red y que fue producida por otros dispositivos mediante **DiscoveryPublishService**.

- Clase **DiscoverySettings**: aquí se almacenan las variables personalizables de las dos clases anteriores, tales como: el puerto a través del cual se recibirá la información difundida por **DiscoveryPublishService** desde otros dispositivos, el tiempo que espera dicha clase entre una difusión de datos y otra, el número de mensajes de abandono difundidos al salir de la aplicación, así como el tiempo que **DiscoveryPublishService** debe esperar entre un envío de dichos mensajes y los siguientes.

- Interfaz **DiscoveryServiceListener**: la clase **DiscoveryCollectService** hace uso de los métodos de esta interfaz a modo de *callbacks*, para indicarle a la aplicación qué hacer cuando se detecta que un dispositivo externo tiene en funcionamiento el servicio de descubrimiento (i.e., cuando dicho dispositivo tiene una instancia de la clase **DiscoveryPublishService** propagando sus datos de identificación o cuando está por abandonar la aplicación).

Cabe mencionar que **DiscoveryPublishService** y **DiscoveryCollectService** extienden la clase **AsyncTask**, a diferencia de las clases que componen a los otros servicios del soporte. Las razones del por qué utilizar un recurso u otro se discuten en el capítulo 4, sección 4.3.

Posteriormente, el diagrama de clases del paquete *services.couplingcontrol* se muestra en la figura 3.10, el cual contiene la implementación del servicio de control de acoplamiento. Las clases e interfaz que lo componen se describen a continuación:

- Clase **LongClickDetector**: es el punto de origen de un intento de acoplamiento dentro del soporte, ya que esta clase es la encargada de detectar un click largo para proceder a procesar un gesto de acoplamiento (ver sección 3.3.5).

- Clase **SendOnePairingRequest**: se encarga de difundir un intento de acoplamiento, el cual está compuesto por la IP del propio dispositivo, los puertos UDP y TCP por los cuales se pueden recibir mensajes u objetos de la red, el lado por el cual se pretende acoplar con otro dispositivo y las especificaciones y funcionalidades especiales que posee un dispositivo.

- Clase **ListenOnePairingRequest**: recibe y procesa un intento de acoplamiento difundido por **SendOnePairingRequest** desde otro dispositivo. Un intento de acoplamiento se considera exitoso cuando el lado recibido, como parte de un intento externo, es contrario al lado difundido a través del intento propio. Como resultado, se crea una sesión colaborativa, se anexa el dispositivo a una sesión existente o se combinan dos sesiones en una sola (ver sección 3.2.2).

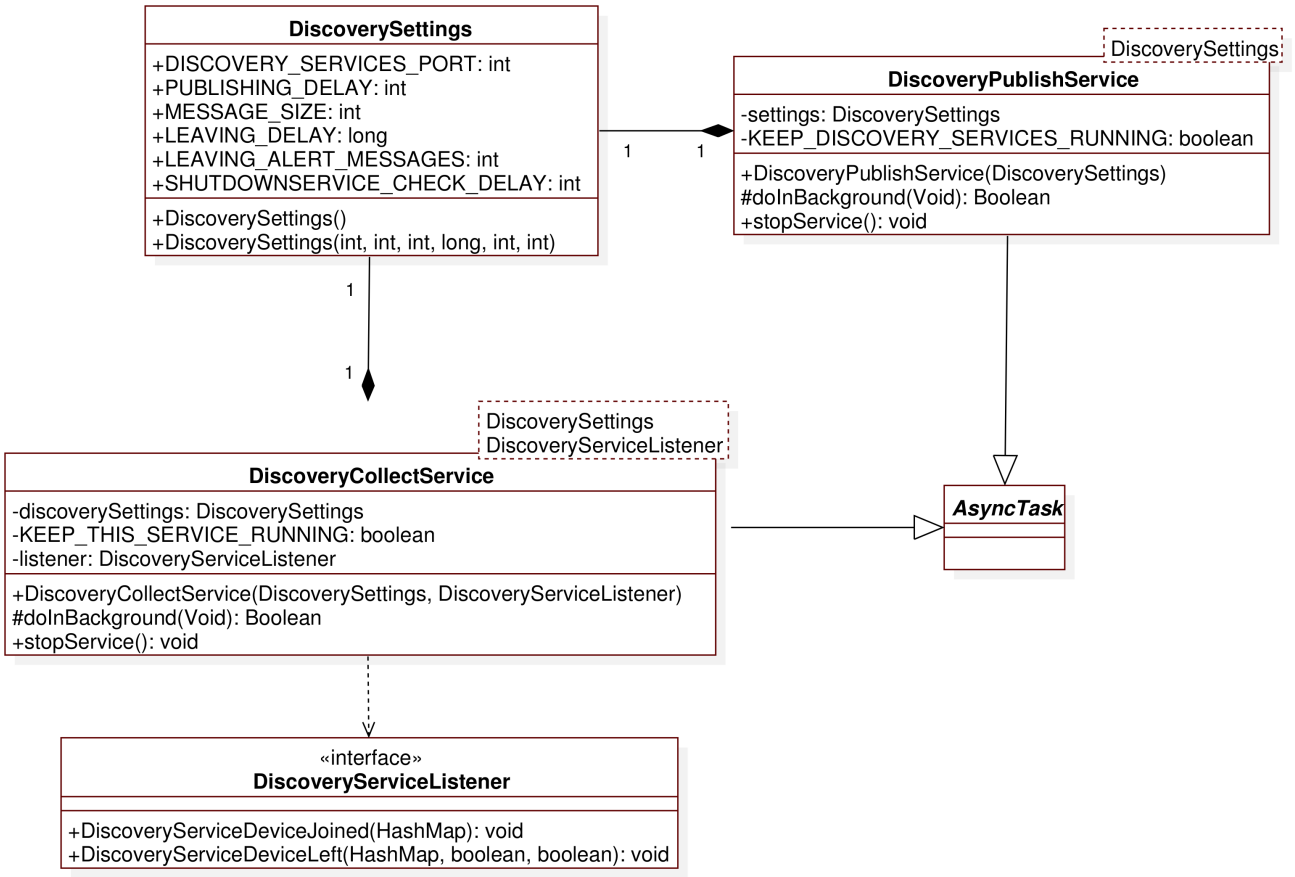

Figura 3.9: Diagrama de clases del paquete *devicediscovery*.

- Clase **SendFeaturesToTheGroup**: tiene como tarea difundir, a los integrantes de la sesión colaborativa, las especificaciones y funcionalidades del dispositivo cuando sucede algún evento que cambie alguno de dichos elementos (e.g., una rotación del dispositivo produciría el intercambio de los valores del ancho y alto de la pantalla, por lo cual es indispensible informar a los otros dispositivos que las especificaciones han cambiado para que puedan reaccionar apropiadamente).

- Clase **CouplingSettings**: almacena las variables que afectan la configuración de cada una de las clases mencionadas, tales como el puerto en donde se recibirán los intentos de acoplamiento, el tiempo de espera por un intento de acoplamiento, etc.

- Interfaz **WorkspaceTouchEventListener**: debido a que la clase **LongClickDetector** procesa los eventos de tipo *touch* detectados sobre la superficie de trabajo, la interfaz **WorkspaceTouchEventListener** proporciona un método que funciona como *callback* para que los desarrolladores puedan indicar al soporte qué hacer con dichos eventos y cómo procesarlos.

En la figura 3.11 se muestra el diagrama de clases del paquete *services.updatedistribution* (implementación del servicio de distribución de actualizaciones). Dicho paquete está compuesto por dos clases principales, dos clases internas y una interfaz, los cuales se describen a continuación:

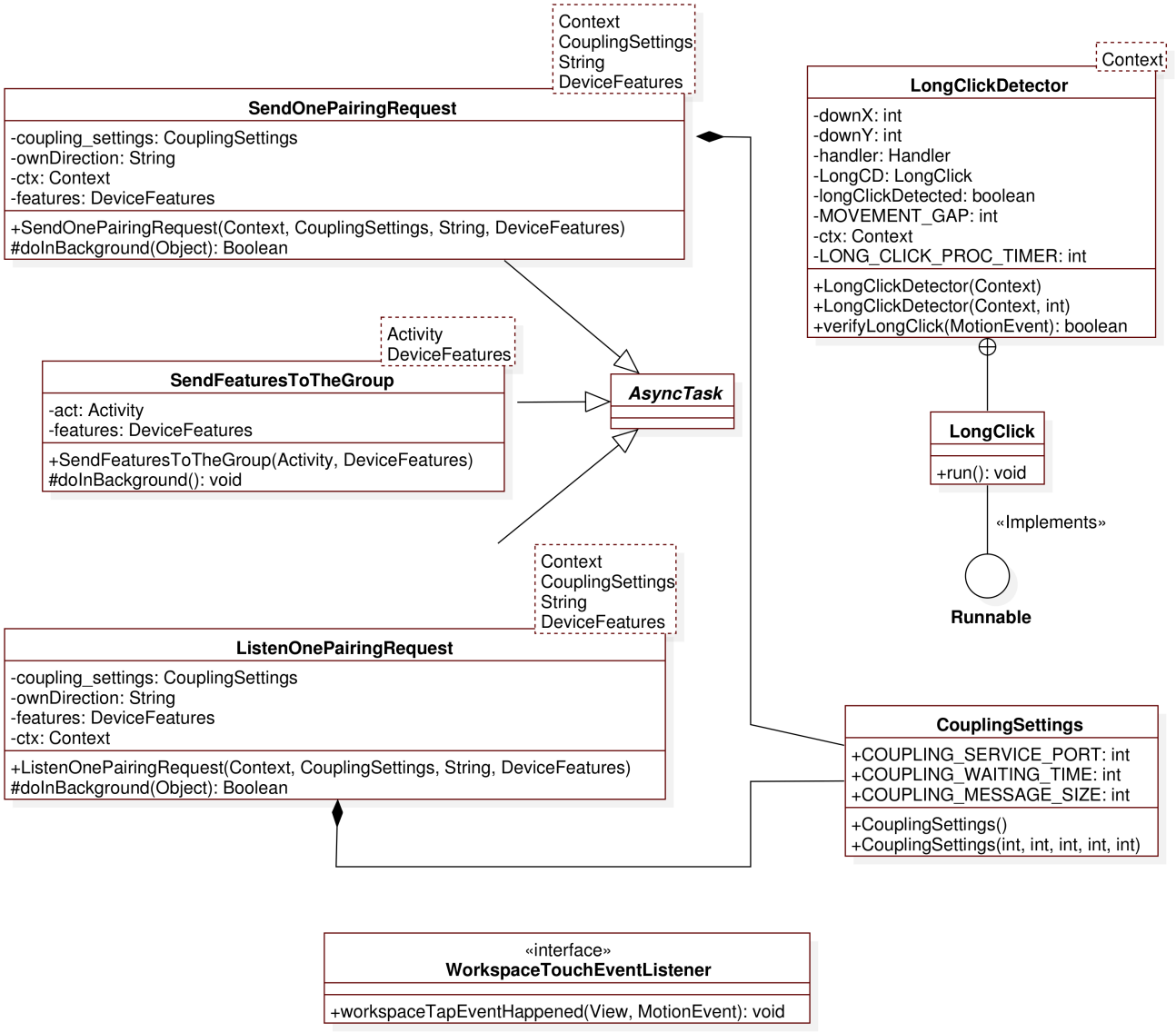

Figura 3.10: Diagrama de clases del paquete *couplingcontrol*.

- Clase **UpdateDistributionService**: es la clase de control del servicio, cuyo objetivo consiste básicamente en arrancar o detener este servicio.
- Clase privada **UpdateDistributionTCPServer**: esta clase está contenida en **UpdateDistributionService** y se encarga de procesar los mensajes (concretamente, objetos) recibidos que forman parte del acoplamiento de dispositivos, de la propagación de características enviadas a través de la clase **SendFeaturesToTheGroup** y de cualquier otro objeto que forme parte del modelo de tareas de la aplicación.
- Clase privada **WorkerThread**: esta clase está contenida en **UpdateDistributionService** y se encarga de enviar a uno o todos los integrantes de una sesión colaborativa, uno o varios objetos. Aunque esta clase está marcada como privada, es posible enviar objetos mediante los métodos *spreadObject(...)* y *sendObject(...)*.

- Clase **UpdateSettings**: contiene los puertos TCP y UDP disponibles para este servicio.
- Interfaz **UpdateDistributionServiceListener**: proporciona la función que opera como *callback* para la clase **UpdateDistributionTCPServer**, la cual se activa al momento de recibir un objeto (enviado mediante los métodos *spreadObject(...)* y *sendObject(...)*) desde otros dispositivos.

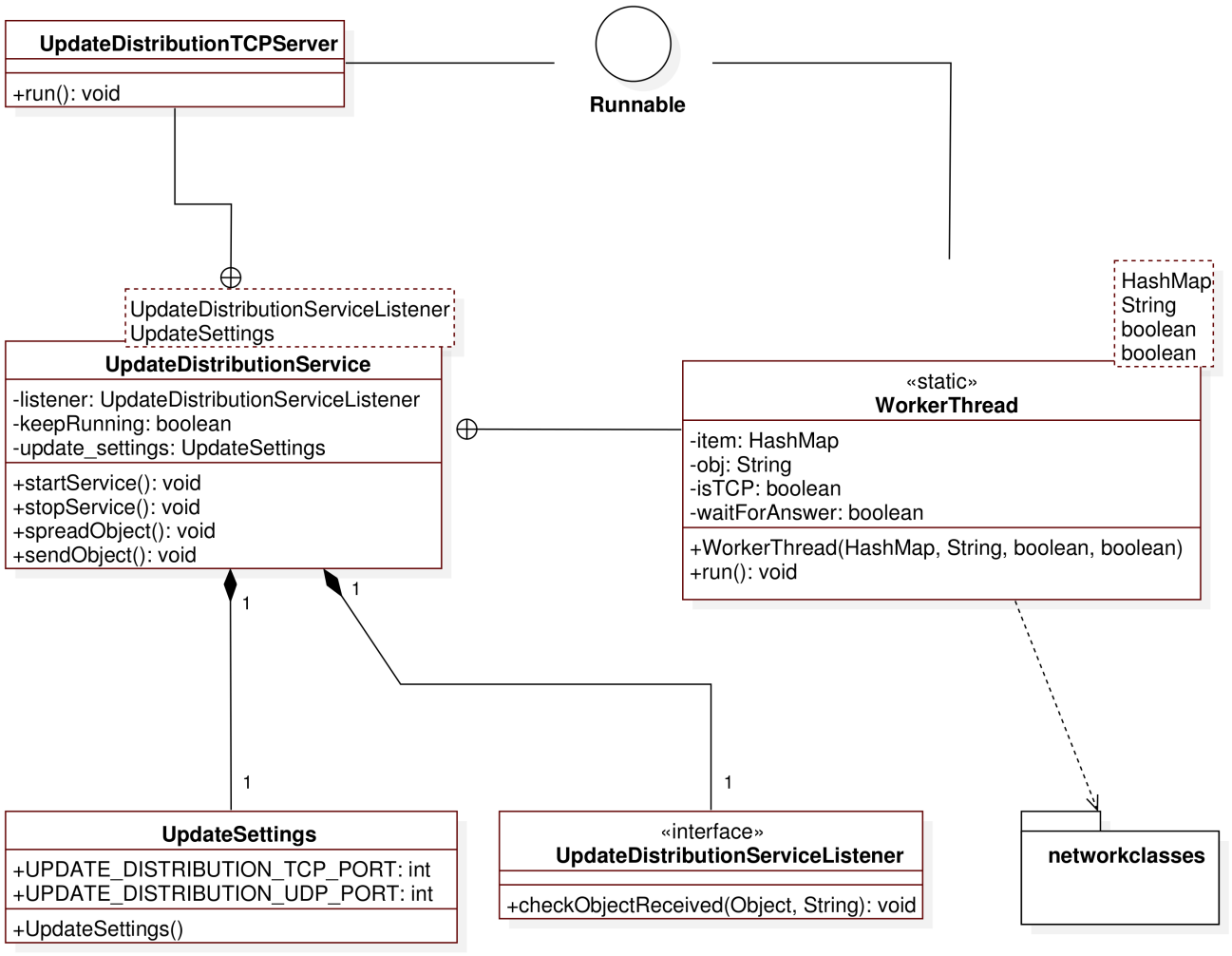

Figura 3.11: Diagrama de clases del paquete *updatedistribution*.

Adicionalmente, este paquete contiene otro llamado *services.updatedistribution.networkclasses*, en el cual se encuentran las clases cuyos objetos podrá procesar (i.e., enviar y recibir) el servidor **UpdateDistributionTCPServer** (ver capítulo 4, sección 4.2.3).

El cuarto diagrama de clases (figura 3.12) corresponde al paquete *gestures*. En dicho diagrama se muestra cómo están compuestas las clases que habilitan los gestos de acoplamiento implementados:

- Clase **SimplePinchGesture**: se encarga de verificar el gesto de arrastre sobre la pantalla de un dispositivo. Puede determinar el lado seleccionado dependiendo de la distancia me-

dida entre el punto de inicio (cuando se coloca un dedo sobre la pantalla) y el punto final (cuando se levanta el dedo). Los datos recibidos con los cuales se contextualiza el gesto (i.e., los datos con los cuales se puede determinar hacia qué lado se arrastró el dedo) provienen del mismo objeto al cual se encuentra adherido el detector de clicks largos definido en la clase **LongClickDetector**.

- Clase **SimpleAccelerometerGesture**: contiene la implementación del gesto de inclinación, el cual permite seleccionar algún lado del dispositivo para el proceso de acoplamiento, dependiendo de la inclinación constante del dispositivo durante un tiempo determinado. Cabe mencionar que este gesto se puede configurar para adaptarlo de mejor forma al ambiente en donde será utilizado o a los requerimientos de la aplicación, ya que es posible modificar el tiempo de muestreo del acelerómetro (i.e., duración del gesto) y la inclinación mínima requerida (proporción del valor de la gravedad que se acepta como mínima permitida).

Como ya se mencionó antes, los gestos implementados pueden ser fácilmente contextualizados, con el objetivo de determinar la intención de un usuario por seleccionar uno de los cuatro lados de un dispositivo. Esto significa que dichos gestos son utilizados como medio de verificación para determinar la intención de un usuario por iniciar o integrarse a una sesión colaborativa, a pesar de que un gesto como tal pueda tener múltiples significados y utilidades (medios de autenticación segura, emparejamiento de dispositivos, etc.). De esta forma, es posible utilizar ambos gestos en diferentes contextos y con otros fines, razón por la cual los gestos de acoplamiento y el servicio de control de acoplamiento se encuentran en diferentes paquetes.

Finalmente, el diagrama de clases del paquete *global* se muestra en la figura 3.13. Dicho paquete está compuesto por tres clases, sin embargo describimos únicamente las dos más representativas:

- Clase **SharedResources**: almacena los recursos compartidos más importantes del soporte. Dichos recursos son utilizados por cada uno de los servicios y clases ya mencionadas, razón por la cual deben ser inicializados antes que cualquier otro servicio. Debido a la concurrencia sucitada en tiempo de ejecución, estos recursos cuentan con soporte de control de concurrencia.

- Clase **DeviceFeatures**: se encarga de obtener y almacenar las características y funcionalidades de un dispositivo. Las clases **SendOnePairingRequest** y **SendFeaturesToTheGroup** hacen uso de objetos pertenecientes a esta clase durante su flujo de operaciones normal.

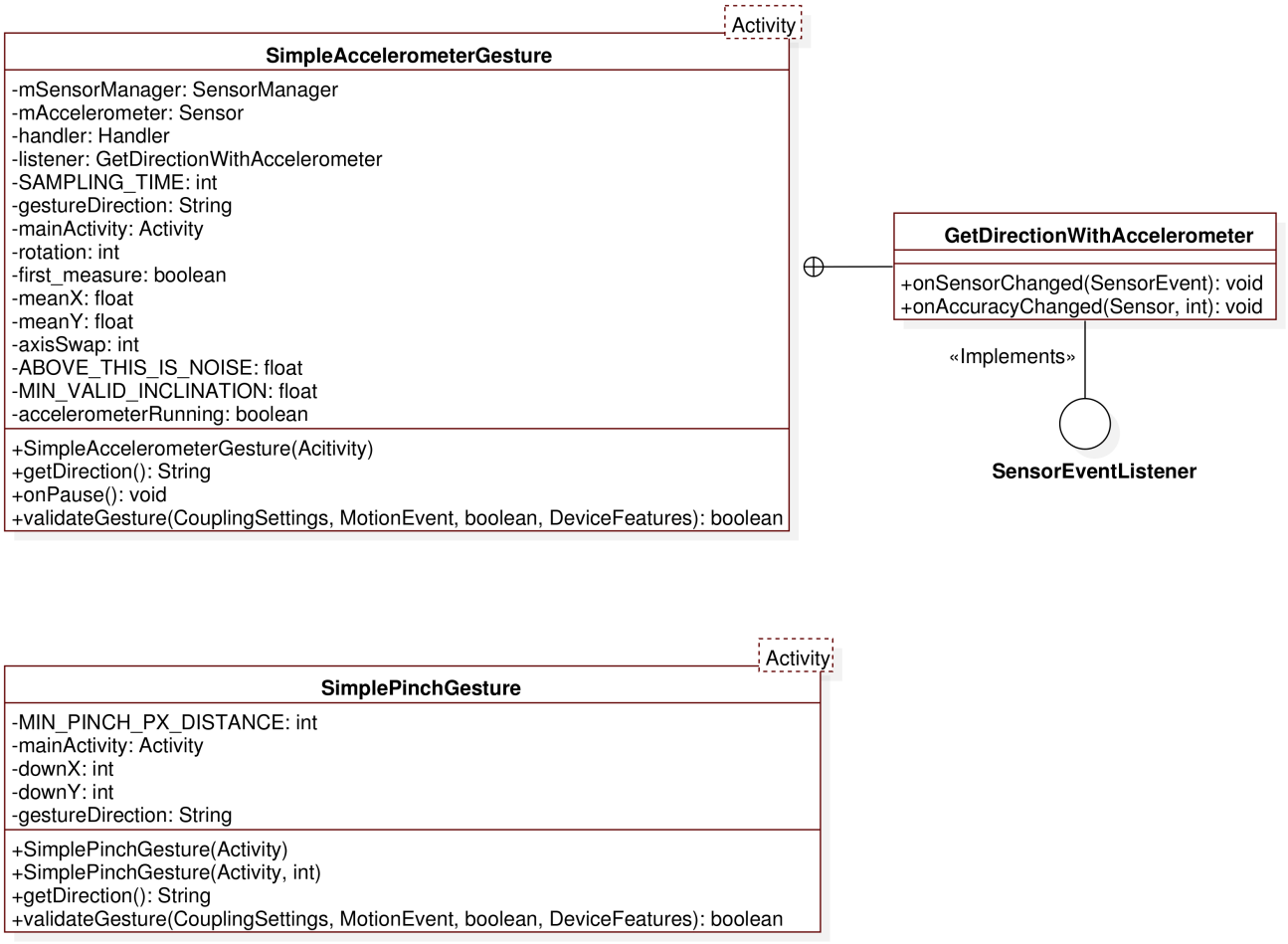

Figura 3.12: Diagrama de clases del paquete *gestures*.

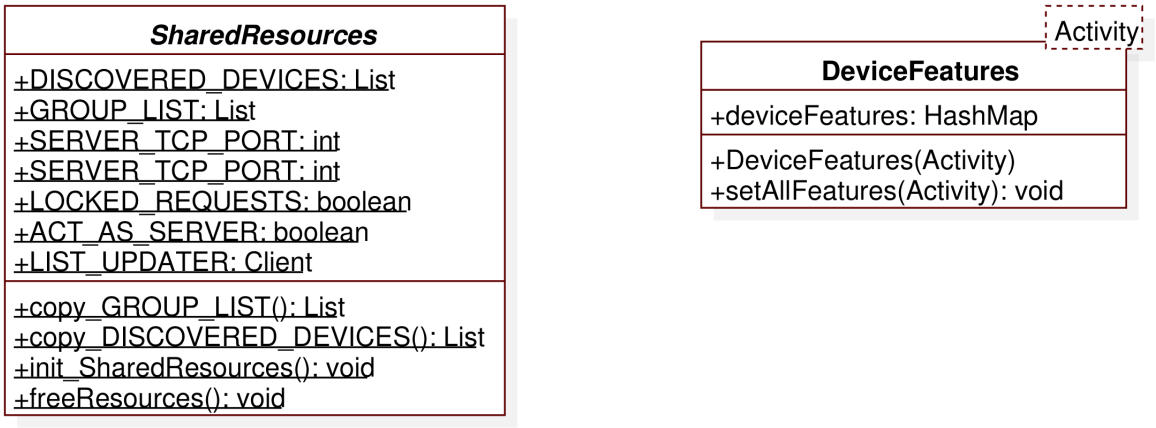

Figura 3.13: Diagrama de clases del paquete *global*.

3.4.3. Diagramas de casos de uso

A continuación se presentan los diagramas de casos de uso generales y específicos más representativos, junto con sus respectivos formatos extendidos. Dichos diagramas guardan una estrecha relación con las definiciones mostradas gráficamente en la secciones anteriores.

Tabla 3.1: Caso de uso extendido: recolectar información de identificación de otros dispositivos.

Caso de uso (figura 3.14)	Recolectar información de identificación de otros dispositivos.
Actores	Servicio de descubrimiento de dispositivos.
Propósito	Permitir conocer en todo momento qué dispositivos se encuentran presentes y disponibles en la red.
Pre-condiciones	Haber iniciado la aplicación y estar conectado a una red inalámbrica (WiFi).

Tabla 3.2: Caso de uso extendido: difundir periódicamente información técnica de identificación.

Caso de uso (figura 3.14)	Difundir periódicamente información técnica de identificación.
Actores	Servicio de descubrimiento de dispositivos.
Propósito	Anunciar la presencia y disponibilidad de un dispositivo a cada una de las entidades conectadas en la red.
Pre-condiciones	Haber iniciado la aplicación y estar conectado a una red inalámbrica (WiFi).

Tabla 3.3: Caso de uso extendido: difundir abandono de la aplicación.

Caso de uso (figura 3.14)	Difundir abandono de la aplicación.
Actores	Servicio de descubrimiento de dispositivos.
Propósito	Difundir el abandono de la aplicación para que otros dispositivos estén conscientes de dicho evento.
Pre-condiciones	Estar conectado a una red inalámbrica (WiFi).

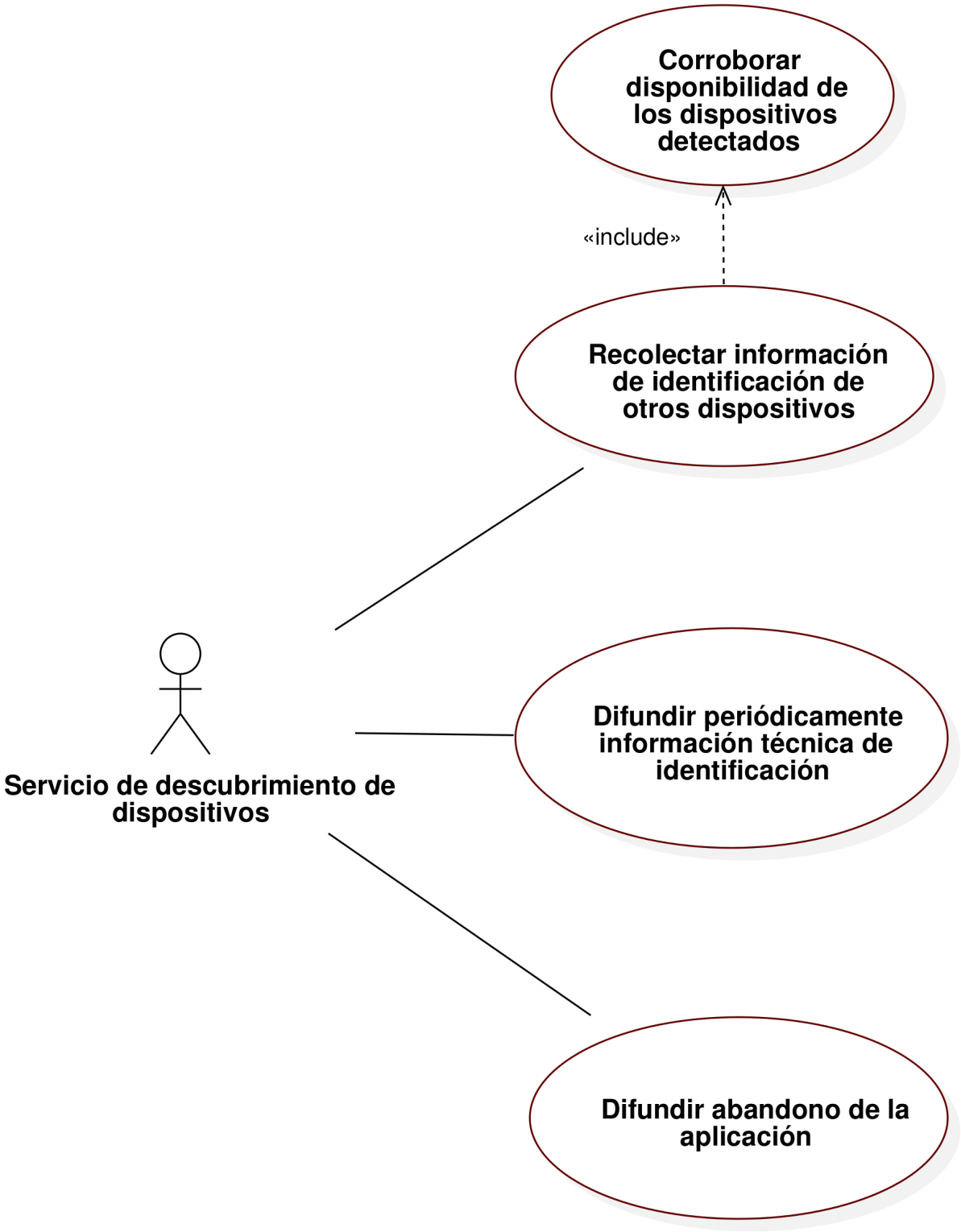

Figura 3.14: Diagrama de casos de uso del servicio de descubrimiento.

Tabla 3.4: Caso de uso extendido: enviar y recibir intentos de acoplamiento.

Caso de uso (figura 3.15)	Enviar y recibir intentos de acoplamiento.
Actores	Servicio de control de acoplamiento.
Propósito	Permitir el acoplamiento de dispositivos a través de intentos de acoplamiento difundidos por la red.
Pre-condiciones	Estar conectado en la misma red inalámbrica (wifi) que los otros dispositivos y haber procesado un gesto de acoplamiento correctamente.

Tabla 3.5: Caso de uso extendido: aceptar y procesar gestos de acoplamiento.

Caso de uso (figura 3.15)	Aceptar y procesar gestos de acoplamiento.
Actores	Servicio de control de acoplamiento.
Propósito	Habilitar un dispositivo para aceptar y procesar un gesto de acoplamiento de uno o varios gestos predefinidos.
Pre-condiciones	Que el dispositivo cuente con los sensores adecuados para procesar dichos gestos.

Tabla 3.6: Caso de uso extendido: controlar sesiones colaborativas.

Caso de uso (figura 3.15)	Controlar sesiones colaborativas.
Actores	Servicio de control de acoplamiento.
Propósito	Administrar la creación, eliminación y unión de sesiones colaborativas.
Pre-condiciones	Estar conectado en la misma red inalámbrica (wifi) que los otros dispositivos y haber procesado un gesto de acoplamiento correctamente.

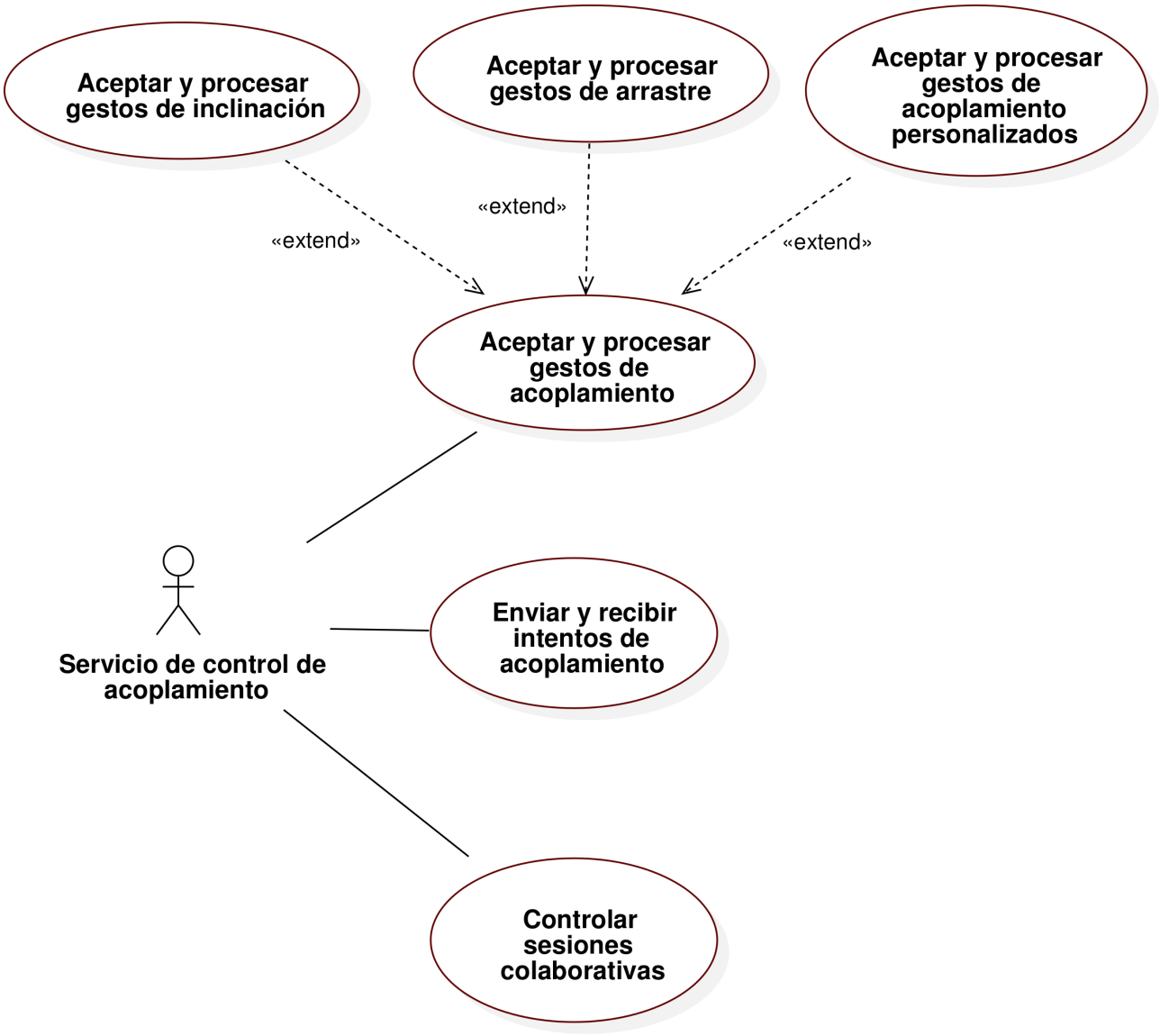

Figura 3.15: Diagrama de casos de uso del servicio de control de acoplamiento.

Tabla 3.7: Caso de uso extendido: comunicar y recibir cambios en el espacio de trabajo compartido.

Caso de uso (figura 3.16)	Comunicar y recibir cambios en el espacio de trabajo compartido.
Actores	Servicio de distribución de actualizaciones.
Propósito	Actualizar el espacio de trabajo compartido desplegado en un dispositivo al recibir actualizaciones por parte de otros colaboradores. Así mismo, enviar las actualizaciones sobre dicho espacio de trabajo al resto de los integrantes de la actual sesión colaborativa.
Pre-condiciones	Pertenecer a una sesión colaborativa.

Tabla 3.8: Caso de uso extendido: enviar y recibir objetos virtuales.

Caso de uso (figura 3.16)	Enviar y recibir objetos virtuales.
Actores	Servicio de distribución de actualizaciones.
Propósito	Actualizar el contenido del espacio de trabajo compartido desplegado en un dispositivo al recibir actualizaciones por parte de otros colaboradores. Así mismo, enviar las actualizaciones sobre dicho contenido al resto de los integrantes de la actual sesión colaborativa.
Pre-condiciones	Pertenecer a una sesión colaborativa.

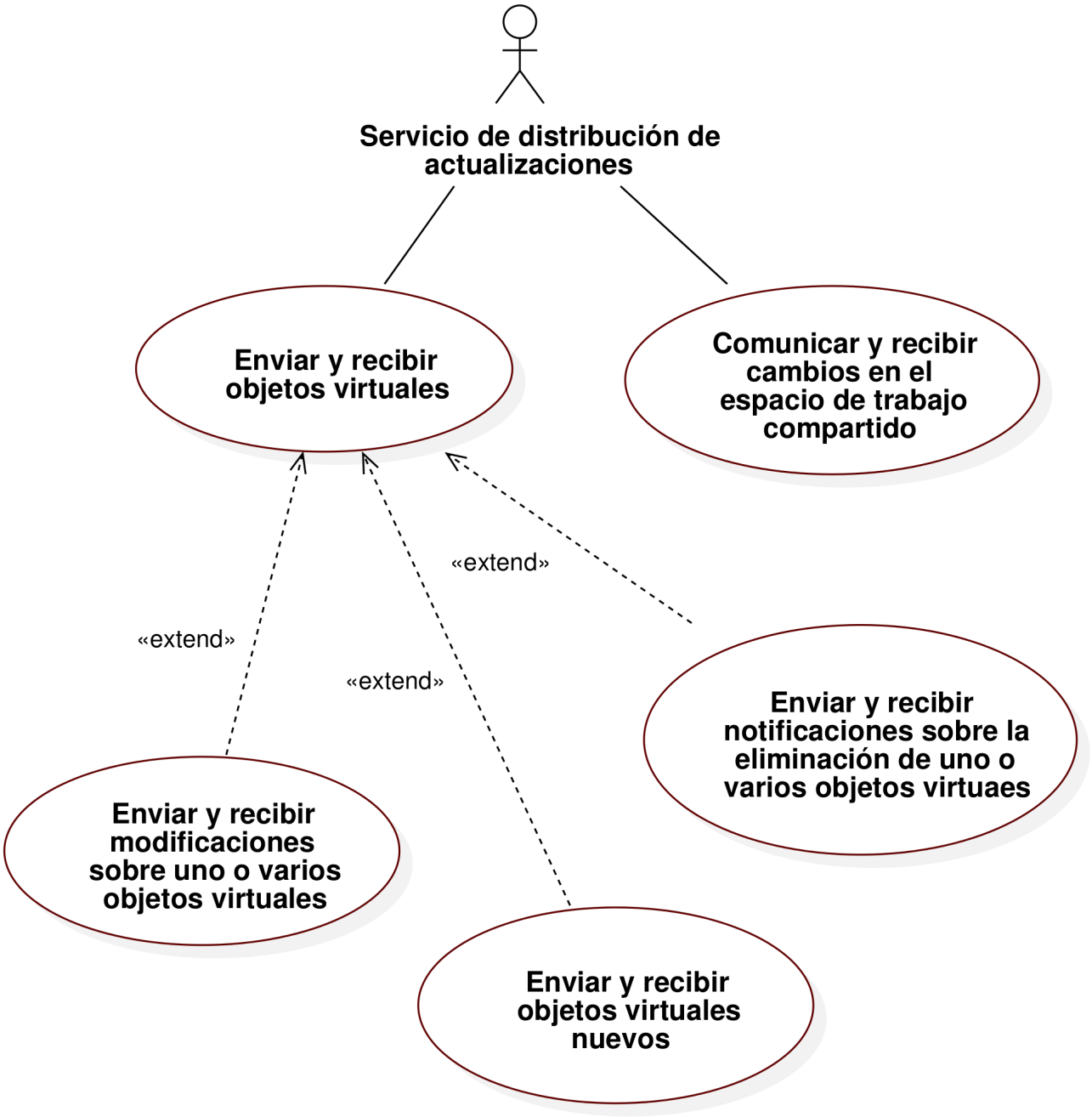

Figura 3.16: Diagrama de casos de uso del servicio de distribución de actualizaciones.

Tabla 3.9: Caso de uso extendido: realizar un gesto de acoplamiento.

Caso de uso (figura 3.17)	Realizar un gesto de acoplamiento.
Actores	Usuario.
Propósito	Habilitar un dispositivo para aceptar y procesar un gesto de acoplamiento de uno o varios gestos predefinidos.
Pre-condiciones	Ninguna.

Tabla 3.10: Caso de uso extendido: realizar un gesto de acoplamiento.

Caso de uso (figura 3.17)	Realizar un gesto de acoplamiento.
Actores	Usuario.
Propósito	Realizar gestos especiales (gestos de acoplamiento) sobre los dispositivos con el objetivo de acoplar dos o más dispositivos.
Pre-condiciones	Haber definido, configurado y habilitado los gestos de acoplamiento que estarán disponibles.

Tabla 3.11: Caso de uso extendido: interactuar con el espacio de trabajo.

Caso de uso (figura 3.17)	Interactuar con el espacio de trabajo.
Actores	Usuario.
Propósito	Interactuar con el espacio de trabajo o el contenido del mismo.
Pre-condiciones	Ninguna.

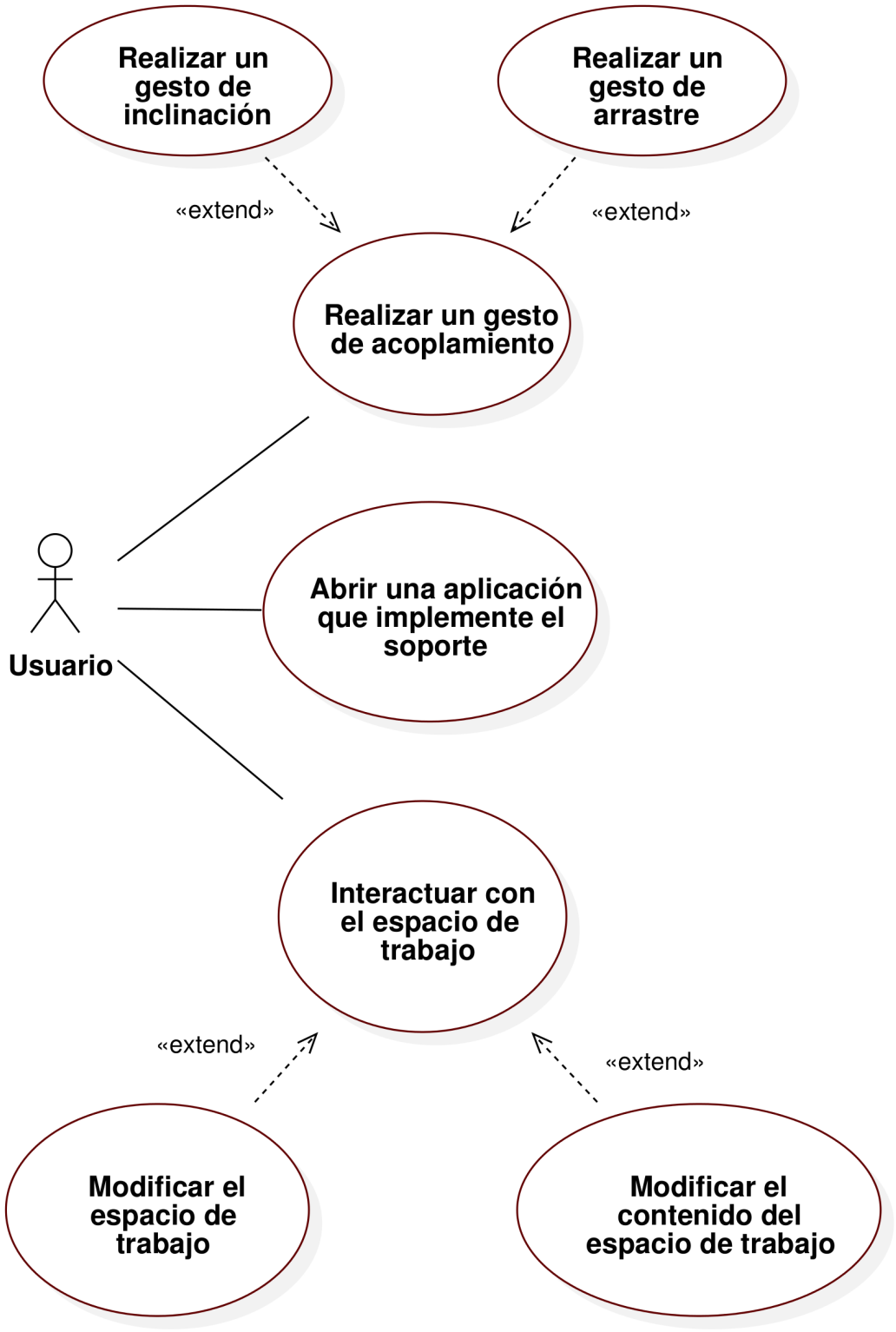

Figura 3.17: Diagrama de casos de uso general (perspectiva del usuario).

Tabla 3.12: Caso de uso extendido: anunciar presencia.

Caso de uso (figura 3.18)	Anunciar presencia.
Actores	Dispositivo.
Propósito	Anunciar la disponibilidad del dispositivo para entablar sesiones colaborativas.
Pre-condiciones	Estar conectado en la misma red inalámbrica (wifi) que los otros dispositivos.

Tabla 3.13: Caso de uso extendido: detectar entidades presentes en la red.

Caso de uso (figura 3.18)	Detectar entidades presentes en la red.
Actores	Dispositivo.
Propósito	Detectar la presencia y disponibilidad de otros dispositivos conectados a la misma red.
Pre-condiciones	Estar conectado a una red inalámbrica (wifi).

Tabla 3.14: Caso de uso extendido: realizar acoplamientos con otros dispositivos.

Caso de uso (figura 3.18)	Realizar acoplamientos con otros dispositivos.
Actores	Dispositivo.
Propósito	Permitir el acoplamiento de dispositivos mediante gestos especiales para la creación de espacios de trabajo compartidos.
Pre-condiciones	Estar conectado en la misma red inalámbrica (wifi) que los otros dispositivos y contar con los sensores adecuados para procesar los gestos de acoplamiento predefinidos.

Tabla 3.15: Caso de uso extendido: difundir actualizaciones y cambios.

Caso de uso (figura 3.18)	Difundir actualizaciones y cambios.
Actores	Dispositivo.
Propósito	Comunicar a todos los integrantes de una sesión colaborativa los cambios que se detecten en cuanto al espacio de trabajo compartido o su contenido.
Pre-condiciones	Pertenecer a una sesión colaborativa después de participar en un intento de acoplamiento exitoso.

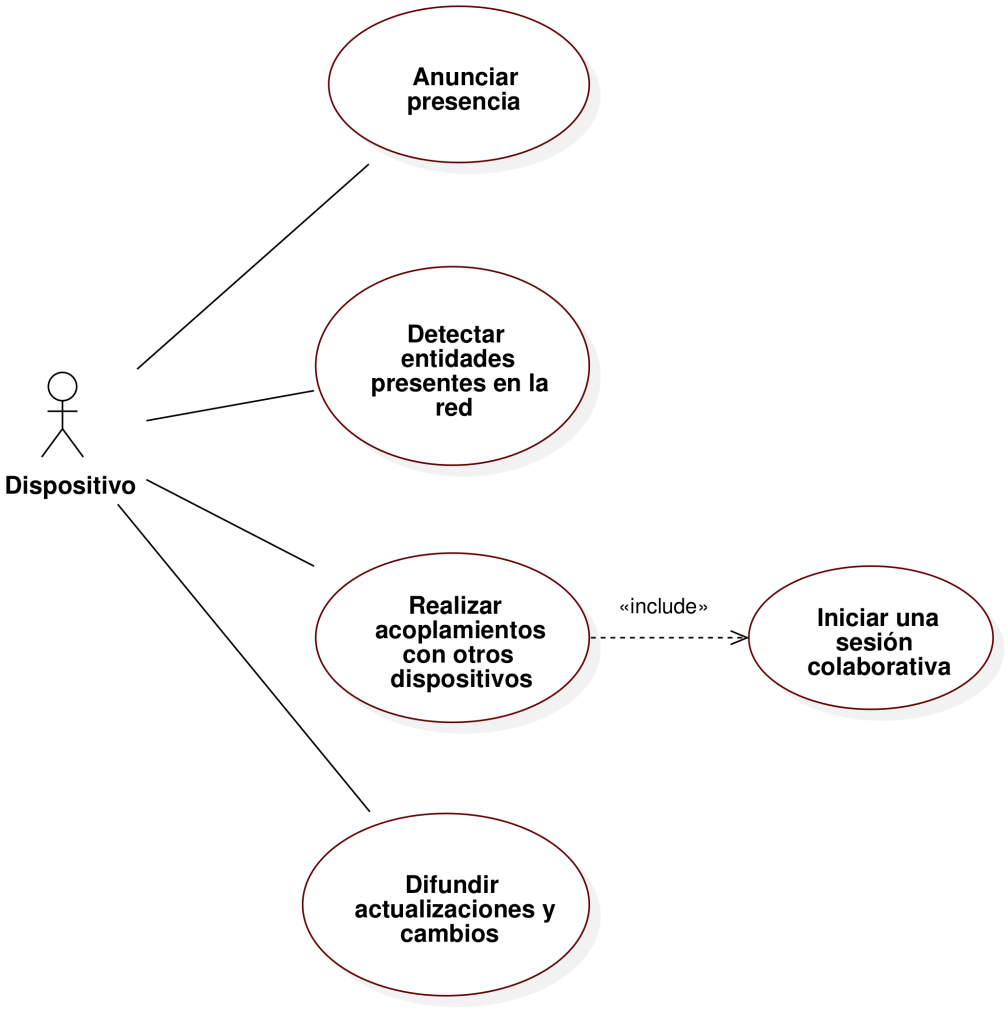

Figura 3.18: Diagrama de casos de uso general (perspectiva del dispositivo).

Capítulo 4

Implementación del soporte

En este capítulo se detalla la implementación de cada uno de los servicios descritos en el capítulo anterior en un nivel más técnico, partiendo desde los mecanismos de comunicación por red y las herramientas utilizadas para el despliegue de las interfaces gráficas, hasta los algoritmos utilizados para: controlar y procesar los gestos de acoplamiento, seleccionar los dispositivos que participarán en el proceso de acoplamiento, propagar las actualizaciones correspondientes a cada uno de los dispositivos que forman parte de una sesión colaborativa, cómo se mantienen activas dichas sesiones después de rotar uno o más dispositivos, entre otros más.

4.1. Metodología de desarrollo

Debido a la naturaleza del soporte propuesto, se requirió de un modelo de desarrollo incremental para poder comprobar la efectividad de los avances que se fueron desarrollando a lo largo de la presente tesis, en cada una de sus fases. Una de las ventajas más resaltables, como producto de la utilización de este modelo de desarrollo, radicó en la posibilidad de efectuar modificaciones en uno o varios servicios del soporte, sin afectar todo el sistema en general.

A continuación se describe el modelo de desarrollo incremental y el paradigma de programación utilizado.

4.1.1. Modelo de desarrollo incremental

El modelo incremental combina elementos del modelo lineal secuencial con la filosofía interactiva de construcción de prototipos. El modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario de actividades (cada secuencia lineal produce un incremento del software entregable) [Pressman, 2002].

Cuando se utiliza un modelo incremental, el primer incremento a menudo es un producto esencial, i.e., se afrontan requisitos básicos sin incluir muchas de las funciones suplementarias (ya sean conocidas o no) las cuales permanecen sin extraerse. Como resultado de la utilización y evaluación del primer incremento, se desarrolla un plan para el incremento siguiente. El plan afronta las modificaciones del sistema central, a fin de cumplir mejor con las necesidades del sistema y la entrega de funciones y características adicionales. Este proceso se repite siguiendo

la entrega de cada incremento, hasta que se desarrolla el sistema completo.

4.1.2. Paradigma de programación y modelado del soporte

Android es el sistema operativo sobre el cual se desarrolló el soporte; es por ello que se utilizó el paradigma de programación orientado a objetos mediante el lenguaje de alto nivel Java.

La programación orientada a objetos es un método que permite modelar abstracciones de objetos, tanto físicos como intangibles, para ciertos problemas acotados por un conjunto de reglas de negocio. Con cada objeto que se modela, se define una serie de atributos para esa entidad y un comportamiento.

Una de las principales ventajas que brinda este paradigma es el de la reutilización de código y modularización del sistema. Como se mencionó en la sección 1.4, el objetivo general de la presente tesis consiste en desarrollar un soporte multi-usuario y multiplataforma para dispositivos móviles mediante técnicas de remodelación y redistribución plásticas, que facilite de forma flexible transiciones del trabajo individual al trabajo colaborativo y viceversa en entornos cara a cara. Este soporte funge entonces únicamente como una capa intermedia entre el dispositivo y la aplicación final, con el objetivo de explotar las capacidades tecnológicas de los dispositivos para permitir a los desarrolladores extender el modelo de tareas de una aplicación inicialmente de carácter individual.

Para el desarrollo del soporte propuesto, se requirieron conocimientos técnicos del lenguaje de programación Java y del sistema operativo Android, además del aprovechamiento inteligente de las funciones y capacidades que proveen los sensores integrados en los dispositivos móviles. Esto nos permitió construir un sistema que cubriera cada uno de los objetivos especificados en la sección 1.4. Un desarrollo adecuado bajo esta plataforma siempre es posible, siguiendo las pautas y recomendaciones descritas en la guía de desarrolladores en línea de Android.

4.2. Implementación del soporte

Aunque la arquitectura y el diseño mostrados en el capítulo 3 han sido implementados para la plataforma Android, es posible utilizar los mismos recursos para futuras implementaciones en otros lenguajes. A continuación, presentamos los detalles de implementación de cada uno de los servicios que provee el soporte propuesto, así como la forma en cómo se utilizan y diversos puntos clave que nos permitieron alcanzar los objetivos planteados.

4.2.1. Servicio de descubrimiento de dispositivos

Como hemos mencionado previamente, este servicio se encarga de determinar una de las variables contextuales más importantes: el modelo del ambiente físico (ver sección 2.1). Para ello, proporcionamos un servicio que se encarga de detectar qué entidades (dispositivos móviles) se encuentran disponibles en una red inalámbrica. De esta forma podemos automatizar el proceso de reconocimiento de aquellos dispositivos que se encuentren ejecutando una aplicación que implemente el soporte propuesto. Para que el proceso de reconocimiento sea efectivo y completamente automático, se deben extraer los siguientes datos de identificación: IP del dispositivo y

puertos TCP y UDP disponibles, a través de los cuales se puedan establecer conexiones con otros dispositivos. Adicionalmente, anexamos a estos datos de identificación un estado de la aplicación, mediante el cual podemos hacerle saber a las otras entidades si el usuario está dentro de la aplicación o está por abandonarla. Dichos datos de identificación son propagados y recolectados automáticamente cada cierto tiempo y durante todo el ciclo de vida de la aplicación.

Este servicio está dividido en dos actividades principales: difusión y recolección de datos. Para la propagación de la información utilizamos JSON, ya que este formato es ampliamente usado y nos permite convertir objetos en `Strings`, facilitando en gran medida la transmisión de paquetes por la red. El algoritmo utilizado para la difusión está implementado en la clase **DiscoveryPublishService**, el cual definimos a continuación:

Algoritmo 1 Difusión de los datos de identificación del dispositivo y estado de la aplicación

```
1:  procedure IDENTIFICATIONDATADIFUSION(IP, puertoTCP, puertoUDP)
2:      Agregar ("IP", IP) al objeto JSONobject
3:      Agregar ("TCP', puertoTCP) al objeto JSONobject
4:      Agregar ("UDP", puertoUDP) al objeto JSONobject
5:      Agregar ("isAvailable", true) al objeto JSONobject
6:
7:      while no se abandone la aplicación do
8:          Propagar el objeto JSONobject mediante mensajes de tipo broadcast (UDP)
9:          Pausar el servicio PUBLISHING_DELAY milisegundos
10:     end while
11:
12:     Agregar ("isAvailable", false) al objeto JSONobject
13:
14:     for i:=1 hasta LEAVING_ALERT_MESSAGES do
15:         Propagar el objeto JSONobject modificado mediante mensajes de tipo broadcast
16:         Pausar el servicio LEAVING_DELAY milisegundos
17:     end for
18: end procedure
```

De este algoritmo podemos resaltar las siguientes características:

- El objeto `JSONobject`, en el cual se almacenan los datos de identificación, tiene un comportamiento similar al de un `HashMap` (almacenamiento de datos en formato llave-valor, en donde la llave es un identificador de tipo `String` y el valor puede ser cualquier tipo de objecto).

- Dentro de la clase **DiscoverySettings**, se encuentran las constantes `PUBLISHING_DELAY` y `LEAVING_DELAY`, las cuales delimitan los intervalos de propagación de mensajes. La primera de estas variables indica al servicio cuánto tiempo debe esperar antes de propagar los datos por la red nuevamente; así mismo, la segunda variable indica cuánto tiempo se debe esperar antes de difundir el siguiente mensaje de abandono de la aplicación. Inicialmente ambas están establecidas en 1000 y 300 milisegundos respectivamente, sin embargo es posible ajustar estos valores conforme a los requerimientos de una aplicación en particular.

- `LEAVING_ALERT_MESSAGES` indica al servicio cuántos mensajes de abandono se deben enviar antes de que la aplicación se cierre. Se definió esta variable ya que es posible que los otros dispositivos no reciban el mensaje de abandono al primer intento, debido a que son enviados utilizando el protocolo UDP.

Por otro lado, se encuentra el algoritmo de recolección, el cual le permite al soporte recabar los datos difundidos y cuya definición se muestra a continuación:

Algoritmo 2 Recolección de los datos de identificación y estado de otros dispositivos

```
1:  procedure identificationDataCollection(IP propia)
2:      while no se abandone la aplicación do
3:          Estar a la espera de paquetes de tipo broadcast (operación bloqueante de sockets)
4:
5:          if paquete recibido then
6:              Extraer los valores del JSONobject recibido
7:              if "IP" not IP propia then
8:                  if "isAvailable" = true then
9:                      Agrupar los valores extraídos en un HashMap
10:                     Agregar el HashMap en la lista DISCOVERED_DEVICES si es que
                        no existía ya
11:                     Notificar que un dispositivo ha sido reconocido y agregado
12:                 else
13:                     if dispositivo se encuentra en GROUP_LIST then
14:                         Eliminarlo de GROUP_LIST
15:                     end if
16:                     Eliminarlo de la lista DISCOVERED_DEVICES
17:                     Notificar que un dispositivo ha abandonado la aplicación y no está
                        disponible
18:                 end if
19:             end if
20:         end if
21:     end while
22: end procedure
```

Con el algoritmo 2 introducimos dos recursos compartidos (definidos en la clase **SharedResources**) que son clave a lo largo del soporte: `DISCOVERED_DEVICES` y `GROUP_LIST` (el cual se describe más adelante en la sección 4.2.2). `DISCOVERED_DEVICES` es básicamente una lista de descubrimiento en la cual se van almacenando los datos recolectados de forma ordenada (i.e., agrupados en objetos `HashMap`), en donde cada entrada representa de forma única a algún dispositivo presente en la red. Debido a que dos o más hilos podrían en algún momento acceder a este recurso, se decidió implementar un control de concurrencia básico, el consiste en aplicar una envoltura de sincronización proporcionada por `Collections.synchronizedList()` para operaciones comunes y en realizar sincronizaciones manuales cuando se deba iterar sobre la lista. Dicha envoltura se encarga de levantar automáticamente una barrera cada vez que se hacen operaciones sobre el recurso compartido; de esta forma aseguramos que solamente un hilo haga uso de

la lista. Es indispensable sincronizar manualmente cuando se itera sobre el recurso, ya que la envoltura no proporciona esa funcionalidad. Debido a que esta operación se realiza constantemente sobre `DISCOVERED_DEVICES`, las iteraciones se realizan en una copia de la lista (la cual se obtiene al instanciar una lista dentro de un bloque sincronizado, con todos los objetos que contiene `DISCOVERED_DEVICES` en ese momento); de esta forma evitamos que el recurso compartido permanezca bloqueado por un tiempo considerable.

Adicionalmente, las líneas 11 y 17 del algoritmo 2 corresponden a los métodos que fungen como *listeners*, los cuales se encargan de ejecutar el código proporcionado por los desarrolladores para el evento en el que un dispositivo sea agregado o retirado de la lista de descubrimiento, i.e., de `DISCOVERED_DEVICES`. Dichos métodos son parte de la interfaz definida en **DiscoveryServiceListener**.

Cabe mencionar que las clases **DiscoveryPublishService** y **DiscoveryCollectService** heredan de la clase abstracta **AsyncTask**, i.e., que ambas tareas (de difusión y recolección) se ejecutan en un mismo hilo, el cual es distinto al hilo principal (*main thread*, o como se le conoce comúnmente: *UI thread*). Es importante recalcar que todas las clases que hereden de **AsyncTask** se ejecutan en el mismo hilo secundario, razón por la cual no se deben llevar a cabo tareas que requieran de un tiempo de procesamiento muy alto o que tengan operaciones bloqueantes (como la función `sleep()`), ya que esto afectaría el tiempo de reacción de este servicio. Como alternativa, se pueden utilizar clases que implementen la interfaz **Runnable**.

4.2.2. Servicio de control de acoplamiento

Este servicio es el percutor principal de la transición entre el modo de trabajo individual y el colaborativo, pues es el encargado de la creación de sesiones colaborativas, tal y como se describió en la sección 3.2.2. Con tal fin, la implementación de este servicio está dividida en las siguientes fases secuenciales:

- **Preparación del dispositivo para procesar un gesto de acoplamiento**: este gestor es el medio a través del cual se indica al soporte que debe proceder a intentar acoplar este dispositivo con otro. Para denotar que se tiene la intención de realizar un gesto de acoplamiento, la clase **LongClickDetector** se encarga de determinar cuándo el usuario ha realizado un click largo sobre la superficie del trabajo (ver sección 3.3.5). Para ello, basta con llamar el método `verifyLongClick(MotionEvent)` en el evento `onTouch(View, MotionEvent)` del objeto que representa al espacio de trabajo (el cual hereda de la clase **View**). El objeto que se envía como parámetro (`MotionEvent`), contiene la información requerida para contextualizar un gesto, i.e., para determinar si un click largo es válido o no.

  Existen alternativas para detectar clicks largos sin la necesidad de implementar la lógica de este evento (e.g., la clase **GestureDetector** o el método `setOnLongClickListener()` que posee cualquier clase que herede de **View**), sin embargo esas soluciones no permiten configurar algunos aspectos del click largo (e.g., duración y presición), por lo cual son menos flexibles. Mediante el algoritmo 3 mostramos la implementación propuesta, de la cual podemos resaltar lo siguiente:

- Cada gesto de tipo *touch* efectuado sobre la pantalla de un dispositivo puede registrar múltiples eventos, no obstante, solamente tres de ellos son relevantes en el contexto de la detección de un click largo, los cuales están denotados por las siguientes constantes: `ACTION_DOWN` (un dedo entró en contacto con la pantalla), `ACTION_MOVE` (un dedo es arrastrado por la pantalla) y `ACTION_UP` (un dedo se ha levantado de la pantalla).
- El hilo en espera mencionado en la línea 7 es el elemento crítico; la lógica del algoritmo indica que dicho hilo deberá esperar cierto tiempo (`LONG_CLICK_PROC_TIMER` milisegundos) antes de poder ejecutarse. Si el hilo se ejecuta, entonces quiere decir que el usuario tocó la pantalla durante al menos `LONG_CLICK_PROC_TIMER` milisegundos (línea 7) y que no arrastró su dedo en los ejes `x` ni `y` más de `MOVEMENT_GAP` pixeles (línea 9), i.e., que dejó su dedo prácticamente en el mismo lugar durante un tiempo determinado. Una vez arrancado el hilo, este avisa al usuario mediante retroalimentación háptica y acústica que el click largo ha sido reconocido y aceptado (línea 19).
- La línea 6 reinicia el reconocimiento de un click largo cada vez que se coloca un dedo sobre la pantalla (i.e., cuando se reconoce un evento de tipo `ACTION_DOWN`).
- La línea 13 cancela el hilo siempre que se levante el dedo de la pantalla. Dicha cancelación solamente tendrá efecto si el hilo no ha sido lanzado, i.e., si no han transcurrido más de `LONG_CLICK_PROC_TIMER` milisegundos.

Algoritmo 3 Detección de clicks largos

```
1:  function verifyLongClick(MotionEvent)
2:      event ← Extraer evento del parámetro MotionEvent
3:      if event fue ACTION_DOWN then
4:          x ← Posición x del evento (obtenida de MotionEvent)
5:          y ← Posición y del evento (obtenida de MotionEvent)
6:          longClickDetected ← falso
7:          Poner un hilo en espera durante LONG_CLICK_PROC_TIMER milisegundos
8:      else if event fue ACTION_MOVE then
9:          if Si el dedo se movió en x o en y más de MOVEMENT_GAP pixeles then
10:             Cancelar el hilo que se puso en espera en la línea 7
11:         end if
12:     else if event fue ACTION_UP then
13:         Cancelar el hilo que se puso en espera en la línea 7
14:     end if
15:     return longClickDetected
16: end function
17: procedure Tareas del hilo de la línea 7
18:     longClickDetected ← verdadero
19:     Dar retroalimentación acústica y háptica
20: end procedure
```

- La línea 9 verifica que el usuario no arrastre su dedo desde la posición en la que se comenzó el gesto. La constante `MOVEMENT_GAP` establece un margen de movimiento

límite, ya que al mantener un dedo sobre la pantalla del dispositivo estaremos tocando varios puntos al mismo tiempo.

- Constantes y variables configurables de este algoritmo: `LONG_CLICK_PROC_TIMER`, volumen de la retroalimentación audible, intensidad de la retroalimentación háptica y `MOVEMENT_GAP`.

- **Reconocimiento de gestos de acoplamiento**: una vez determinado que se ha realizado un click largo, el dispositivo está listo para procesar un gesto de acoplamiento. Esta fase es particularmente configurable, ya que de ser necesario, los desarrolladores podrían utilizar algún otro mecanismo de interacción para determinar el lado por el cual se desea acoplar un dispositivo con otro. En la presente tesis se propone habilitar a un dispositivo con la capacidad de procesar dos o más gestos de acoplamiento, los cuales deberían poder ser utilizados de forma indistinta. La implementación proporcionada ya cuenta con dos gestos que cumplen esta característica (descritos en el capítulo 3, sección 3.2.2).

  Debido a que los gestos de acoplamiento siempre proceden a un click largo válido, estos deben ser evaluados dentro del mismo evento `onTouch(View, MotionEvent)` que el click largo. El desarrollador es quien habilita dichos gestos proporcionados, instanciando un objeto de las clases **SimplePinchGesture** y/o **SimpleAccelerometerGesture** y llamando a sus correspondientes funciones `validateGesture(...)`. El algoritmo 4 muestra una configuración sugerida para la detección del click largo y la activación de los gestos de acoplamiento.

Algoritmo 4 Configuración sugerida para la detección de un click largo y la activación de ambos gestos de acoplamiento

```
1:  procedure REGULARSETUP(enablePinchGesture, enableAccelerometerGesture)
2:      pinchGesture ← Instacia de la clase SimplePinchGesture
3:      accelerometerGesture ← Instacia de la clase SimpleAccelerometerGesture
4:      procedure ONTOUCH(View, MotionEvent)
5:          validLongClick ← Evaluar si el evento es un click largo con verifyLongClick(...)
6:          validPinchGesture ← falso
7:          if validLongClick = verdadero and enablePinchGesture = verdadero then
8:              validPinchGesture ← Evaluar si es un gesto de arrastre con la función
                                    pinchGesture.validateGesture(...)
9:          end if
10:         if validLongClick = verdadero and enableAccelerometerGesture = verdadero
              and validPinchGesture = falso then
11:             validAccelerometerGesture ← Evaluar si es un gesto de arrastre con la función
                                            accelerometerGesture.validateGesture(...)
12:         end if
13:
14:         Notificar que se realizó un evento de tipo touch sobre la pantalla
15:     end procedure
16: end procedure
```

Algunos puntos importantes de este algoritmo son los siguientes:

- Los parámetros `enablePinchGesture` y `enableAccelerometerGesture` sirven para indicar al soporte qué gestos de acoplamiento habilitar si se utiliza una configuración típica.
- En todo momento, solamente un gesto debe ser correcto y proceder con la consecuente difusión de un intento de acoplamiento. La línea 10 permite verificar que el gesto de inclinación no sea procesado si ya se detectó un gesto de arrastre correctamente. Se deben añadir verificaciones adicionales por cada gesto de acoplamiento que sea proporcionado por los desarrolladores.
- La línea 14 es una notificación (a modo de *listener*), en la cual se ejecuta el código proporcionado por los desarrolladores para cuando ocurre un evento de tipo *touch* sobre el espacio de trabajo y no se ha detectado un click largo. Dicho método de notificación está definido en la interfaz **WorkspaceTouchEventListener**.

Con la configuración presentada, es posible brindar una noción acerca de la forma en cómo se deben utilizar los gestos implementados. En el algoritmo 5 se muestra el funcionamiento interno de cada uno de dichos gestos.

Algoritmo 5 Detección de un gesto de arrastre

```
1: function validateGesture(MotionEvent, validLongClick)
2:     event ← Extraer evento del parámetro MotionEvent
3:     if event fue ACTION_DOWN then
4:         x ← Posición x del evento (obtenida de MotionEvent)
5:         y ← Posición y del evento (obtenida de MotionEvent)
6:     else if event fue ACTION_UP and validLongClick = verdadero then
7:         direction ← ""
8:         deltaX ← x - posición x del evento (obtenida de MotionEvent)
9:         deltaY ← y - posición y del evento (obtenida de MotionEvent)
10:        if |deltaX| > MIN_PINCH_PX_DISTANCE then
11:            if deltaX < 0 then
12:                direction ← "Derecha"
13:            else
14:                direction ← "Izquierda"
15:            end if
16:        else if |deltaY| > MIN_PINCH_PX_DISTANCE then
17:            if deltaY < 0 then
18:                direction ← "Abajo"
19:            else
20:                direction ← "Arriba"
21:            end if
22:        end if
23:        if direction no está vacía and LOCKED_REQUESTS = falso then
24:            LOCKED_REQUESTS ← verdadero
```

```
25:             Estar a la escucha de un intento de acoplamiento externo
26:             Difundir intento de acoplamiento
27:             return verdadero
28:         end if
29:     end if
30:     return falso
31: end function
```

Como es posible apreciar en el algoritmo 5, el parámetro `validLongClick` se obtiene de la evaluación realizada en la línea 5 del algoritmo 4, lo cual implica que este gesto no se puede contextualizar si no se ha detectado un click largo que le anteceda. Básicamente, lo que hace este gesto es comparar el primer y último punto de contacto del dedo con la pantalla; si la diferencia entre ambos es lo suficientemente grande (i.e., es mayor a `MIN_PINCH_PX_DISTANCE` pixeles), ya sea en el eje `x` o en el `y`, entonces se selecciona el lado correspondiente para ser difundido en un intento de acoplamiento (línea 26). El algoritmo 5 introduce el recurso compartido `LOCKED_REQUESTS`, el cual básicamente se comporta como un seguro para evitar que se difunda otro intento de acoplamiento, antes de que se termine de procesar el que está en curso. Este seguro es indispensable, ya que un usuario podría realizar dos gestos de acoplamiento correctos, consecutivos y con direcciones opuestas, sin darle la oportunidad al soporte de procesar un intento externo (línea 25). La lógica utilizada para la difusión se encuentra en el algoritmo 7 y para la recepción de intentos de acoplamiento en los algoritmos 8 y 9.

Al igual que el gesto de arrastre, el gesto de inclinación (algoritmo 6) no se puede contextualizar si no se ha detectado un click largo y, a diferencia del primero, la función `validateGesture(...)` solamente se llamará una sola vez al levantar el dedo de la pantalla (línea 3). De esta forma, el gesto de inclinación comenzará inmediatamente después de terminar el click largo y de retirar el dedo.

Algoritmo 6 Detección de un gesto de inclinación

```
1: function validateGesture(MotionEvent, validLongClick)
2:     event ← Extraer evento del parámetro MotionEvent
3:     if event fue ACTION_UP and validLongClick = verdadero and
           isRunning = falso then
4:         isRunning ← verdadero
5:         Deshabilitar la rotación automática de la pantalla
6:         rotation ← Rotación de la pantalla
7:         Habilitar el acelerómetro y muestrear durante SAMPLING_TIME milisegundos
8:         meanX ← Aceleración en el eje de las x de la primera muestra
9:         meanY ← Aceleración en el eje de las y de la primera muestra
10:        Transformar meanX y meanY de acuerdo a rotation
11:        for all muestras do
12:            x ← Aceleración en el eje de las x
13:            y ← Aceleración en el eje de las y
```

Así mismo, es necesario deshabilitar la rotación automática de la pantalla previamente al muestreo del acelerómetro, ya que de no hacerlo, el usuario podría provocar que la pantalla rote al inclinar el dispositivo y afectar el resultado final del gesto (línea 5).

```
14:            Transformar x y y de acuerdo a rotation
15:            if |x| > ABOVE_THIS_IS_NOISE or |y| > ABOVE_THIS_IS_NOISE then
16:                Ignorar esta muestra
17:            else
18:                meanX ← (meanX + x)/2
19:                meanY ← (meanY + y)/2
20:            end if
21:        end for
22:        direction ← ""
23:        if |meanX| > |meanY| then
24:            if meanX < −MIN_VALID_INCLINATION then
25:                direction ← "Derecha"
26:            else if meanX > MIN_VALID_INCLINATION then
27:                direction ← "Izquierda"
28:            end if
29:        else
30:            if meanY < −MIN_VALID_INCLINATION then
31:                direction ← "Abajo"
32:            else if meanY > MIN_VALID_INCLINATION then
33:                direction ← "Arriba"
34:            end if
35:        end if
36:        Habilitar rotación automática de la pantalla
37:        if direction no está vacía and LOCKED_REQUESTS = falso then
38:            LOCKED_REQUESTS ← verdadero
39:            Estar a la escucha de un intento de acoplamiento externo
40:            Difundir intento de acoplamiento
41:            isRunning ← falso
42:            return verdadero
43:        end if
44:    end if
45:    isRunning ← falso
46:    return falso
47: end function
```

El muestreo se lleva a cabo durante SAMPLING_TIME milisegundos, tiempo durante el cual se calcula el promedio de las aceleraciones registradas en los ejes x y y (el eje z no brinda información relevante para poder seleccionar un lado del dispositivo). Dichas aceleraciones registradas no deberían sobrepasar el valor de la gravedad si el dispositivo se mantiene inclinado e inmóvil. Para mantener el valor del promedio coherente cuando se realizen movimientos bruscos por accidente, se descartan todas aquellas muestras que sobrepasen

la cota de aceleración máxima `ABOVE_THIS_IS_NOISE` m/s^2, cuyo valor es de 9.8 m/s^2 por omisión. Una vez terminado el periodo de muestreo, se verifican los promedios de las aceleraciones finales en cada eje y se selecciona un lado si alguno de los ejes registró un promedio mayor a `MIN_VALID_INCLINATION` m/s^2, lo cual implica que el dispositivo se inclinó intencionalmente durante más tiempo hacia un solo lado. Posterior a la selección de un lado del dispositivo, se siguen las mismas pautas que se utilizan en el algoritmo 5 para difundir el intento de acoplamiento.

Cabe mencionar que las transformaciones realizadas en las líneas 10 y 14 sirven para rotar los valores de las muestras obtenidas dependiendo de la orientación del dispositivo, ya que el acelerómetro no las transforma automáticamente [NVIDIA, 2010].

- **Difusión de un intento de acoplamiento**: una vez procesado un gesto y seleccionado algún lado del dispositivo, se procede a difundir un intento de acoplamiento. Para la transmisión de datos se utiliza el mismo mecanismo que para el servicio de descubrimiento de dispositivos, i.e., se almacenan los datos en un objeto de tipo `JSONobject` en tuplas de tipo llave-valor, el cual se transforma en un `String` para su posterior difusión por la red. La implementación de esta tarea se encuentra en la clase **SendOnePairingRequest**, cuyo algoritmo se muestra a continuación.

Algoritmo 7 Difusión de un intento de acoplamiento

```
1:  class SendOnePairingRequest (IP, puertoTCP, puertoUDP, direction, features)
2:      Agregar ("IP", IP) al objeto JSONobject
3:      Agregar ("TCP", puertoTCP) al objeto JSONobject
4:      Agregar ("UDP", puertoUDP) al objeto JSONobject
5:      Agregar ("DIR", direction) al objeto JSONobject
6:      Agregar ("SPECS", features) al objeto JSONobject
7:
8:      Propagar el objeto JSONobject mediante un mensaje de tipo broadcast (UDP)
9:
10:     Dar retroalimentación audible
11: end class
```

Como es posible apreciar en el algoritmo 7, un intento de acoplamiento está compuesto por la dirección IP del dispositivo, los puertos TCP y UDP en donde se encuentra operando el servicio de distribución de actualizaciones, el lado seleccionado a través de un gesto de acoplamiento y las especificaciones del dispositivo (líneas 2-6) extraídos del parámetro `features`. Dichas especificaciones se obtienen a través de la clase **DeviceFeatures**, las cuales deben ser recabadas al inicio de la aplicación y cuando se rota la pantalla (a través del método `onConfigurationChanged(...)`). El método `setAllFeatures(...)` es el encargado de recabar los siguientes datos:

- Dimensiones y densidad de pixeles de la pantalla.
- Sensores disponibles.
- Interfaces de comunicación.

- Modelo del dispositivo.
- Versión del Sistema Operativo.
- Funcionalidades adicionales (e.g., si tiene soporte para múltiples ventanas, soporte para *multitouch*, protocolos de comunicación disponibles).

Por lo tanto, un intento de acoplamiento se puede considerar como una representación lógica de un dispositivo, a través de la cual este puede ser identificado de forma única en la red. La difusión del intento se lleva a cabo una sola vez y de forma similar a como se difunden los mensajes con el servicio de descubrimiento.

- **Recepción de un intento de acoplamiento externo**: esta tarea es llevada a cabo en conjunto con la difusión del intento propio, cuyo objetivo consiste en capturar el intento de acoplamiento difundido por algún otro dispositivo. La implementación de dicha tarea se encuentra en la clase **ListenOnePairingRequest**, la cual se muestra a continuación mediante los algoritmos 8 y 9.

Algoritmo 8 Captura de un intento de acoplamiento - validación

```
1: class ListenOnePairingRequest (direction, features)
2:     ACT_AS_SERVER ← falso
3:     Esperar por un intento externo durante COUPLING_WAITING_TIME milisegundos
4:     if Intento recibido then
5:         if Intento proviene de este mismo dispositivo then
6:             Esperar nuevamente por un intento externo
7:             if Intento recibido then
8:                 Llamar función processIntent(direction, features, message)
9:             else
10:                LOCKED_REQUESTS ← falso
11:                Dar retroalimentación para indicar que no se recibió un intento
12:            end if
13:        else
14:            Llamar función processIntent(direction, features, message)
15:        end if
16:    else
17:        Dar retroalimentación para indicar que no se recibió un intento
18:        LOCKED_REQUESTS ← falso
19:    end if
20: end class
```

Básicamente, lo que el algoritmo 8 hace es esperar `COUPLING_WAITING_TIME` milisegundos por un intento externo; si el intento no es recibido en ese lapso de tiempo, el servicio de control determina que el acoplamiento de dispositivos no procede, liberando el seguro `LOCKED_REQUESTS` (puesto por los algoritmos 5 o 6 al procesar un gesto de acoplamiento exitoso) y brindando retroalimentación audible para notificar al usuario que el dispositivo está listo para procesar otro gesto de acoplamiento en cualquier momento. Si por el contrario se recibe algún intento, entonces se verifica que este no sea el mismo que emitió el propio

dispositivo (línea 5). Esta verificación es indispensable, ya que cuando se difunden datos por la red mediante mensajes de tipo *broadcast*, el dispositivo emisor también recibe su propio mensaje. Si ese es el caso, entonces se espera nuevamente `COUPLING_WAITING_TIME` milisegundos por otro intento. Para el caso en el cual se reciba un intento externo, entonces se procede al algoritmo 9 (líneas 8 y 14).

Adicionalmente, el recurso compartido `ACT_AS_SERVER` es regresado a su estado inicial. Dicho recurso forma parte del protocolo de actualización de las listas de grupo y del espacio de trabajo (detallado en la última fase del servicio de control de acoplamiento).

Algoritmo 9 Captura de un intento de acoplamiento - procesamiento

```
1: procedure processIntent(direction, features, message)
2:     newDevice ← Crear un objeto de tipo HashMap
3:     Obtener de message, los valores asociados a las llaves "IP", "TCP", "UDP", "DIR",
       "SPECS" y almacenarlos en newDevice
4:     if direction es contraria a la dirección recibida and GROUP_LIST no contiene el objeto
         newDevice then
5:         Dar retroalimentación audible de que el acoplamiento es exitoso
6:         Agregar newDevice a GROUP_LIST
7:         if GROUP_LIST contiene 1 solo elemento then
8:             ownDevice ← Crear un objeto de tipo HashMap
9:             ownDevice ← Las mismas llaves que en newDevice pero con los valores propios
10:            Agregar ownDevice a GROUP_LIST
11:        end if
12:        Comenzar el protolo de actualización de la lista de grupo y el espacio de trabajo
13:    end if
14: end procedure
```

La parte de procesamiento, posterior a la recepción de un intento externo, consiste en extraer los datos del otro dispositivo y verificar que el acoplamiento se pueda llevar a cabo. Dicha verificación está dividida en dos fases consecutivas:

1. Comparar el valor asociado a la llave ‘‘`DIR`’’ (i.e., el lado recibido en el intento, dentro del objeto `message`) con el lado obtenido a través de la contextualización del gesto de acoplamiento (i.e., el lado seleccionad propio pasado como el parámetro `direction` en el algoritmo 9). El intento se considera válido si ambos lados comparados son opuestos entre sí (e.g., “Arriba” y “Abajo” o “Izquierda” y “Derecha”).
2. Revisar que la lista de grupo (`GROUP_LIST`) no contenga el dispositivo a agregar, i.e., que no exista una entrada en la lista que contenga exactamente los mismos elementos que `newDevice`.

Una vez verificado que `newDevice` es un dispositivo válido y nuevo, se agrega a la lista de grupo. Así mismo, se verifica si la lista de grupo tiene un solo elemento, i.e., si se creará una sesión colaborativa o se agregará el dispositivo a una sesión ya existente (ver sección 3.2.2). De existir sólo un elemento, se debe agregar una entrada con los datos

propios. Es importante recordar que existe una lista `GROUP_LIST` en cada dispositivo y que es precisamente el servicio de control de acoplamiento quien se encarga de administrarla. Así mismo, existe un orden en cuanto a la fase de envío y recepción de un intento: se decidió que el dispositivo primero habilitara la recepción y después difundiera el intento propio. Esto se debe a que, si se finalizan dos gestos de acoplamiento al mismo tiempo en dos dispositivos distintos, podría suceder que ambos difundan su intento de acoplamiento, pero que el otro dispositivo no estuviera listo para recibirlo y procesarlo debido a que estaba ocupado realizando el envío. Al habilitar la fase de recepción en primer lugar, es más probable que al menos un dispositivo reciba un intento externo, lo que aseguraría el acoplamiento en caso de que éste sea valido.

- **Actualización de la lista de grupo y del espacio de trabajo**: posteriormente, se procede con el protocolo para la actualización de la lista de grupo y del espacio de trabajo. Una vez agregadas las entradas correspondientes a la lista de grupo, inicia un procedimiento para decidir cuál de los dos dispositivos recién acoplados se encargará de realizar la última fase. Dicho procedimiento está dividido en los algoritmos 10 y 11. El algoritmo 11 (etapa de respuesta) opera en el dispositivo$_i$, con el objetivo de contestar la solicitud realizada por el algoritmo 10 (etapa de solicitud y reacción), el cual es ejecutado en el dispositivo$_j$ con el cual se llevó a cabo el acoplamiento.

Algoritmo 10 Proceso de acuerdo - solicitud y reacción

```
1: procedure DISTRIBUTIONMANNAGEMENT_ASKANDREACTPHASE
2:     Preguntar al otro dispositivo si ya se encuentra a cargo de la distribución
3:     Esperar la respuesta del otro dispositivo, el cual ejecuta el algoritmo 11 al recibir la
       consulta
4:     if El otro dispositivo no está a cargo then
5:         ACT_AS_SERVER ← verdadero
6:         Combinar la lista de grupo propia con la lista recibida del otro dispositivo
7:         Combinar el espacio de trabajo propio con el espacio recibido del otro dispositivo
8:         Distribuir la lista de grupo nueva y los objetos contenidos en el espacio de trabajo
           recién formado
9:     end if
10: end procedure
```

Algoritmo 11 Proceso de acuerdo - respuesta

```
1: procedure DISTRIBUTIONMANNAGEMENT_RESPONSEPHASE
2:     if ACT_AS_SERVER = verdadero then
3:         Responder que este dispositivo ya está a cargo
4:     else
5:         Responder que este dispositivo no está a cargo, adjuntando la lista de grupo propia y
           los objetos del espacio de trabajo (WORKSPACE)
6:     end if
7: end procedure
```

Como se mencionó anteriormente, es el servicio de distribución de actualizaciones quien se encarga de llevar a cabo la actualización de la lista de grupo y del espacio de trabajo, sin embargo, es el servicio de control quien le indica que debe iniciar todo el procedimiento de acuerdo para la actualización de la lista de grupo y del espacio de trabajo en todos los dispositivos involucrados.

El algoritmo 10 es ejecutado cada vez que un dispositivo$_i$ recibe un intento de acoplamiento j válido. Dicho algoritmo consiste básicamente en consultar con el dispositivo$_j$, si éste ya se encuentra a cargo de la distribución de la lista de grupo y el espacio de trabajo. La consulta comienza siempre con el envío de un objeto de la clase **RequestForBecomingServer**. Cuando el servicio de distribución de actualizaciones en el dispositivo$_j$ recibe dicho objeto, entonces éste sabe que su intento de acoplamiento fue procesado y validado. Una vez recibido el objeto **RequestForBecomingServer**, el dispositivo$_j$ ejecuta el algoritmo 11, verificando el valor de la variable `ACT_AS_SERVER`, cuyo propósito es servir como referencia para saber si se está o no a cargo de la distribución. En caso de que `ACT_AS_SERVER` sea $falso$, el dispositivo$_j$ enviará como respuesta al dispositivo$_i$, un objeto de la clase **ResponseToRequestFromBecomingServer**, el cual contendrá su lista de grupo y su espacio de trabajo. Al recibir este objeto como respuesta a su consulta, el dispositivo$_i$ deberá cambiar el valor del recurso `ACT_AS_SERVER` a *verdadero* y combinar las listas de grupo y los espacios de trabajo (los propios con los recibidos dentro del objeto **ResponseToRequestFromBecomingServer**), de tal forma que la nueva lista y espacio de trabajo compartido, contengan todos los elementos de sus respectivas sesiones colaborativas anteriores (en caso de que alguno de los dos dispositivos ya perteneciera a un grupo de trabajo, ver capítulo 3, sección 3.2.2). El valor del recurso `ACT_AS_SERVER` se debe cambiar para que, en caso de que el dispositivo$_j$ consulte con el dispositivo$_i$ si este ya se encuentra a cargo de la distribución, se le responda afirmativamente y este ya no tenga que hacer acción alguna. Finalmente, el dispositivo a cargo de la distribución se encarga de enviar los recursos recién combinados a cada uno de los integrantes de la nueva sesión, todo esto a través del método `spreadObject`.

Es importante recalcar que si ambos dispositivos i y j recibieron cada quien el intento difundido por el otro dispositivo, entonces ambos llevarán a cabo las dos partes del algoritmo durante el proceso de acuerdo, i.e., en un momento dado ambos actuarán como el dispositivo$_i$ y el dispositivo$_j$. Por el contrario, si solamente uno de ellos recibe el intento, dicho dispositivo ejecutará el algoritmo 10 y, aquel que no lo captó, únicamente ejecutará el algoritmo 11 de forma reactiva. Esta situación se puede dar debido a que los intentos son difundidos mediante el protocolo UDP, el cual no nos asegura que la información difundida sea recibida.

La implementación del algoritmo 10 se encuentra en la clase **WorkerThread** y la del algoritmo 11 en la clase **UpdateDistributionTCPServer**, las cuales se pueden localizar dentro de la clase **UpdateDistributionService**.

- **Activar los procedimientos de remodelación y redistribución**: por último, se deben aplicar los principios de remodelación y redistribución descritos en el capítulo 2, sección 2.2, readaptando la interfaz de usuario y el modelo de tareas del sistema. Debido a la

variedad de aplicaciones en las cuales se podría utilizar el soporte, es indispensable que sean los desarrolladores quienes provean directamente esta funcionalidad, debido a que una generalización de estos procedimientos delimitaría el espectro de aplicaciones en las cuales poder implementar la presente propuesta.

4.2.3. Servicio de distribución de actualizaciones

La implementación del tercer y último servicio se encuentra principalmente dentro de la clase **UpdateDistributionService**. Una vez terminada la actualización de la lista de grupo y del espacio de trabajo (recurso compartido `WORKSPACE`), el dispositivo queda en espera de acciones por parte del usuario que requieran transmitir objetos a uno o varios dispositivos acoplados. Para ello, este servicio cuenta con dos métodos para transmitir dichos objetos (`sendObject(...)` y `spreadObject(...)`) y con un servidor local definido en la clase **UpdateDistributionTCPServer**, el cual se encarga de recibir y procesar los objetos transmitidos desde otros dispositivos a través de este servicio. La diferencia principal entre `sendObject(...)` y `spreadObject(...)` radica básicamente en que la primera permite enviar objetos únicamente a un solo dispositivo y la segunda envía objetos automáticamente a todo el grupo de trabajo, i.e., a cada uno de los dispositivos registrados en `GROUP_LIST`.

Durante el protocolo descrito en la penúltima fase del acoplamiento de dispositivos (actualización de la lista de grupo y del espacio de trabajo), el envío de la instancia de la clase **RequestForBecomingServer** que sirve como consulta, es realizado mediante el método `sendObject(...)` y procesado en el otro dispositivo (por el servidor local definido en **UpdateDistributionTCPServer**). De la misma forma, `spreadObject(...)` es utilizado para enviar la lista de grupo y el espacio de trabajo a todos los integrantes de la sesión colaborativa, una vez que éstos han sido actualizados.

A continuación se muestra el funcionamiento interno de los métodos de envío y del servidor local.

Algoritmo 12 Procedimiento para el envío de objetos a un solo dispositivo

```
1: procedure SENDOBJECT(target, object, protocol, waitForAnswer)
2:     jsonObject = conversión de object en una cadena de caracteres mediante JsonWriter
3:     if protocol = "TCP" then
4:         Instanciar WorkerThread para enviar el objeto jsonObject mediante el protocolo
           TCP (ver algoritmo 14)
5:     else
6:         Instanciar WorkerThread para enviar el objeto jsonObject mediante el protocolo
           UDP (ver algoritmo 14)
7:     end if
8: end procedure
```

Algoritmo 13 Procedimiento para el envío de objetos a múltiples dispositivos

```
1: procedure SPREADOBJECT(object, protocol)
2:     jsonObject = conversión de object en una cadena de caracteres mediante JsonWriter
```

```
3:     GROUP_LIST_COPY = generar una copia de GROUP_LIST para iterar sobre ella
4:     for each device in GROUP_LIST_COPY do
5:         if protocol = "TCP" then
6:             Instanciar WorkerThread para enviar el objeto jsonObject mediante el
               protocolo TCP (ver algoritmo 14)
7:         else
8:             Instanciar WorkerThread para enviar el objeto jsonObject mediante el
               protocolo UDP (ver algoritmo 14)
9:         end if
10:    end for
11: end procedure
```

Algoritmo 14 Hilo en segundo plano encargado de enviar un objeto en formato JSON

```
1: class WorkerThread (target, jsonObject, protocol, waitForAnswer)
2:     Extraer los datos de target para la conexión con el otro dispositivo
3:     Preparar el envío de jsonObject con los datos de conexión extraídos de acuerdo a protocol
4:     Enviar el objecto a target de acuerdo a protocol
5:     if protocol = "TCP" and waitForAnswer = verdadero then
6:         Esperar por la respuesta del otro dispositivo y procesarla
7:     end if
8: end class
```

Como es posible apreciar, los algoritmos 12 y 13 son muy parecidos. Ambos transforman el objeto a enviar `object` en otro que esté en formato `JSON` a través de una instancia de la clase **JsonWriter** (perteneciente a la biblioteca **json-io**) y preparan, dependiendo del protocolo `protocol` requerido, el hilo (o los hilos, para el caso de `spreadObject(...)`) en segundo plano que estará(n) a cargo del envío o distribución del objeto transformado `jsonObject`. El único requerimiento que deben cumplir las clases, cuyas instancias vayan a ser enviadas mediante los métodos `spreadObject(...)` y `sendObject(...)`, es que tengan definido un constructor. Dicho requerimiento es establecido por las clases **ObjectInputStream** y **ObjectOutputStream** utilizadas para la serialización y deserialización de objetos enviados y recibidos por la red.

Una vez preparado el objeto a transmitir, se arranca el hilo que estará a cargo de dicha tarea. Para el caso del algoritmo 13, se arrancan tantos hilos como dispositivos estén registrados en la lista de grupo. Este procedimiento requiere iterar sobre el recurso compartido `GROUP_LIST`, es por ello que primero se crea una copia de la lista de forma segura (forma conocida en inglés como *thread safe*) y después se itera sobre la copia recién creada `GROUP_LIST_COPY`. Es necesario crear una copia de este recurso ya que, al ser compartido, cualquier hilo podría alterar su contenido mientras se realizan las iteraciones, provocando comportamientos indeseados y excepciones relacionadas con accesos concurrentes.

WorkerThread es el hilo encargado de enviar el objeto y recibir respuestas a dichos envíos (un hilo por objeto para el caso de `spreadObject(...)`). Para llevar a cabo el envío, cada instancia de **WorkerThread** obtiene los datos de conexión del parámetro `target` (el cual es básicamente una entrada de la lista de grupo, i.e., un objeto de tipo `HashMap` con los datos de conexión al-

macenados en formato llave-valor). Después de obtener los datos necesarios, se realiza el envío dependiendo del protocolo `protocol`. Adicionalmente, es posible esperar una respuesta del servidor local en el dispositivo `target` si el protocolo utilizado fue TCP y si así lo requiere el modelo de tareas de una aplicación (definido por los desarrolladores). Un ejemplo ya implementado de dicho comportamiento, se puede apreciar en la espera de un dispositivo, por la respuesta a una consulta (i.e., la recepción de un objeto de la clase **RequestForBecomingServer**) posterior a un acoplamiento exitoso.

Todo objeto enviado por los métodos `sendObject(...)` y `spreadObject(...)`, desde otros dispositivos, es procesado por el servidor local en la clase **UpdateDistributionTCPServer**. La lógica de dicho servidor se muestra en el algoritmo 15.

Algoritmo 15 Servidor local encargado de recibir y procesar objetos

```
1:  class UpdateDistributionTCPServer
2:      while no se abandone la aplicación do
3:          Estar a la espera de objetos externos (operación bloqueante de sockets)
4:          if objeto recibido then
5:              object = conversión del objeto recibo al objeto original mediante JsonReader
6:              Verificar la clase a la cual pertenece object
7:              if object es utilizado por el propio soporte then
8:                  Procesarlo y regresar a la línea 3
9:              else
10:                 Notificar que se recibió un objeto que no es utilizado por el soporte
11:             end if
12:         end if
13:     end while
14: end class
```

Al igual que el servicio de descubrimiento de dispositivos, el servidor local permanece ejecutándose en segundo plano desde que se arranca la aplicación hasta que es terminada. Este comportamiento es requerido, ya que dicho servidor es utilizado durante el acoplamiento de dispositivos cuando un intento es válido. De esta forma, aunque un intento difundido y válido no sea capturado (i.e., que un dispositivo no es consciente de que ya es partícipe de un acoplamiento exitoso) ambos dispositivos (aquellos que difundieron su intento) tendrán la capacidad de recibir y procesar objetos, aún cuando no pertenezcan a una sesión colaborativa.

Una vez que se recibe un objeto, este se convierte de una cadena de caracteres en formato `JSON`, a un objeto de tipo **Object** mediante una instancia de la clase **JsonReader**. Posteriormente, se revisa a qué clase pertenece `object`. Dependiendo de la clase a la cual pertenezca, el servidor local realizará alguna acción. Si el objeto recibido fue enviado por el servicio de control de acoplamiento desde otro dispositivo, entonces ese objeto es procesado de acuerdo a los algoritmos 10 y 11, de lo contrario se notifica que se ha recibido un objeto distinto. Dicha notificación se lleva a cabo mediante el método `checkObjectReceived(...)` definido en la interfaz **UpdateDistributionServiceListener**. Dentro de este método es en donde el desarrollador puede proceder a revisar a cuál de sus propias clases pertene el objeto recibido para procesarlo.

Cabe mencionar que las clases utilizadas por el soporte y cuyos objetos están destinados a ser enviados, se encuentran en el paquete *services.updatedistribution.networkclasses*. Aunque no es necesario, es recomendable que los desarrolladores almacenen sus clases personalizadas (aquellas cuyos objetos serán enviados mediante `sendObject(...)` o `spreadObject(...)`) en dicho paquete por motivos de escalabilidad y orden.

4.3. Asignación de tareas en segundo plano

Como se mencionó en el capítulo 3, sección 3.4.2, todos los servicios hacen uso, ya sea de la interfaz **Runnable** o de la clase **AsyncTask**. Ambas permiten realizar operaciones en segundo plano, lo cual es indispensable si se requiere de largos tiempos de procesamiento que pudieran afectar el rendimiento de la interfaz gráfica (i.e., tareas que bloqueen parcialmente el hilo principal, conocido como *UI thread*); tal es el caso del envío y recepción de datos por la red.

Cualquier clase que implemente la interfaz **Runnable** debe definir dentro del método `run()` lo que hará un hilo cuando sea ejecutado. Por otro lado, aquellas clases que heredan de **AsyncTask**, definen las tareas a realizar en segundo plano dentro de la función `doInBackground(...)`. Una de las diferencias entre ambos recursos radica en la forma de asignación del trabajo por hilo. La interfaz **Runnable** (cuando se le indica) es ejecutada en un hilo por implementación, i.e., por cada clase que implemente dicha interfaz y sea ejecutada, el código definido dentro del método `run()` se llevará a cabo en un hilo distinto. Por otro lado, el número de hilos (si es que es posible utilizar más de uno) para aquellas clases que heredan de **AsyncTask**, está definido por la versión de Android del dispositivo. Esto implica que, independientemente del número de tareas de tipo **AsyncTask** ejecutadas, el sistema operativo es quien decide si se ejecutan en un mismo hilo o no. Adicionalmente, existen ciertas restricciones que limitan de cierta forma la usabilidad de esta clase (e.g., cualquier objeto de una clase que herede de **AsyncTask** debe ser instanciado en el hilo principal y sólo podrá ser utilizado una sola vez) [Google, 2009]. La implementación del soporte utiliza la interfaz **Runnable** y la clase **AsyncTask** como se muestra en las tablas 4.1 y 4.2.

Clase o método	Número de hilos	Tarea
UpdateDistributionService	1	Manejo del servidor local
spreadObject(...)	Tamaño de `GROUP_LIST`	Difusión de un objeto
sendObject(...)	1	Transmisión de un objeto
SendFeaturesToTheGroup	Tamaño de `GROUP_LIST`	Actualización
LongClickDetector	1	Temporización
SimpleAccelerometerGesture	1	Temporización

Tabla 4.1: Utilización de la interfaz **Runnable**

Clase o método	Número de hilos	Tarea
DiscoveryPublishService	1	Difusión de datos
DiscoveryCollectService	1	Recolección de datos
SendOnePairingRequest	1	Difusión de un intento
ListenOnePairingRequest	1	Recolección de un intento

Tabla 4.2: Utilización de la clase **AsyncTask**

4.4. Configuración básica de los servicios implementados

Como se mencionó en la sección 3.4.1, junto con los servicios se proporciona una clase que brinda una configuración básica para el soporte. Dicha clase lleva por nombre **RegularGerberaSetup** y se encuentra en el paquete *global*. Dentro de **RegularGerberaSetup**, se puede apreciar a modo de ejemplo, el orden en el cual se inicializan o liberan los recursos y servicios proporcionados. La lógica de la configuración básica se muestra en el algoritmo 16.

Algoritmo 16 Configuración básica del soporte

```
1:  class RegularGerberaSetup
2:      Inicializar recursos compartidos por los servicios
3:      Inicializar las configuraciones de los servicios
4:      Inicializar los objetos de cada servicio
5:      Arrancar los servicios
6:      Agregar los notificadores para los servicios de descubrimiento y distribución
7:      Inicializar el detector de clicks largos
8:      Inicializar los objetos de los gestos de acoplamiento
9:      Adherir los objetos de los gestos de acoplamiento al espacio de trabajo
10:     Habilitar la actualización de características cuando se rota el dispositivo
11:     Habilitar la finalización de los servicios cuando se cierra la aplicación
12: end class
```

A continuación describimos cada uno de los pasos definidos por el algoritmo 16:

- **Línea 2**: ejecutar el método `init_SharedResources()` de la clase **SharedResources**.
- **Línea 3**: crear un objeto de las clases **DiscoverySettings**, **CouplingSettings** y **UpdateSettings**.
- **Línea 4**: crear una instancia de las clases **UpdateDistributionService**, **DiscoveryPublishService** y **DiscoveryCollectService**.
- **Línea 5**: arrancar el servicio de distribución de actualizaciones con el método `startService()` y los otros dos servicios con sus respectivos métodos `execute(...)`.
- **Línea 6**: agregar los *listeners* de tipo **DiscoveryServiceListener** y **UpdateDistributionServiceListener**, en los cuales se procesan los datos u objetos recibidos de acuerdo a lo descrito en las secciones 4.2.1 y 4.2.3, respectivamente.

- **Línea 7**: crear un objeto de la clase **LongClickDetector**
- **Línea 8**: crear un objeto de cada gesto de acoplamiento (clases **SimplePinchGesture** y **SimpleAccelerometerGesture**), si es que se desea habilitar ambos. De lo contrario crear sólo el objeto requerido.
- **Línea 9**: agregar los validadores del click largo (función `verifyLongClick(...)`) y los gestos de acoplamiento (función `validateGesture(...)` de cada objeto creado en la línea 8) dentro del evento `onTouch(...)` de la clase **View** cuyo objeto representa el espacio de trabajo.
- **Línea 10**: crear un hilo dentro del evento `onConfigurationChanged()` de la actividad principal, cuyo código a ejecutar es precisamente el definido en la clase **SendFeaturesToTheGroup**.
- **Línea 11**: ejecutar los métodos `stopService()` de cada uno de los servicios dentro del evento `onStop()` de la actividad principal, así como el método `freeResources()` de la clase **SharedResources**.

4.5. Descubrimiento de servicios en la red

NSD [Google, 2012] (*Network Service Discovery* por sus siglas en inglés) es una API, cuyo objetivo es permitir a un usuario identificar, de forma sencilla, a otros dispositivos que se encuentren conectados en la misma red, siempre y cuando se les haya dado la capacidad de hacerlo. El descubrimiento de servicios en la red es práctico y bastante útil cuando se requiere conectar múltiples dispositivos, con el fin de compartir archivos, habilitar comunicaciones entre juegos, imprimir documentos, entre otras funciones más.

Uno de los puntos a tomar en cuenta (si es que se desea hacer uso de esta API) es que resulta indispensable que los dispositivos cuenten con una versión del sistema operativo Android 4.1 o superior. De acuerdo a la figura 4.1, Dicho requerimiento es la principal razón por la cual es inviable utilizarlo en el soporte propuesto, ya que lo haría inutilizable para aproximadamente el 36.2 % de los usuarios que cuentan con dispositivos, cuya versión del sistema operativo oscila entre 2.3 y 4.0 [Google, 2014b] (limitando en gran medida el espectro de heterogeneidad de dispositivos).

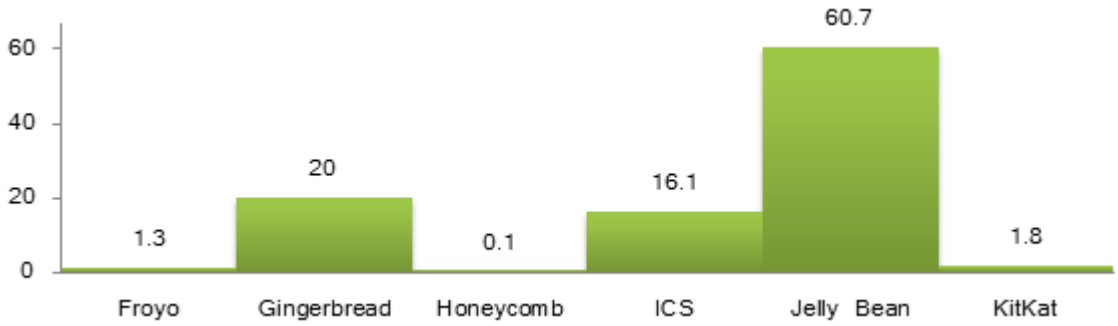

Figura 4.1: Porcentaje de usuarios según los datos recolectados por la tienda de aplicaciones de Google durante un periodo de siete días, terminando el 4 de febrero del 2014.

Capítulo 5

Desarrollo de aplicaciones basadas en el soporte propuesto

En los capítulos 3 y 4 presentamos, respectivamente, el diseño y la implementación del soporte propuesto en esta tesis, de tal forma que sea posible reproducir la presente solución en otra plataforma o simplemente utilizar y ajustar la implementación proporcionada, de acuerdo al modelo de tareas de una aplicación en específico. A continuación presentamos la forma en la cual es posible desarrollar aplicaciones que hagan un uso adecuado del soporte propuesto, junto con la implementación de una aplicación de ejemplo, con la finalidad de demostrar la utilidad y versatilidad de cada uno de los servicios proporcionados.

5.1. Descripción de la aplicación de ejemplo

Básicamente, el objetivo de la aplicación desarrollada es demostrar la utilidad del soporte propuesto y la facilidad con la cual se puede adaptar a las necesidades de los desarrolladores. El modelo de tareas con el cual cuenta la aplicación de ejemplo es el siguiente:

- Modo de trabajo individual (figura 5.1): cuando la aplicación es iniciada, se le asigna un color aleatorio al espacio de trabajo, se inicializa un contador de clicks en cero y se genera un identificador de sesión aleatorio, a pesar de que aún no se haya creado una sesión colaborativa. Dicho identificador se puede considerar como un *token* único a través del cual se puede diferenciar en un futuro cuáles dispositivos pertenecen a la misma sesión (ver sección 5.2.2). Posteriormente, la aplicación se queda en espera por clicks cortos o gestos de acoplamiento para crear una sesión colaborativa. Cada vez que se detecta un click corto, el contador de clicks se incrementa en uno y, cuando se detecta un click largo, el contador permanece intacto y se procede a procesar un gesto de acoplamiento.

- Modo de trabajo colaborativo (figuras 5.2 y 5.3): una vez creada una sesión colaborativa como producto de un intento de acoplamiento exitoso, el contador de clicks se convierte en un recurso compartido y sincronizado en todos los dispositivos. Así mismo, se decide un identificador de sesión único para el grupo de trabajo y se agrega una funcionalidad nueva: cada vez que alguno de los dispositivos acoplados detecte un click corto, dicho integrante comunicará a los otros dispositivos el evento registrado y su color aleatorio generado al

iniciar la aplicación. Al recibir un mensaje con el evento registrado y el color aleatorio, cada dispositivo incrementará el contador de clicks en uno y adicionalmente mostrará un círculo con el color recibido. De esta forma, cada vez que un dispositivo muestre un círculo con un color, se puede conocer el origen del último mensaje recibido. Finalmente, el dispositivo que detectó el click desaparece el círculo de color (si es que tenía alguno en pantalla) en su parte correspondiente del espacio de trabajo.

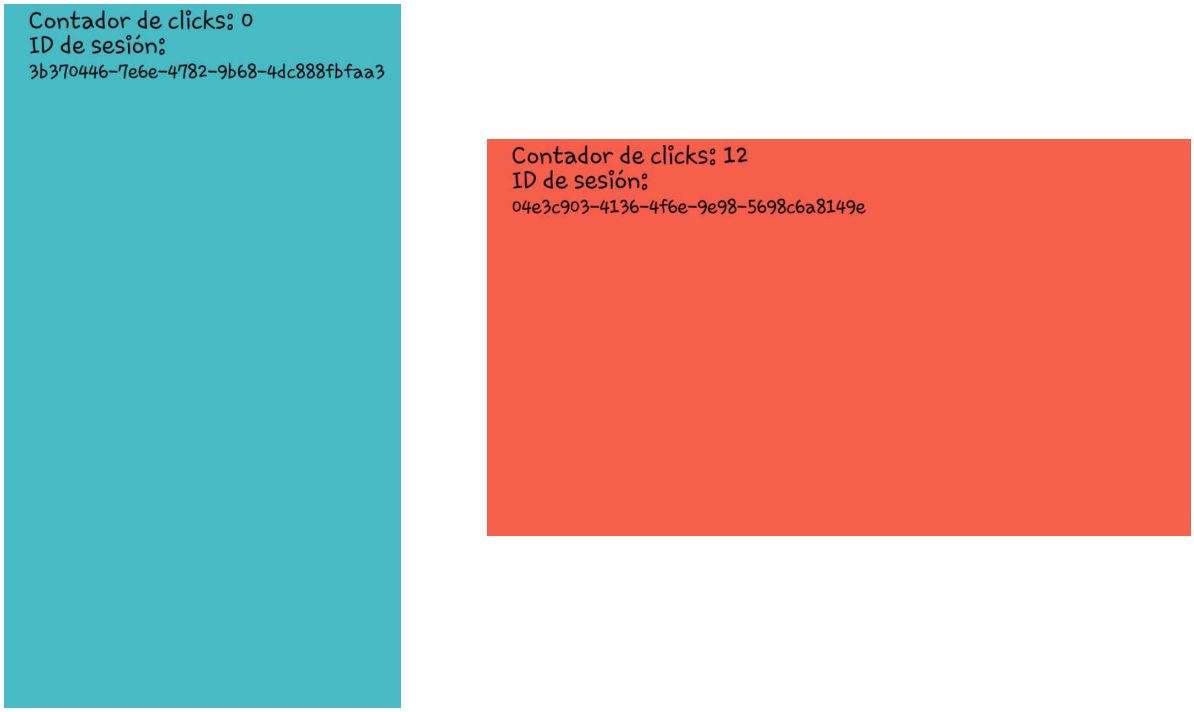

Figura 5.1: Modo de trabajo individual en dos dispositivos distintos (aplicación recién iniciada en la pantalla de la izquierda y una vez que ha detectado 12 clicks cortos en la pantalla de la derecha)

5.2. Desarrollo de la aplicación de ejemplo

Aunque el modelo de tareas descrito en la sección 5.1 es básico y directo, la implementación de la aplicación de ejemplo proporciona las bases para explotar la variedad de configuraciones con las cuales es posible personalizar los servicios proporcionados. En las siguientes secciones se describe a detalle cada uno de los aspectos técnicos y puntos clave detectados para la construcción de la aplicación y la extensión de la funcionalidad de la presente propuesta.

5.2.1. Interfaz gráfica de usuario

Como se mencionó en la sección 4.2.2, el espacio de trabajo utilizado tanto en el modo de trabajo individual como en el colaborativo, está representado por un objeto perteneciente a la clase `View`.

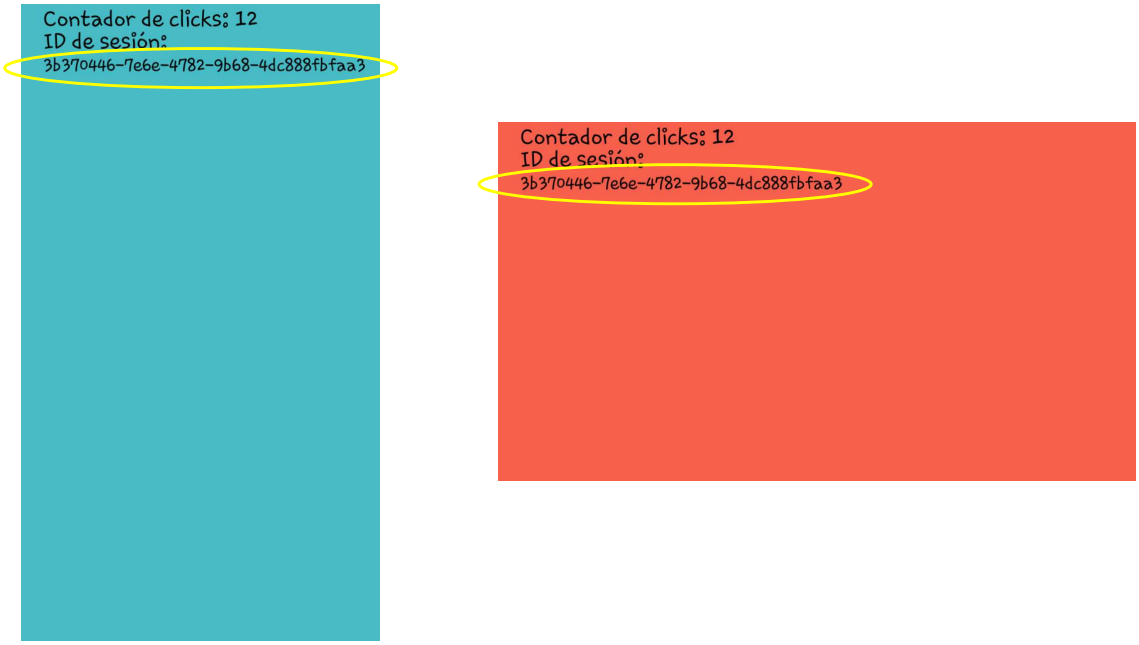

Figura 5.2: Modo de trabajo colaborativo en dos dispositivos distintos recién acoplados, demostrando la sincronización del identificador de sesión y contador de clicks cortos

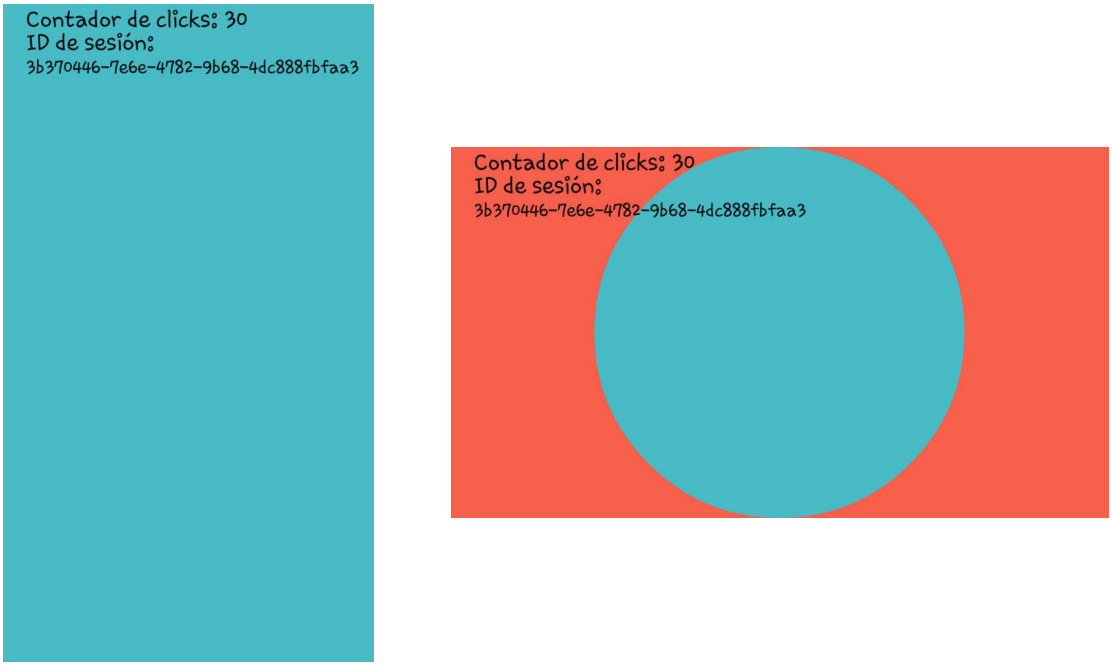

Figura 5.3: Modo de trabajo colaborativo en dos dispositivos distintos, demostrando la continuidad de sincronización del contador de clicks cortos y la detección de qué dispositivo detectó el último click corto (la pantalla de la izquierda en este ejemplo)

Estos objetos son comúnmente utilizados para la implementación de las interfaces gráficas de usuario, ya que están compuestos por áreas rectangulares sobre las cuales se dibujan y controlan los eventos que se disparan sobre ellas. En realidad, `View` es la clase base en Android para la construcción de *widgets* (componentes interactivos listos para ser utilizados en la construcción de las interfaces gráficas de usuario, tales como botones, cuadros de texto, etiquetas, etc.). No obstante, es posible extender dicha clase para aumentar el comportamiento típico de un objeto `View`, proporcionándole una funcionalidad más robusta y atípica (comparada con la de un *widget*).

El objeto de la clase **View** utilizado como espacio de trabajo abarca la totalidad de la pantalla del dispositivo en todo momento. Sobre dicho objeto, se dibujan los elementos requeridos mediante un objeto de tipo **Canvas**, el cual permite trazar primitivas en forma de figuras, líneas o caminos, texto, imágenes, así como la posibilidad de asignar colores y estilos a dichas primitivas.

La interfaz gráfica está compuesta únicamente por un objeto de la clase **View**, sobre el cual se dibujan los siguientes elementos (figura 5.1):

- Contador de clicks
- Identificador de sesión
- Color de fondo aleatorio

Adicionalmente, cuando se activa el modo de trabajo colaborativo, como resultado de un acoplamiento exitoso, se remodela la interfaz gráfica de usuario mostrando un círculo con un color distinto, el cual corresponde al color de fondo del dispositivo desde el cual se detectó el último click corto (figura 5.3).

5.2.2. Detalles de implementación

Para el desarrollo de la aplicación, básicamente se crearon cinco clases nuevas así como una interfaz y se llevó a cabo una serie de modificaciones en ciertas clases de los servicios. Los nuevos recursos implementados tienen como propósito controlar la lógica de la aplicación y agregar algunos notificadores nuevos. Por otro lado, las modificaciones en el soporte permiten ajustar los servicios para procesar correctamente dichos notificadores.

A continuación, procedemos a describir cada uno de los recursos y modificaciones que forman parte de la lógica de la aplicación:

- **GerberaExample**: clase principal en donde se preparan los servicios del soporte y se proporciona una implementación para cada uno de los notificadores.
- **Board**: concretamente, es una clase que extiende de **View** y representa el espacio de trabajo. Dentro de esta clase, se llevan a cabo las operaciones de dibujo y se controla la lógica que permite disparar el procedimiento de remodelación. Los algoritmos 17 y 18 muestran los pasos a seguir para la inicialización de la aplicación y el funcionamiento interno de la clase **Board**, respectivamente.

  En el algoritmo 17, es posible apreciar que utilizamos una instancia de la clase **RegularGerberaSetup** (línea 2), aunque con ciertas modificaciones para que se pueda implementar

Algoritmo 17 Lógica de la clase **GerberaExample** para la inicialización de la aplicación

1: Crear una instancia de la clase **Board**
2: Crear una instancia de la clase **RegularGerberaSetup**
3: Implementar los métodos `discoveryServiceDeviceJoined(...)` y `discoveryServiceDeviceLeft(...)` de la interfaz **DiscoveryServiceListener**
4: Implementar el método `workspaceTapEventHappened(...)` de la interfaz **WorkspaceTouchEventListener**
5: Implementar el método `checkObjectReceived(...)` de la interfaz **UpdateDistributionServiceListener**
6: Implementar los métodos `retrieveCounter()`, `setCounter(...)`, `retrieveSessionID()` y `setSessionID(...)` de la interfaz **CustomListener**
7: Agregar la llamada a los métodos `onPause()`, `onStop()` y `onConfigurationChanged(...)` a través de la instancia de la clase **RegularGerberaSetup** en los eventos correspondientes

la interfaz adicional **CustomListener**. Dicha interfaz provee los métodos y funciones necesarias para poder manipular variables que no pertenecían originalmente al soporte dentro de cualquiera de los servicios (línea 6), pero que deberán ser sincronizados durante el protocolo de acuerdo para la distribución de la lista de grupo (ver capítulo 4, sección 4.2.2). El resto de los métodos son parte de la versión original de las interfaces explicadas en el capítulo 4, sección 4.2.

Algoritmo 18 Lógica de la clase **Board** para el desplegado de la interfaz gráfica

1: Generar color aleatorio y establecerlo como fondo del espacio de trabajo
2: `counter` ← Inicializar el contador de clicks en 0
3: `sessionID` ← Generar un identificador de sesión
4:
5: **procedure** ANIMATEBIGCIRCLE(`color`)
6: Remover la transparencia del círculo
7: Llamar al método `onDraw(...)`
8: **end procedure**
9:
10: **procedure** ONDRAW(`canvas`)
11: Dibujar un círculo de color `color`
12: Colocar el texto referente al contador y su respectivo valor `counter`
13: Colocar debajo del contador, el identificador de sesión `sessionID`
14: **end procedure**

En cuanto al algoritmo 18, primero se genera el color aleatorio y el identificador de sesión, ya que ambos se utilizan como parámetros posteriormente. Después, se definen dos métodos: `animateBigCircle(color)` y `onDraw(canvas)`. El primero de ellos es ejecutado cuando se recibe un objeto de tipo **Net_Message** y sirve para mostrar un círculo de un cierto color. El segundo método es propio de cualquier objeto de tipo **View**, pues se encarga de redibujar la GUI cuando sea necesario. Adicionalmente, coloca el valor del contador de clicks y el identificador de sesión en una parte visible de dicha interfaz gráfica.

Continuando con la lista de los recursos agregados y las modificaciones requeridas como parte de la implementación de la aplicación de ejemplo:

- **Net_Message**, **Net_Counter** y **Net_SessionID**: son básicamente clases contenedoras de los objetos a enviar por la red, que forman parte de la lógica de la aplicación. Siguiendo la pauta recomendada para el soporte, estas clases se encuentran almacenadas dentro del paquete *gerberal.services.updatedistribution.networkclasses*. Internamente, sirven para transmitir respectivamente: el contador de clicks `counter`, el identificador de la sesión `sessionID` y el color `color` generado aleatoreamente.

- **CustomListener**: es un notificador que sirve para permitir la obtención y establecimiento de `counter` y `sessionID` en lugares específicos durante el protocolo para la actualización de la lista de grupo (ver capítulo 4, sección 4.2.2). Esta interfaz es necesaria si se requiere mover el valor de un recurso entre el soporte y la aplicación de ejemplo de forma indistinta y sin realizar cambios significativos en el código original de los servicios. Es posible utilizar el patrón de diseño *Observer* (ver sección 5.3.2) para mover el valor de otros recursos que requieran ser enviados o recibidos para su posterior sincronización entre todos los dispositivos.

- **Cambios en el protocolo de actualización de la lista de grupo**: esta nueva configuración es una forma típica de modificar el contenido de los mensajes utilizados durante el protocolo de acuerdo. Cada vez que se requiera sincronizar el valor o contenido de uno o más recursos, esta es la forma más recomendable de hacerlo, ya que las modificaciones en la implementación del soporte propuesto son mínimas. El algoritmo 19 muestra los pasos a seguir para modificar los servicios del soporte en forma correcta:

Algoritmo 19 Ajuste del protocolo de actualización de la lista de grupo

1: Definir la interfaz **CustomListener**, la cual contiene los métodos que actúan a modo de *listeners* y sirven para establecer u obtener el valor de una variable o recurso
2: Modificar el constructor de la clase **RegularGerberaSetup** para que acepte un objeto que implemente la interfaz **CustomListener**
3: Modificar la función `validateGesture(...)` de cada gesto de acoplamiento (clases **SimplePinchGesture** y **SimpleAccelerometerGesture**), de tal forma que reciba como parámetro adicional, el objeto que implementa **CustomListener** y que fue pasado a la instancia de la clase **RegularGerberaSetup**
4: Modificar la clase **ListenOnePairingRequest** para que pueda recibir la instancia que implementa **CustomListener**, la cual ha sido pasada de una clase a otra en los pasos anteriores
5: Agregar la variable a sincronizar como elemento de las clases **Net_RequestForBecomingServer** y **Net_ResponseToRequestFromBecomingServer**; por cada variable agregada, debe existir una función para obtener y establecer el valor de dicha variable en la interfaz **CustomListener**
6: Obtener el valor de una o más variables a sincronizar, mediante los métodos correspondientes de la interfaz personalizada y agregarlos a **Net_RequestForBecomingServer** cuando se dispare un intento de acoplamiento
7: Cuando se recibe el o los valores de las variables a sincronizar desde otro dispositivo

con el cual se está llevando a cabo el acoplamiento, determinar en qué dispositivo(s) es necesario actualizar las variables
8: **if** Se requiere actualizar en este dispositivo **then**
9: Establecer los valores recibidos de las variables a sincronizar
10: Informar a los dispositivos de la sesión anterior que deben actualizar sus valores
11: **else if** Se requiere actualizar en el otro dispositivo **then**
12: Comunicarle que debe realizar lo indicado en la línea 10, añadiendo los valores que debe establecer en el objeto de respuesta **Net_ResponseToRequestFromBecomingServer**
13: **end if**
14: Proseguir con el protocolo tal y como se describió en la sección 4.2.2

Como es posible apreciar, el algoritmo 19 introduce múltiples cambios y da pie a una variedad de alternativas para hacer más robusto este protocolo. La interfaz personalizada lleva como nombre **CustomListener** dentro de la aplicación de ejemplo, sin embargo, se pueden definir tantas interfaces como sean necesarias. Así mismo, contamos con dos recursos que requieren ser sincronizados (el contador de clicks `counter` y el identificador de sesión `sessionID`). Para ello, definimos sus respectivos métodos para establecer y obtener sus valores (comúnmente llamados *setters* y *getters*), los cuales llevan por nombre `setCounter(...), retrieveCounter(), setSessionID(...)` y `retrieveSessionID()`.

En este caso como se mencionó al inicio de la presente sección, hacemos uso de la clase auxiliar **RegularGerberaSetup**, la cual modificamos de tal forma que acepte como parámetro adicional, un objeto de una clase que implemente nuestra interfaz personalizada. En realidad, la clase que nos interesa que reciba nuestra interfaz como parámetro es **ListenOnePairingRequest**, sin embargo, como hacemos uso de la envoltura que proporciona **RegularGerberaSetup**, es necesario propagar la interfaz **CustomListener** por cada una de las clases hasta llegar a **ListenOnePairingRequest** (líneas 2-4). Una vez que la interfaz ha sido ligada, utilizamos las funciones *getter* (`retrieveCounter()` y `retrieveSessionID()`) para obtener los valores de las variables (justo antes de que comience el protocolo de actualización de la lista de grupo) y los agregamos al objeto que sirve para indicar al otro dispositivo que está por comenzar dicho protocolo (líneas 5 y 6).

La siguiente modificación se lleva a cabo en la clase interna **UpdateDistributionTCPServer** (encargada de controlar el servidor local) en donde se deben extraer los valores de las variables a sincronizar del objeto que sirvió como petición para iniciar el protocolo, i.e., de **Net_RequestForBecomingServer**. Posteriormente se comparan los valores recibidos con los propios y se determina si debe existir una actualización de `counter` o `sessionID`. El procedimiento de comparación puede variar dependiendo del tipo de variable; para la aplicación de ejemplo basta con comparar si el contador recibido es mayor al propio y verificar el resultado de la función `compareTo(...)` (perteneciente a la clase **UUID**, ver sección 5.3.4) la cual se encarga de comparar dos identificadores y de determinar cuál es el mayor. Dependiendo del resultado de dichas comparaciones, se determina qué variable se debe actualizar y en qué dispositivo. Para llevar a cabo dicho procedimiento, se utiliza la función *setter* de cada variable (`setCounter(...)` y `setSessionID(...)`). Si la actualización debe ser local, entonces se ejecutan las funciones *setter* correspondientes en caso de que aplique

(línea 9). Por otro lado, si la actualización corresponde al dispositivo que está en espera del objeto de respuesta **Net_ResponseToRequestFromBecomingServer**, entonces se agregan los valores de `counter` y `sessionID` para su respectiva actualización cuando el dispositivo en espera reciba la respuesta (línea 12). Cabe mencionar que en todo momento, todos los dispositivos deben tener exactamente el mismo valor del contador e identificador de sesión. Es por ello que las funciones *setter* se deben encargar también de distribuir el nuevo valor de una variable sincronizada a todos los dispositivos acoplados, a través del método `spreadObject(...)` que proporciona el servicio de distribución de actualizaciones (línea 10). Como nota adicional, la aplicación de ejemplo refresca la GUI cada vez que ocurre un proceso de actualización del contador de clicks o del identificador de la sesión, esto por motivos de coherencia visual.

A modo de resumen, las siguientes clases deben ser modificadas si se requiere un proceso de sincronización como el ya descrito:

- **RegularGerberaSetup**
- **SimpleAccelerometerGesture**
- **SimplePinchGesture**
- **UpdateDistributionService**
- **ResponseToRequestFromBecomingServer**
- **RequestForBecomingServer**

5.3. Versatilidad del soporte propuesto

Es importante recalcar que el único punto de acceso al soporte no es la clase auxiliar **RegularGerberaSetup**. Es posible tomar cada una de las clases que componen a los servicios y ajustarlas de la manera que resulte más conveniente para el modelo de tareas que se requiera implementar. De esta forma, los desarrolladores tienen la posibilidad de utilizar los servicios que requieran (sin necesidad de utilizar los tres) y agregar diferentes capas de desarrollo (e.g., seguridad, protocolos de comunicación, etc.) para hacer más robusta su arquitectura. Es por ello que la presente propuesta separa la lógica de cada uno de los servicios, de tal forma que resulte completamente modular.

5.3.1. Componentes típicos para el despliegue de interfaces gráficas

Uno de los objetivos más importantes al momento de diseñar la interfaz gráfica de una aplicación, es el de conservar un alto nivel de usabilidad, tomando en cuenta que existe una gama muy grande de dispositivos móviles. Es por ello que se debe seleccionar el componente visual que mejor se adapte al modelo de tareas de una aplicación. Como mencionamos en la sección 5.2.1, la familia de clases que hereda de **View** son los bloques de construcción más básicos con los cuales se implementan interfaces gráficas de usuario. Existen mútliples especializaciones de dicha clase, tales como:

- **SurfaceView**: es una superficie de dibujo dedicada que está situada debajo de la superficie regular sobre la cual se colocan los objetos visuales tradicionales. Uno de los propósitos de esta clase es proporcionar una superficie sobre la cual un hilo secundario pueda dibujar tan rápido como le sea posible (i.e., no interviene el hilo principal o *UI thread*). Si se requieren gráficos personalizados y muy dinámicos, esta clase es una buena opción.
- **ViewGroup**: es un tipo de objeto especial, ya que tiene como objetivo contener y acomodar otros objetos que hereden de la clase **View**, mas no es utilizada para desplegar gráficos directamente. Esta clase puede ser usada como contenedor de distintas partes del espacio de trabajo (*widgets*). Si la interfaz gráfica es mayormente estática y no se requiere una gran cantidad de actualizaciones de pantalla, **ViewGroup** puede ser utilizada correctamente.
- **ImageView**: su propósito es básicamente mostrar una imagen en pantalla, la cual puede ser cargada de distintas maneras y con diferentes configuraciones.
- **TextureView**: al igual que **ImageView**, esta clase tiene un único propósito, el cual consiste en mostrar el contenido de un flujo de datos, e.g., un video o una escena de OpenGL.
- Entre otros más.

La única condición que impone el soporte, con respecto al objeto que representará el espacio de trabajo, es que herede de la clase **View** y, por lo tanto, pueda detectar eventos de tipo *touch*. En cuanto a las demás características, dependerá de los desarrolladores elegir la vista más adecuada.

Inicialmente se desarrolló un prototipo sencillo con conectividad de red básica, que empleaba únicamente **TextViews** como componentes gráficos, debido a que son fácilmente configurables y ya tienen implementado el comportamiento deseado. El modelo de tareas del usuario estaba compuesto por las siguientes actividades:

- Crear elementos dentro del espacio de trabajo mediante un click largo sobre la pantalla.
- Arrastrar los elementos creados dentro de la zona de dibujo para colocarlos en otra posición.
- Mover la zona de dibujo para apreciar diferentes partes de espacio del trabajo.

El objetivo de las pruebas iniciales con dicho prototipo fue determinar el impacto sobre dispositivos con una menor capacidad de procesamiento (utilizando componentes que permitieran un rápido desarrollo, tales como los objetos de la clase **TextView**). En realidad no resultó necesario agregar más tareas al sistema (e.g., sincronización con otros dispositivos o soporte completo de gestos requeridos por la aplicación) para extender las pruebas y obtener un resultado más completo, ya que utilizando únicamente **TextViews** como bloques de construcción del soporte, dispositivos con un poder de procesamiento bajo reflejaron una reducción de respuesta del sistema al arrastrar elementos o mover la zona de dibujo después de agregar alrededor de 32 elementos al espacio de trabajo, lo cual confirma que el uso de estos componentes ve reducida la experiencia del usuario en ambientes muy dinámicos.

5.3.2. Notificadores

Los notificadores o *listeners* son parte del patrón de diseño *Observer* [Gamma et al., 1995]. Debido a que el soporte está basado completamente en eventos que disparan ciertas acciones, existe una amplia gama de oportunidades para tomar ventaja de este patrón. En la sección 5.2.2 presentamos solamente una de las posibles alternativas que existen para agregar notificadores al soporte, adicionales a los ya proporcionados:

- Detección de eventos de tipo *touch* sobre el espacio de trabajo.
- Adición o eliminación de un dispositivo a la lista de descubrimiento.
- Recepción de un objeto en el servidor local.

A continuación listamos algunos eventos que podrían ser de interés para la implementación de algún tipo de notificador:

- Adición o eliminación de un dispositivo de la lista de acoplamiento.
- Finalización de un gesto de acoplamiento correcto o incorrecto.
- Actualización de las características de un dispositivo acoplado o de las características propias.
- Inicialización o terminación de uno o más servicios.
- Entre otros más.

5.3.3. Espacio de trabajo

Es importante hacer la siguiente aclaración con respecto al espacio de trabajo en sus variantes individual y compartida: aunque físicamente la distinción entre un tipo de espacio y otro está definida por las pantallas de los dispositivos, el concepto de "espacio de trabajo compartido" puede ser extendido más allá de dichos límites. Dependiendo de los objetivos que se tengan para habilitar el acoplamiento de dos o más dispositivos, puede ser que se requiera o no visualizar exactamente el mismo contenido (o parte de) en cada uno de los dispositivos. De esta forma, definimos tres tipos de visualizaciones soportadas por la presente propuesta:

- **Única**: cuando todos los dispositivos muestran exactamente la misma parte del espacio de trabajo, permitiendo a los usuarios finales interactuar con los mismos objetos virtuales de forma indistinta en diferentes pantallas. El modelo de tareas en este tipo de visualización es uniforme, i.e., todos los dispositivos permiten llevar a cabo el mismo conjunto de tareas.
- **Complementaria**: sucede cuando se muestra una parte distinta del espacio de trabajo y de la interfaz gráfica de usuario en cada una de las pantallas, de tal forma que sea posible interactuar con distintos objetos virtuales o partes del espacio de trabajo y llevar a cabo distintas tareas dependiendo del dispositivo con el cual se interactúe. Un ejemplo de este tipo de visualización podría ser una aplicación que muestre diferentes partes del mismo mapa en cada una de las pantallas acopladas y que, a través de un gesto especial, se pueda

seleccionar alguno de los dispositivos para mostrar una interfaz gráfica mediante la cual se puedan llevar a cabo acciones específicas, como cargar otros mapas, buscar sitios de interés o cambiar la vista de mapa (e.g., real, satelital o reconstrucción mediante modelos 3d).

- **Fragmentada**: similar a la visualización complementaria, pero sin la dispersión del modelo de tareas (e.g., una aplicación que permita dibujar sobre las pantallas de los dispositivos y desplazar el lienzo de dibujo en cualquier dirección).

La visualización del espacio de trabajo está relacionada directamente con una de las dimensiones de las interfaces plásticas: los medios de adaptación. Esto implica que, de forma reactiva y posterior a un acoplamiento de dispositivos, ocurrirá un procedimiento de remodelación y redistribución plástica (ver sección 2.2). En la aplicación de ejemplo, los medios de adaptación consisten básicamente en un procedimiento de **adaptación** mediante visualización de tipo única.

Por otro lado, independientemente del tipo de visualización que se requiera, el espacio de trabajo sigue siendo único a lo largo de una sesión colaborativa, i.e., todos los dispositivos acoplados tienen almacenados exactamente los mismos objetos virtuales.

5.3.4. Identificadores de sesión

En la aplicación de ejemplo, utilizamos identificadores de sesión de tipo **UUID** (identificadores únicos universales) [IETF, 1981]. Dichos identificadores son números de 128 bits en formato hexadecimal, los cuales pueden ser generados por cualquier sistema, con la característica de que son difícilmente repetibles (la aplicación de ejemplo tiene una validación en este sentido; si por alguna razón la función de comparación determina que el identificador propio y el recibido son exactamente iguales, entonces se genera uno nuevo para ambos dispositivos y, por consecuencia, para todos aquellos dispositivos acoplados). Existen diferentes versiones de identificadores **UUID**, sin embargo, en Java solamente es posible generar identificadores versión 3 (basados en una URL) y 4 (basados en un generador pseudo-aleatorio). La aplicación de ejemplo hace uso de identificadores **UUID** versión 4, ya que son generados automáticamente y tienen la misma garantía (difíciles de repetir).

Capítulo 6

Pruebas finales y resultados

A continuación, se presentan los detalles de las pruebas realizadas a usuarios finales, con el fin de validar diversos aspectos del soporte desarrollado. Los resultados son presentados visualmente (con el fin de permitir una rápida inspección) junto con sus respectivas interpretaciones. Adicionalmente, se introducen los cuestionarios aplicados y las herramientas utilizadas para realizar cada una de las validaciones descritas.

6.1. Pruebas finales

Con el fin de evaluar diversos aspectos del soporte, así como de identificar posibles mejoras con respecto a los métodos de interacción y las interfaces de usuario en general (gráficas y ubicuas), se llevaron a cabo tres cuestionarios distintos con diferentes grupos de usuarios. A continuación iremos detallando cada uno de ellos, junto con los resultados obtenidos.

6.1.1. Selección de los gestos de acoplamiento

El primer cuestionario se realizó en etapas tempranas de desarrollo, el cual tenía como objetivo consultar con usuarios finales cuáles de los siguientes gestos de acoplamiento consideraban más intuitivos y, por lo tanto, les gustaría ver implementados:

a) Desplazar ambos dispositivos en direcciones opuestas con un movimiento rectilíneo hasta que dos de su lados hagan contacto.

b) Con un dedo sobre la pantalla de cada dispositivo, realizar un gesto concurrente en línea recta que indique cuáles serán los lados por los cuales se acoplarán los dispositivos.

c) Una combinación de los dos gestos annteriores, i.e., que se puedan utilizar de forma indistinta.

Adicionalmente, se les daba la posibilidad de proporcionar algún gesto de acoplamiento distinto o agregar alguna sugerencia adicional. Dicho cuestionario se aplicó a un grupo de 28 usuarios, obteniendo los resultados mostrados en la figura 6.1.

Cabe mencionar que la base de usuarios encuestada está familiarizada con el uso de dispositivos móviles, además, no se agregaron gestos adicionales a la gráfica ya que no se proporcionaron

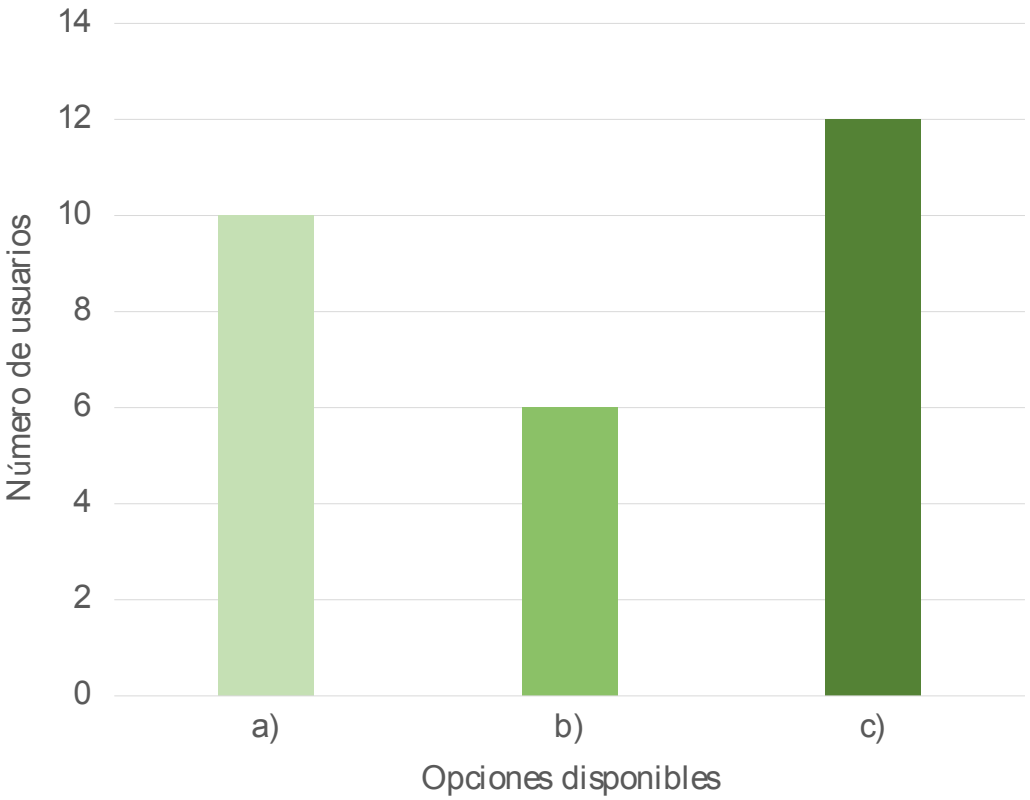

Figura 6.1: Gestos de acoplamiento preferidos por un grupo de 28 usuarios

sugerencias.

Como es posible apreciar, la mayoría considera la opción **a)** como la más intuitiva, seguida de la utilización combinada de ambos gestos (opción **c)**) como segundo lugar y, finalmente, el gesto de arrastre sobre las pantallas de los dispositivos. Aunque los resultados obtenidos muestran la opción **a)** como la preferida, se determinó no incluirla en el soporte por las razones expuestas en la sección 3.3.1. Es por ello que se sustituyó el gesto de desplazamiento por el gesto de inclinación (el cual hace uso del acelerómetro y, por consiguiente, pertenece a la misma categoría que el gesto de desplazamiento, i.e., aquellos basados en el uso de sensores consientes de contexto), respetando la preferencia de los usuarios por mantener dos gestos de acoplamiento disponibles al proporcionar sus implementaciones en las clases **SimpleAccelerometerGesture** y **SimplePinchGesture**.

6.1.2. Evaluación de los gestos de acoplamiento

El segundo cuestionario está relacionado con la evaluación de los gestos de acoplamiento. Para ello, se utilizó una aplicación básica de prueba en la cual era posible probar ambos gestos implementados. Básicamente, cada vez que un usuario realizaba un gesto de acoplamiento en dos dispositivos, la aplicación de prueba respondía mediante retroalimentación audible si los gestos habían sido o no procesados y si habían sido correctos. Cada usuario era libre de experimentar con la aplicación, probando los gestos (y sus combinaciones) hasta que sintieran más confianza al llevarlos a cabo y entendieran completamente la dinámica de cada uno de estos.

El cuestionario utilizado está basado en un índice de carga de trabajo denominado *NASA Task Load Index* (NASA TLX por sus siglas en inglés), el cual nos permitió evaluar la carga de trabajo desde diferentes aspectos (6 dimensiones):

- Demanda mental: ¿cuánta actividad mental y perceptual fue necesaria para comprender el funcionamiento del acoplamiento de dispositivos?
- Demanda física: ¿cuánta actividad física se requiere para controlar los gestos de acoplamiento?
- Demanda temporal: ¿cuánta presión de tiempo se sintió debido al ritmo con el cual se efectuaron los gestos de acoplamiento?
- Desempeño general: ¿cuán exitosamente fue realizado el acoplamiento y qué tan satisfecho se siente con su desempeño?
- Nivel de frustación: ¿cuál fue su nivel de frustación durante la realización del acoplamiento?
- Esfuerzo requerido: ¿qué tan arduamente es necesario trabajar (física y mentalmente) para lograr el nivel de desempeño deseado?

El usuario asigna entonces un puntaje y un peso a cada aspecto, con los cuales podemos determinar la carga total que conlleva utilizar cada una de los gestos de acoplamiento (y sus combinaciones). Esto implica que son los usuarios quienes determinan la importancia de cada una de las dimensiones evaluadas; de esta forma, la prueba se vuelve mucho más precisa. Dicho cuestionario se aplicó tres veces a cada usuario (5 usuarios en total), uno por cada opción (gesto de arrastre, gesto de inclinación y su uso combinado).

La primera evaluación corresponde al gesto de arrastre; la figura 6.2 muestra el puntaje ajustado de cada uno de los participantes para cada una de las dimensiones. Dicho puntaje se obtuvo de la siguiente manera:

$$Puntaje\ ajustado = puntaje\ x\ peso$$

En donde el puntaje es un valor dentro del rango [0, 100] y el peso un valor dentro de [0, 5] (i.e., que el máximo puntaje ajustado posible es 500). De esta forma, se puede cuantificar la experiencia de un usuario al evaluar cada uno de los gestos y su respectiva combinación.

A continuación se interpretan los resultados obtenidos como producto de la evaluación del acoplamiento de dispositivos utilizando el gesto de arrastre (ver las tablas 6.1 y 6.2 para un desglose), i.e., cuando los usuarios acoplaron dos o más dispositivos mediante el mismo gesto:

- **Demanda mental**: los usuarios comprendieron rápidamente el funcionamiento del gesto de arrastre, lo cual se ve reflejado en un puntaje muy bajo de demanda mental y, aunque la desviación estándar nos indica que existieron variaciones en cuanto a los puntajes individuales, ninguno de ellos fue lo suficientemente representativo como para determinar que el gesto evaluado pudiera ser considerado como difícil de comprender.
- **Demanda física**: al igual que la dimensión anterior, se obtuvo un puntaje bajo en este aspecto. Independientemente de la variación de puntajes, en general los usuarios determinaron que para efectuar el gesto de arrastre, no se requiere de una gran demanda física.

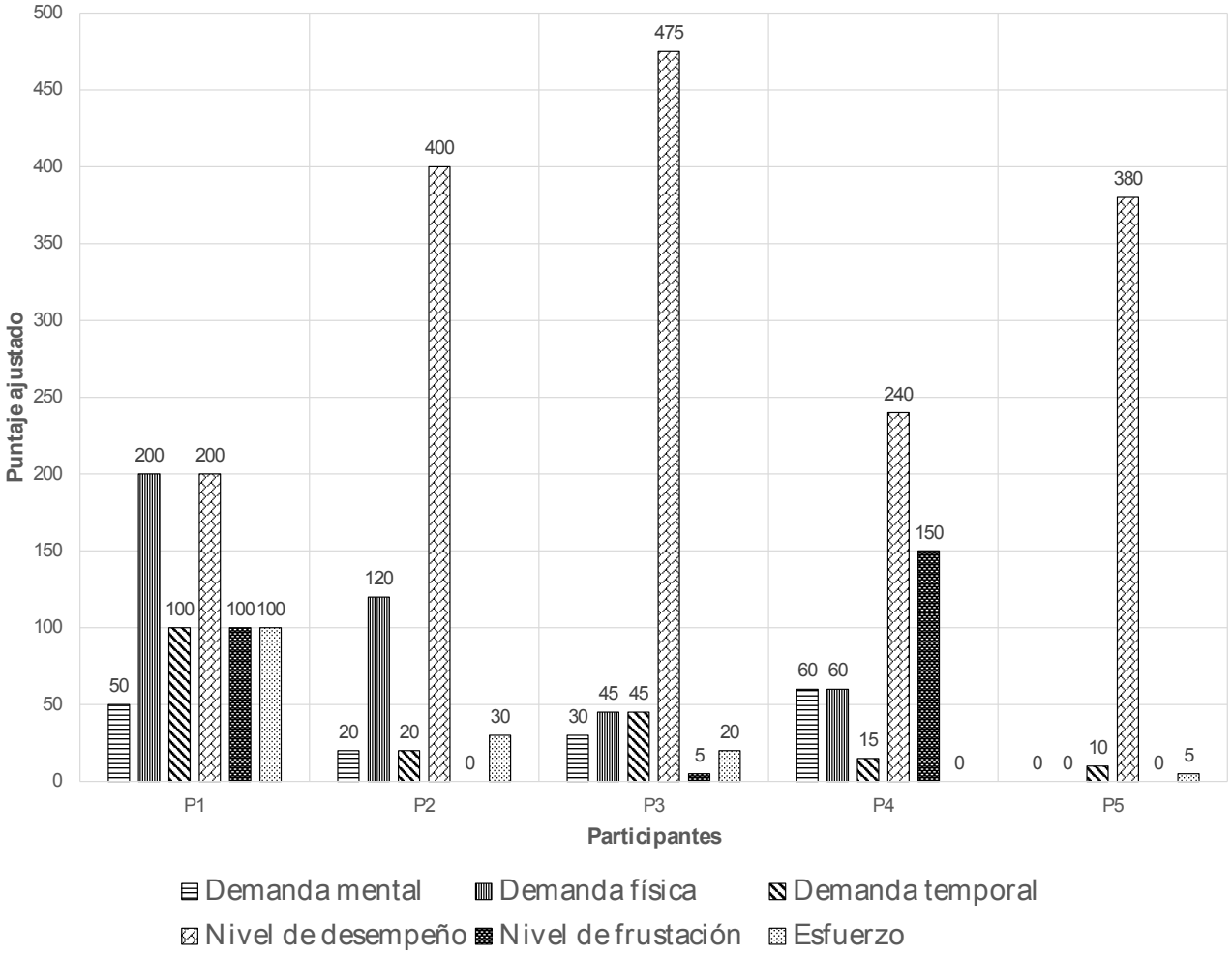

Figura 6.2: Puntaje ajustado para cada una de las 6 dimensiones evaluando el gesto de arrastre

- **Demanda temporal**: el gesto de arrastre no imprimió presión temporal considerable, esto de acuerdo a los puntajes registrados.
- **Desempeño general**: la mayoría de los usuarios determinó que utilizar el gesto de arrastre como medio para llevar a cabo un acoplamiento, es suficiente y cumple con sus expectativas.
- **Nivel de frustación**: utilizar este gesto imprime un bajo nivel de frustación (si acaso llegara a existir), lo cual está directamente relacionado con una tasa de éxito alta al acoplar dos dispositivos.
- **Esfuerzo requerido**: tanto física como mentalmente, los usuarios consideraron que el esfuerzo requerido para realizar el gesto y acoplar dos dispositivos es mínimo.
- **Carga de trabajo total**: este puntaje fue sumamente bajo y uniforme para todos los usuarios, lo cual implica que el gesto de arrastre puede ser usado fácil y eficientemente para acoplar dispositivos, con la certeza de que se tendrá una tasa de éxito alta.

Tabla 6.1: Desglose de resultados para el gesto de arrastre (1)

Usuarios	**Demanda mental**	**Demanda física**	**Demanda temporal**	**Desempeño general**	**Nivel de frustración**	**Esfuerzo requerido**
P1	50	200	100	200	100	100
P2	20	120	20	400	0	30
P3	30	45	45	475	5	20
P4	60	60	15	240	150	0
P5	0	0	10	380	0	5
$\overline{x}$	32	85	38	339	51	31
σ	23.87	77.3	37.18	115.13	69.86	40.37

Tabla 6.2: Desglose de resultados para el gesto de arrastre (2)

Usuarios	**Carga de trabajo total**
P1	50
P2	39.3
P3	41.3
P4	35
P5	26.3
$\overline{x}$	38.4
σ	8.68

El siguiente gesto evaluado fue el gesto de inclinación (ver figura 6.3). De acuerdo al desglose de las tablas 6.3 y 6.4, es posible concluir lo siguiente:

Tabla 6.3: Desglose de resultados para el gesto de inclinación (1)

Usuarios	**Demanda mental**	**Demanda física**	**Demanda temporal**	**Desempeño general**	**Nivel de frustración**	**Esfuerzo requerido**
P1	70	350	70	280	150	40
P2	30	150	10	300	10	200
P3	20	60	45	450	10	10
P4	105	165	0	120	20	275
P5	0	5	20	300	0	0
$\overline{x}$	45	146	29	290	38	105
σ	42.13	131.6	28.37	117.05	63.01	124.7

- **Demanda mental**: los usuarios comprendieron rápidamente el funcionamiento del gesto de inclinación. Existieron variaciones en cuanto a los puntajes registrados, sin embargo, éstos se mueven en rangos de puntajes bajos.

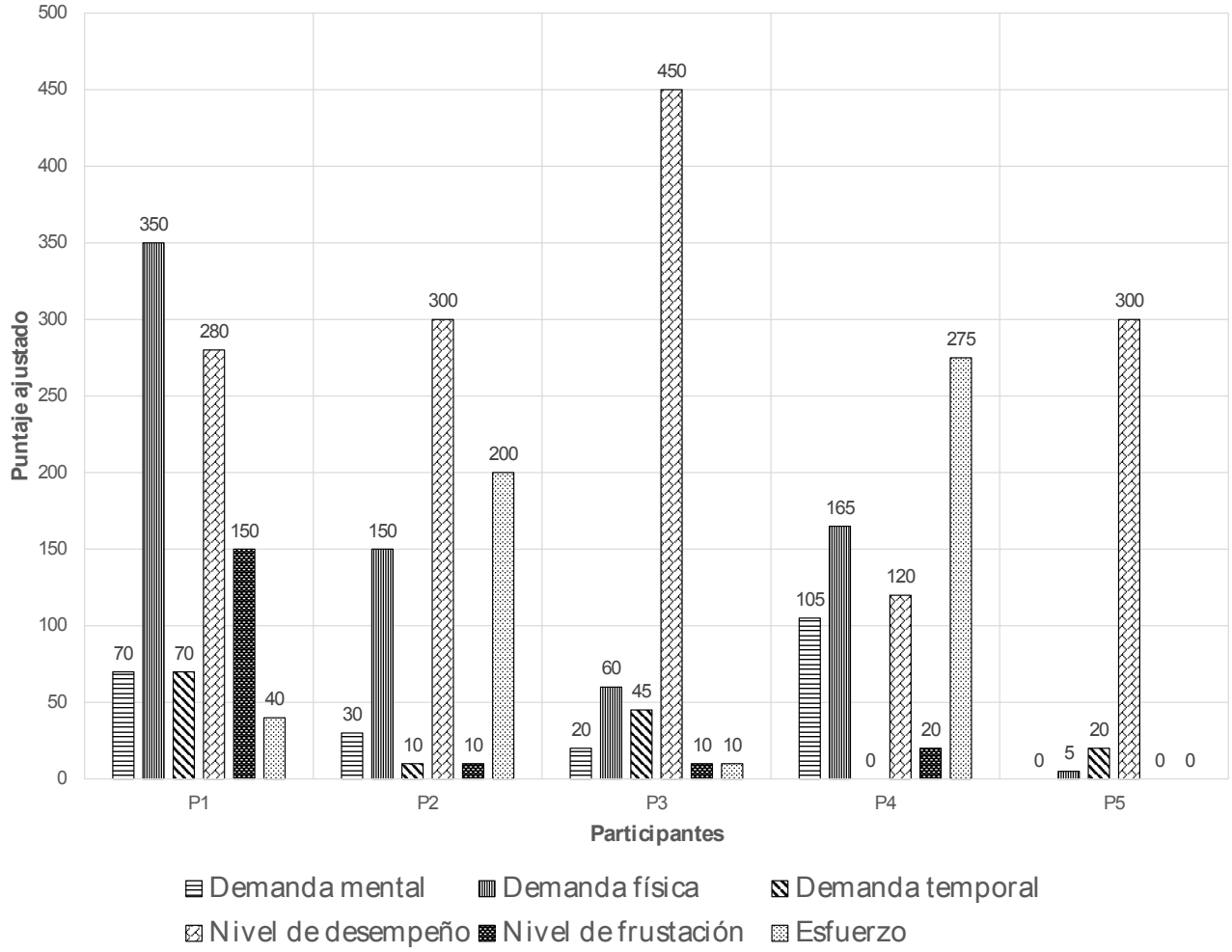

Figura 6.3: Puntaje ajustado para cada una de las 6 dimensiones evaluando el gesto de inclinación

Tabla 6.4: Desglose de resultados para el gesto de inclinación (2)

Usuarios	**Carga de trabajo total**
P1	64
P2	46.7
P3	39.7
P4	45.7
P5	21.7
$\overline{x}$	43.53
σ	15.22

- **Demanda física**: opuesto al gesto de arrastre, el gesto de inclinación requirió una mayor demanda física en los usuarios, aunque no a grados altos.
- **Demanda temporal**: no se detectó la presencia de presión temporal considerable, lo cual

implica que los usuarios pudieron llevar a cabo el gesto y su consecuente acoplamiento sin presiones de tiempo.

- **Desempeño general**: la mayoría de los usuarios determinó que utilizar el gesto de inclinación como medio para llevar a cabo un acoplamiento, es suficiente y también cumple con sus expectativas. El puntaje para este caso fue un poco más bajo (lo cual se debe a que es más complejo de realizar, ya que existen más variables contextuales en juego), sin embargo, sigue siendo lo suficientemente representativo como para considerar que el gesto es eficaz.
- **Nivel de frustación**: prácticamente nulo, solamente uno de los usuarios registró un nivel de frustación perceptible.
- **Esfuerzo requerido**: relacionado directamente con la demanda física y mental, los usuarios requieren de un mayor esfuerzo para llevar a cabo correctamente el gesto de inclinación.
- **Carga de trabajo total**: al igual que con el gesto de arrastre, este puntaje fue sumamente bajo y uniforme para todos los usuarios, lo cual implica que el gesto de inclinación puede ser usado eficientemente para acoplar dispositivos, con la certeza de que se tendrá una tasa de éxito alta. La única consideración a tomar en cuenta, es el esfuerzo requerido para llevarlo a cabo.

Contrario a lo esperado, se registraron puntajes más bajos en ciertos aspectos (comparados con los obtenidos para el gesto de arrastre), tales como el nivel de frustación y demanda temporal. En teoría, el gesto de inclinación es ligeramente más complejo y el esfuerzo requerido para realizarlo correctamente es mayor (tal y como lo hicieron notar los usuarios en el cuestionario), sin embargo, esto no se ve reflejado en el puntaje promedio del nivel de frustación. Adicionalmente, no existió una demanda temporal perceptible, a pesar de que el gesto de inclinación debe llevarse a cabo dentro de un límite de tiempo prestablecido.

El tercer cuestionario recopila los puntajes de cada usuario para cuando se utilizan los gestos de arrastre e inclinación de forma combinada. La figura 6.4 muestra gráficamente los resultados obtenidos; adicionalmente, se agrega su respectivo desglose en las tablas 6.5 y 6.6.

Es posible resumir los resultados obtenidos de la siguiente manera:

- Básicamente, los puntajes para cada una de las dimensiones evaluadas (cuando se utilizan ambos gestos de acoplamiento, uno en cada dispositivo), son muy similares a los dos casos anteriores, sin embargo, la carga de trabajo final es ligeramente mayor para este caso. Esto se puede deber a varios factores, tales como la sincronización de los gestos de acoplamiento (recordando que el gesto de arrastre puede ser tan largo o corto como el usuario desee, a diferencia del gesto de inclinación, el cual muestrea el acelerómetro por un tiempo determinado) o la coordinación motriz para realizar los gestos de forma apropiada en dispositivos heterogéneos (se pueden iniciar al mismo tiempo, pero se nota en los usuarios cierta incertidumbre sobre en qué momento deben terminar el gesto de arrastre para que coincida con el gesto de inclinación).

Cabe mencionar que el click largo tuvo cierto impacto en los resultados obtenidos para el gesto de inclinación y el uso combinado de los gestos. Algunos usuarios manisfestaron que a veces podía

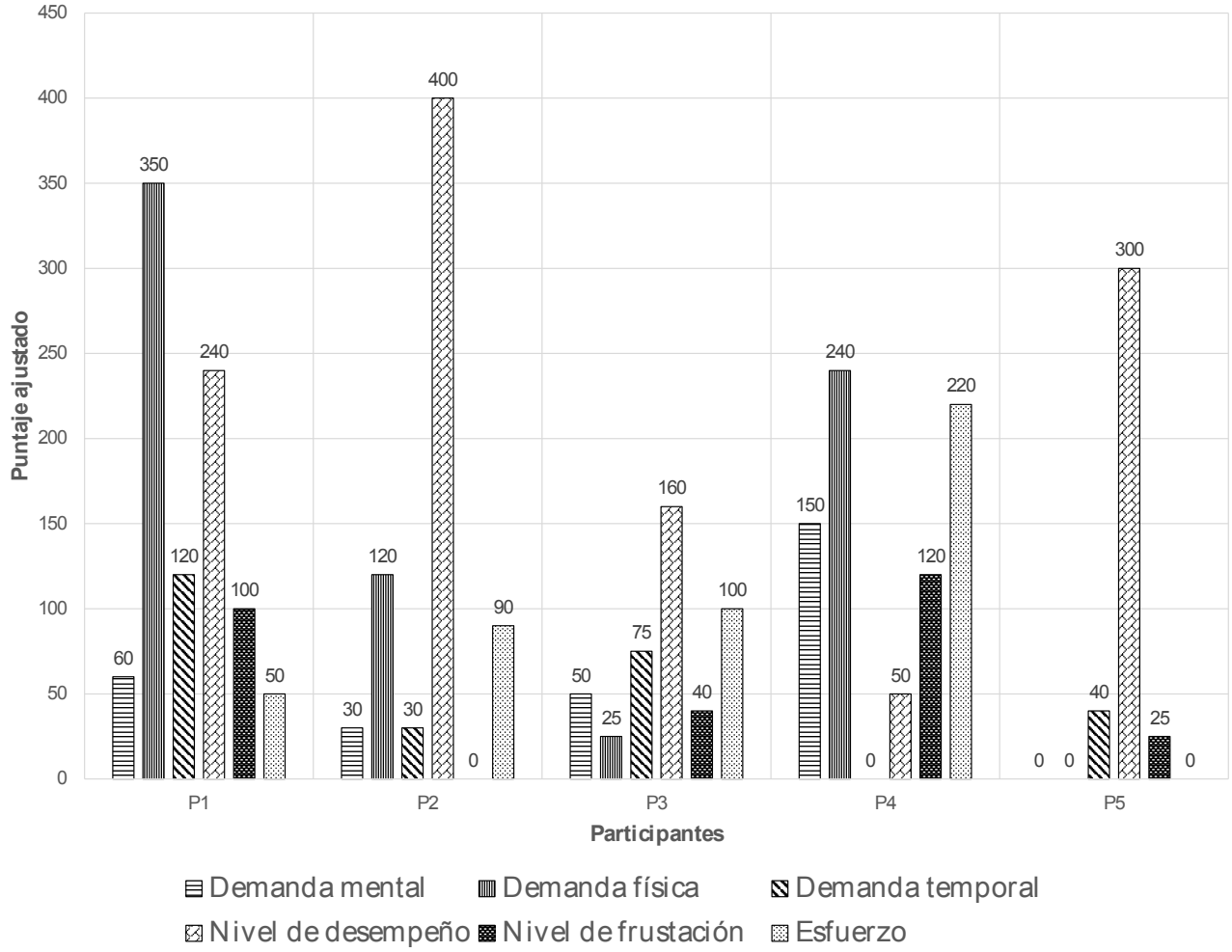

Figura 6.4: Puntaje ajustado para cada una de las 6 dimensiones evaluando los gestos combinados

Tabla 6.5: Desglose de resultados para los gestos combinados (1)

Usuarios	**Demanda mental**	**Demanda física**	**Demanda temporal**	**Desempeño general**	**Nivel de frustración**	**Esfuerzo requerido**
P1	60	350	120	240	100	50
P2	30	120	30	400	0	90
P3	50	25	75	160	40	100
P4	150	240	0	50	120	220
P5	0	0	40	300	25	0
$\overline{x}$	58	147	53	230	57	92
σ	56.3	147.55	46.04	133.42	50.94	81.67

ser confuso el procedimiento para indicarle al dispositivo que se tiene la intención de realizar un gesto de inclinación. Esto se debe a que dicho gesto se activa al momento de levantar el dedo después de realizar un click largo. Para algunos usuarios, era más intuitivo poder inclinar el

Tabla 6.6: Desglose de resultados para los gestos combinados (2)

Usuarios	**Carga de trabajo total**
P1	61.3
P2	44.7
P3	30
P4	52
P5	24.3
$\overline{x}$	42.47
σ	15.3

dispositivo sin tener que levantar el dedo de la pantalla después de realizar un click largo (tal y como sucede con el gesto de arrastre). Este cambio en el comportamiento es posible modificando el método `validateGesture(...)` de la clase **SimpleAccelerometerGesture**.

6.1.3. Evaluación de los criterios de calidad hedónicos y pragmáticos

El último de los objetivos específicos de la presente tesis, consiste en evaluar los criterios de calidad hedónicos y pragmáticos de la interfaz de usuario de la aplicación de ejemplo (ver sección 1.4). Con tal fin, aplicamos el cuestionario propuesto por AttrakDiff^{TM}, el cual es una herramienta utilizada para medir cuán atractivo es un producto interactivo (una interfaz ubicua en nuestro caso). AttrakDiff^{TM} evalúa las siguientes dimensiones:

- **Calidad pragmática (PQ)**: indica la usabilidad de una interfaz y que tan exitosamente pueden cumplir sus metas los usuarios, si utilizan el producto.
- **Calidad hedónica - Estimulación (HQ-S)**: describe que tan novedosa, interesante o estimulante es una interfaz.
- **Calidad hedónica - Identidad (HQ-I)**: indica hasta qué grado se identifica un usuario con la interfaz.
- **Atractivo (ATT)**: es un descriptor del valor de un producto basado en la percepción del usuario.

El cuestionario se aplicó a un grupo de 5 usuarios, todos familiarizados con el uso de dispositivos móviles y sus formas típicas de interación. Básicamente, el procedimiento consistió en explicar a los usuarios el modelo de tareas de la aplicación de prueba (ver sección 5.1) y el uso de los gestos de acoplamiento. Posteriormente, se les propuso probar la aplicación con total libertad, acoplando los dispositivos con los gestos implementados. Una vez finalizada la fase de pruebas, se les proporcionó el cuestionario en línea para que procedieran con la evaluación. Dicho cuestionario consta básicamente de un conjunto de adjetivos opuestos escalados, en donde el usuario debe seleccionar qué tanto se ajusta la experiencia de la prueba de la aplicación de ejemplo con alguno de los adjetivos listados. A continuación, se muestran y describen los resultados obtenidos:
En la figura 6.5 se muestra una gráfica de resultados, en la cual se pueden apreciar dos elementos: la media de las dimensiones evaluadas (cuadrado con una letra P) y un rectángulo de confianza

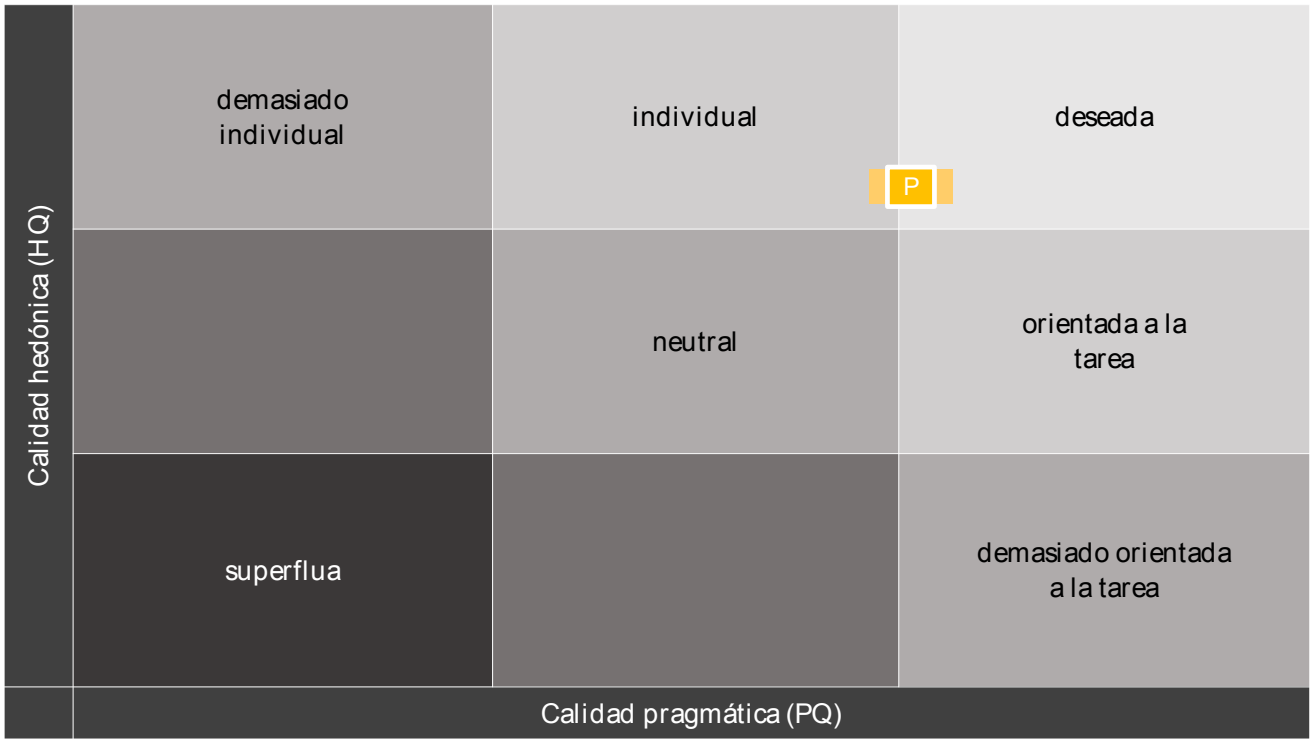

Figura 6.5: Criterios de calidad hedónicos y pragmáticos de la interfaz de usuario de la aplicación de prueba

(rectángulo de color naranja detrás de la media). Dichos elementos están ubicados en dos regiones: individual y deseada. Esto significa que la interfaz tiende a favorecer los criterios hedónicos por encima de los pragmáticos. En este sentido, los usuarios se identifican de cierta forma con la interfaz y consideran a los métodos de interación como interesantes o novedosos, pero ello no significa que no cumplan con su propósito de forma adecuada.

El rectángulo de confianza es pequeño, lo cual implica que las opiniones y evaluaciones de los usuarios fueron uniformes. Esto puede deberse a que todos tienen una experiencia semejante en términos de métodos de interacción con dispositivos móviles (gestos para interactuar con el contenido en ciertas aplicaciones, e.g., sacudir el dispositivo para descartar un cambio, inclinarlo para rotar la pantalla, desplazarlo para ver distintas partes de una imagen, etc.).

Algunas interpretaciones que proporciona AttrakDiff son las siguientes:

- La interfaz no es considerada pragmática del todo, ya que existe un traslape entre una región y otra. El usuario se siente auxiliado por la interfaz, sin embargo, existe la posibilidad de mejorar.
- En términos de la calidad hedónica, el usuario se siente estimulado por la interfaz e identificado con ella.
- El valor de la atracción que sienten los usuarios por utilizar la interfaz es alto.

Capítulo 7

Conclusiones y trabajo a futuro

Para concluir la presente propuesta, presentamos finalmente una recapitulación del problema en la sección 7.1, para posteriormente introducir las conclusiones y contribuciones como producto del desarrollo del soporte (sección 7.2). Finalmente, en la sección 7.3 enumeramos múltiples actualizaciones que consideramos viables y posibles mejoras que detectamos a lo largo de cada una de las fases de la presente tesis, todo esto con el fin de crear una herramienta que resulte aún más robusta, eficiente, usable y adaptable.

7.1. Recapitulación del problema

Como se describió en las secciones 1.2 y 1.3, las formas de interactuar entre dos o más personas han evolucionado con el paso de los años. Dicha evolución está relacionada directamente con el uso y aprovechamiento de los avances tecnológicos.

Uno de los objetivos de la presente tesis consistía precisamente, en tomar ventaja de la tecnología desde el punto de vista de los dispositivos móviles, utilizándolos como un medio a través del cual sea posible llevar a cabo interacciones más dinámicas y espontáneas. Concretamente, la forma en cómo se ha manifestado el desarrollo tecnológico en dichos dispositivos (sensores, adaptadores de red, poder de procesamiento) y su creciente disponibilidad, han provocado un interés por desarrollar aplicaciones de tipo comercial y de investigación, con formas de interacción más complejas e intuitivas.

Hoy en día, es fácil encontrar aplicaciones de carácter individual o colaborativo, sin embargo, son sumamente escasas aquellas que proveen el soporte necesario para realizar transiciones entre un tipo de trabajo y otro. Esto implica que aplicaciones diseñadas para ser utilizadas de forma individual no puedan ser utilizadas para colaborar y viceversa.

Ofrecer herramientas que permitan realizar transiciones entre un tipo de trabajo y otro, explotando las capacidades tecnológicas de los dispositivos móviles para brindar formas de interacción novedosas y soporte para la mayor parte de los dispositivos, sigue siendo un reto. Actualmente, los ecosistemas de dispositivos así como sus respectivas aplicaciones, se han vuelto más complejas y heterogéneas, abriendo una brecha de oportunidades para diversos campos de investigación, tales como: el cómputo ubicuo, el cómputo móvil, los sistemas colaborativos, HCI, entre otros más.

7.2. Conclusiones y contribuciones

Hoy en día, la tecnología con la cual cuentan los dispositivos móviles y la velocidad de comunicación de las redes inalámbricas, da paso a una nueva gama de aplicaciones más robustas y complejas, gracias a las cuales, es posible llevar a cabo interacciones mucho más dinámicas, prácticamente en cualquier momento y en cualquier lugar. Así mismo, las interfaces de usuario con las cuales es posible interactuar con dichos dispositivos han evolucionado, de tal forma, que es posible tomar ventaja de la tecnología embebida en los dispositivos móviles para crear formas de interacción más eficaces e intuitivas.

A lo largo de la presente tesis, describimos un soporte multi-usuario y multi-plataforma para dispositivos móviles (teléfonos inteligentes y tabletas, específicamente), que facilita de forma flexible transiciones del trabajo individual al trabajo colaborativo y viceversa, mediante técnicas de remodelación y redistribución plásticas. La contribución principal consiste en el diseño e implementación de un soporte (a modo de framework) que cumple con el propósito ya descrito. El diseño se realizó de tal forma, que pueda ser migrado a otras plataformas para poder crear una implementación que sea nativa en otros sistemas operativos. Adicionalmente, la implementación se entrega como un proyecto de tipo biblioteca (*project library*) para Android. Dicho proyecto cuenta con la documentación necesaria para entender el funcionamiento de cada una de las funciones y clases que lo componen.

El soporte cuenta con tres servicios principales: descubrimiento de dispositivos, control de acoplamiento y distribución de actualizaciones, los cuales pueden ser utilizados, si así se requieriera, de forma individual para aumentar la funcionalidad de cualquier tipo de aplicaciones, cuyo modelo de tareas pueda beneficiarse por el uso de alguno de los servicios.

Como se mencionó en la sección 4.5, ya existen algunas herramientas como JmDNS y NSD, los cuales fungen como bibliotecas para el descubrimiento de servicios, sin embargo, ambos tienen ciertas desventajas tales como la incompatibilidad entre diferentes versiones de Android o el hecho de que no están disponibles para dispositivos que no son tan recientes. Es por ello que se decidió implementar un servicio con el mismo propósito, pero que pudiera ser utilizado tanto por dispositivos antiguos como por aquellos que acaban de ser lanzados al mercado.

Uno de los aspectos más importantes dentro del contexto del soporte, es el reconocimiento y procesamiento de los gestos de acoplamiento. Existe un pontencial inherente en estos percutores que va más allá de los ecosistemas de dispositivos propuestos en la presente tesis: si bien los gestos son utilizados como el medio para determinar variables contextuales y así obtener la relación espacial entre un dispositivo y otro, podría ser posible, en un futuro, utilizar dispositivos especializados embebidos en el ambiente para establecer interacciones dinámicas entre un dispositivo móvil y dichos dispositivos especiales. Para ello, sería necesario que el hardware requerido sea mucho más económico, se diseñen nuevas interfaces de usuario (ubicuas) y se establezca un conjunto de protocolos de comunicación y de seguridad.

Los gestos de acoplamiento pueden ser tan complejos y robustos como lo permita el hardware y software de un dispositivo móvil. En la presente tesis, se propuso la siguiente categorización de dichos gestos: basados en el uso de interfaces y sensores conscientes de contexto, basados en el uso de gestos sobre la pantalla táctil y basados en el uso de interfaces de comunicación alter-

nativas. Cabe mencionar que los gestos que engloban estas categorías, no necesariamente deben tener como objetivo acoplar dos o más dispositivos, tal y como lo demuestran algunos prototipos presentados en el estado del arte.

7.3. Trabajo a futuro

A continuación, enumeramos algunas posibles mejores y adiciones que podrían implementarse con el objeto de hacer más robusto el soporte:

- Desde hace ya algunos años, se han desarrollado múltiples protocolos para el envío confiable de mensajes multicast, los cuales hacen uso de *sockets* de tipo UDP. Lo que se busca con estos protocolos es aprovechar la velocidad de las conexiones UDP y la seguridad que brinda TCP en cuanto a la recepción de paquetes. Existen ya algunas bibliotecas (adaptaciones de dichos protocolos) capaces de brindar estos beneficios, tales como **Scalable Reliable Multicast** y **Uniform Reliable Group Communication Protocol** [Obraczka, 1998] (cada uno carga con sus respectivas desventajas), sin embargo aún no existe un estándar como tal, ya que el envío confiable de mensajes multicast depende en gran medida de los requerimientos de la aplicación final.

- Si se implementa algún gesto de acoplamiento que dependa del acelerómetro y de dejar en reposo el dispositivo sobre una superficie, es posible agregar un proceso de calibración para evitar lecturas erróneas cuando se utilicen los dispositivos sobre superficies con inclinaciones considerables.

- Agregar un mecanismo que revise la disponibilidad de los dispositivos que pertenecen a una sesión colaborativa cuando éstos hayan abandonado la aplicación y que por alguna razón sus mensajes de abandono no hayan sido recibidos. Esta adición está relacionada con la utilización de un protocolo de comunicación seguro basado en comunicaciones de tipo UDP.

- El servidor local actual (**UpdateDistributionTCPServer**) solamente recibe peticiones vía puertos TCP. Aunque existe un parámetro que permite propagar un puerto UDP disponible, falta agregar la implementación para que también exista un servidor que reciba conexiones por puestos UDP (una clase interna **UpdateDistributionUDPServer** contenida en **UpdateDistributionService**).

- El envío de objetos se realiza de la forma un objeto por hilo. Es posible explorar otras opciones cambiando la forma en cómo se reparte el trabajo en **WorkerThread** (ver sección 4.2.1), modificando el número de operaciones en la red que se realizan por hilo, sobre todo si en algún momento el número de dispositivos acoplados pudiera crecer aún más.

- Agregar notificadores o *listeners* adicionales para cada uno de los eventos mencionados en la sección 5.3.2, lo cual implica crear interfaces adicionales y la consecuente modificación de los servicios afectados.

- Sincronización de los gestos de acoplamiento para que siempre sean recibidos ambos intentos de acoplamiento. Actualmente, es probable que solamente uno de los intentos de

acoplamiento sea recibido, razón por la cual fue necesaria implementar el protocolo para la distribución de la nueva lista de grupo.

- Implementar, precisamente, un nuevo protocolo para la distribución de la nueva lista de grupo.
- Verificar que el dispositivo cuente con una conexión a una red inalámbrica.
- Ampliar el número de variables contextuales que pueden ser evaluadas para determinar si un intento de acoplamiento es válido.

Capítulo 8

Apéndice

Como última contribución, añadimos información relevante relacionada con las bases del soporte que pueden resultar de utilidad general, así como un ejemplo de cada uno de los cuestionarios aplicados para la fase de diseño y pruebas de la presente propuesta.

8.1. Plataforma para dispositivos móviles: Android

Las aplicaciones para el sistema operativo Android [Google, 2014a] están escritas en el lenguaje de programación Java. Google proporciona las herramientas necesarias para poder construir aplicaciones para esta plataforma, junto con toda la documentación y especificaciones más recientes. Todo el código se organiza en un solo archivo con extensión .apk, el cual es considerado como el instalador de una aplicación (ya situado en un dispositivo móvil).

Al instalar una aplicación en un dispositivo, existe un entorno propio limitado de seguridad:

- El sistema operativo Android es un sistema multiusuario basado en Linux, en el que cada aplicación se considera como un usuario distinto.
- El sistema operativo asigna a cada aplicación un ID de usuario único. Así mismo, establece permisos sobre todos los archivos en una aplicación para que sólo el ID de usuario asignado a esa aplicación pueda acceder a ellos.
- Cada proceso tiene su propia máquina virtual, por lo que el código de una aplicación se ejecuta de forma aislada con respecto a otras aplicaciones.
- De forma predeterminada, cada aplicación se ejecuta en su propio proceso. Android comienza el proceso cuando alguno de los componentes de la aplicación debe ser ejecutado y luego cierra el proceso cuando ya no se necesite o cuando el sistema debe recuperar la memoria para otras aplicaciones.

De esta manera, el sistema operativo implementa el **principio de privilegios mínimos**, i.e., cada aplicación por defecto, sólo tiene acceso a los componentes que necesita para hacer su trabajo. Esto conlleva a la creación de un entorno muy seguro en el que una aplicación no puede acceder a partes del sistema sobre los cuales no tiene permisos.

Existen maneras para que una aplicación pueda compartir datos con otras aplicaciones, así como para la solicitud de acceso a los servicios del sistema:

- Es posible hacer que dos aplicaciones compartan el mismo ID de usuario, en cuyo caso son capaces de acceder a los archivos de cada uno. Para ahorrar recursos del sistema, las aplicaciones con el mismo ID de usuario también se pueden ejecutar bajo el mismo proceso y compartir la misma máquina virtual (posible únicamente si las solicitudes están firmadas con el mismo certificado).
- Una aplicación puede solicitar permiso para acceder a los datos del dispositivo, tales como los contactos del usuario, mensajes SMS, entre otros . Todos los permisos de las aplicaciones debe ser concedidos por el usuario durante sus respectivas instalaciones.

8.2. Componentes de aplicación

Los **componentes de aplicación** son esenciales para una aplicación Android. Cada componente es un punto diferente a través del cual el sistema puede entrar a una aplicación. No todos los componentes son los puntos de entrada reales para un usuario y algunos pueden depender de otros, pero cada uno existe como una entidad propia y juega un rol específico: cada uno es un elemento exclusivo que ayuda a definir el comportamiento general de la aplicación.

Existen cuatro tipos diferentes de componentes de aplicación. Cada tipo tiene un propósito distinto y un ciclo de vida diferente que define cómo se crea y se destruye el componente. A continuación se enumera y explica cada uno de ellos:

- **Actividades:** una actividad (*Activity*) representa una única pantalla con una interfaz de usuario (e.g., una aplicación de correo electrónico puede tener una actividad que muestra una lista de nuevos mensajes, otra actividad para redactar un correo electrónico, otra para la lectura de los mismos, etc.). Aunque las actividades trabajan juntas para formar una experiencia de usuario coherente en la aplicación, cada una es independiente de las demás. Como tal, una aplicación diferente puede iniciar cualquiera de estas actividades si la aplicación que contiene a la actividad referenciada lo permite (e.g., una aplicación que utilice la cámara puede iniciar la actividad en la aplicación de correo electrónico, mediante la cual se redacta un correo nuevo con el fin de que el usuario pueda compartir una fotografía).

  Una actividad se implementa como una subclase de *Activity*.

- **Servicios:** un servicio (*Service*) es un componente que se ejecuta en segundo plano para realizar operaciones de larga duración o para realizar un trabajo de procesos remotos. A diferencia de las actividades, un servicio no proporciona una interfaz de usuario, por lo tanto no se puede interactuar directamente con éste (e.g., un servicio puede reproducir música en segundo plano mientras el usuario está utilizando otra aplicación o puede obtener datos mediante la red sin bloquear el flujo de interacción con alguna actividad). Otro componente de aplicación (tal como una actividad) puede iniciar un servicio y dejar que se ejecute en segundo plano, unirse a éste con el fin de interactuar y realizar operaciones de forma conjunta o simplemente detenerlo.

Un servicio se implementa como una subclase de *Service*.

- **Proveedor de contenido:** un proveedor de contenido (*Content provider*) gestiona un conjunto de datos compartidos de la aplicación. Este componente puede almacenar los datos en el sistema de archivos, una base de datos SQLite, en la web o en cualquier otro lugar de almacenamiento permanente al cual la aplicación pueda tener acceso. A través de un proveedor de contenido, otras aplicaciones pueden consultar o incluso modificar los datos si ese proveedor de contenido en específico lo permite (e.g., Android proporciona un proveedor de contenidos que gestiona la información de contactos del usuario). Como tal, cualquier aplicación con los permisos adecuados puede consultar parte del proveedor de contenidos para leer y escribir información sobre una persona en particular.

  Los proveedores de contenido también son útiles para la lectura y escritura de datos privados que no se deben compartir (e.g., la aplicación NotePad utiliza un proveedor de contenido para guardar notas).

  Un proveedor de contenido se implementa como una subclase de *ContentProvider* y debe implementar un conjunto estándar de APIs que permiten a otras aplicaciones llevar a cabo transacciones.

- **Receptor de difusiones:** un receptor de difusiones (*Broadcast receiver*) es un componente que responde a mensajes de difusión para todo el sistema. Muchas emisiones pueden originarse en el sistema (e.g., una emisión para anunciar que la pantalla se ha apagado, la batería está baja o que una imagen fue capturada). Las aplicaciones también pueden iniciar emisiones de forma manual (e.g., para permitir que otras aplicaciones sepan que algunos datos se han descargado en el equipo y están disponibles para su utilización). Aunque los receptores de difusiones no se muestran en una interfaz de usuario, pueden crear notificaciones en la barra de estado para alertar al usuario cuando se produce un evento de difusión. Comúnmente, un receptor de difusión es sólo un disparador para arrancar otros componentes y está destinado a hacer una cantidad mínima de trabajo.

  Un receptor de difusión se implementa como una subclase de *BroadcastReceiver* y cada emisión se entrega como un objeto *Intent*.

Debido a que el sistema ejecuta cada aplicación en un proceso separado con permisos de archivos que restringen el acceso a otras aplicaciones, una aplicación no puede activar directamente un componente de otra aplicación, para ello se realiza una solicitud al sistema operativo. Por lo tanto, para activar un componente de otra aplicación, se debe entregar un mensaje al sistema que especifique la intención de iniciar un componente en particular. A estas peticiones se les conoce como *Intentos*.

8.2.1. El archivo *Manifest*

Antes de que el sistema operativo pueda iniciar un componente de aplicación, el sistema debe saber que existe ese componente mediante la lectura del archivo *AndroidManifest.xml* de la aplicación. Este archivo debe estar situado en la raíz del directorio del proyecto y contener las declaraciones de todos los componentes a utilizar .

El archivo *Manifest* permite realizar algunas declaraciones y especificaciones, tales como:

- Identificar cualquier permiso de usuario que la aplicación requiere (e.g., el acceso a Internet, lectura y acceso a la lista de contactos, etc.).
- Declarar el nivel de API mínimo requerido por la aplicación.
- Declarar las características de hardware y software requeridos por la aplicación (e.g., cámara, Bluetooth, pantalla multitouch, etc.).
- Bibliotecas externas (e.g., Google Maps).
- Entre otras.

8.2.2. Conceptos básicos para el procesamiento de gestos sobre la pantalla táctil

Android brinda el soporte necesario para poder detectar y procesar de forma adecuada cada una de los posibles alternativas al momento de que el dispositivo detecte gestos táctiles sobre la pantalla. En sí, están definidos los siguientes eventos:

- ACTION_DOWN: se dispara cuando el usuario coloca un dedo o una pluma sobre la pantalla del dispositivo.
- ACTION_UP: evento detectado cuando el usuario separa el dedo o la pluma de la pantalla.
- ACTION_MOVE: detecta el gesto de arrastre sobre la pantalla, siguiendo a ACTION_DOWN y precedido por ACTION_UP.
- ACTION_POINTER_DOWN: utilizado para detectar cuando un usuario coloca otro dedo sobre la pantalla de un dispositivo.
- ACTION_POINTER_UP: permite detectar cuando un usuario separa alguno de los dedos secundarios de la pantalla.
- ACTION_CANCEL: se dispara cuando una vista comienza a consumir un gesto iniciado con ACTION_DOWN, pero que no puede procesar las siguientes partes del gesto actual.

Todos los gestos siempre comienzan con ACTION_DOWN y terminan con ACTION_UP. Al ser detectado algún evento (disparado por un apuntador que haya tocado la pantalla del dispositivo), éste será propagado por el árbol de vistas (a través de despachadores, mediante las llamadas internas a los métodos *dispatchTouchEvent()* de cada una de las vistas en el árbol) hasta que:

- Se alcance el fondo del árbol de vistas.
- Alguna de las vistas tenga implementado un método para procesar el evento que se está propagando, en cuyo caso puede optar por cortar la propagación del evento a través del arbol devolviendo *True* (desde el método *onTouchEvent()*, lugar donde se procesará dicho evento) o permitir que continúe la propagación devolviendo *False* (básicamente, esto significa que

mientras se devuelva True, se está llevando a cabo un gesto apropiado, i.e., un gesto con un comportamiento o características esperadas). Una vez alcanzado el fondo del árbol, el evento es devuelto a la cima del mismo (la actividad en sí) en orden inverso. Es importante notar que por motivos de optimización, una vez que una vista ha cortado la propagación de un gesto perteneciente a un evento *Touch*, las vistas restantes del árbol no captarán dato alguno acerca del gesto activo, "truncándolo" temporalmente y evitando que las vistas restantes puedan responder a ese evento a lo largo del gesto. Una vez finalizado el gesto, ACTION_DOWN será el encargado de devolver las banderas correspondientes a su estado original para que el árbol de vistas completo pueda consumir el siguiente gesto de forma apropiada).

Algunos dispositivos tienen la capacidad para procesar múltiples apuntadores sobre sus respectivas pantallas. Como se mencionó anteriormente, serán ACTION_POINTER_DOWN y ACTION_POINTER_UP los posibles eventos que se pueden detectar cuando se coloca o se despega un apuntador extra de la pantalla. Para poder identificar el apuntador que generó el evento capturado en el método *onTouchEvent()* de alguna vista, es posible obtener el ID y por consiguiente el índice dentro del array de tipo *MotionEvent*, el cual se proporciona como parámetro de dicho método y del cual se pueden extraer más características del evento capturado.

Publicación del autor

Marco Castro and Sonia Mendoza, Supporting Face to Face Collaboration through Dynamic Arrays of Mobile Devices, The 2014 11th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2014). Ciudad del Carmen, Campeche, México. September 29 - October 3, 2014. IEEE Computer Society, ISBN 978-1-4799-6228-0, pp. 209 - 214.

Bibliografía

[Bevan, 2001] Bevan, N. (2001). International standards for hci and usability. *Int. J. Hum.-Comput. Stud.*, 55(4):533–552.

[BluetoothSIG, 2014] BluetoothSIG (2014). Specification adopted documents. `https://www.bluetooth.org/en-us/specification/adopted-specifications`.

[Calvary et al., 2001a] Calvary, G., Coutaz, J., and Thevenin, D. (2001a). Supporting context changes for plastic user interfaces: A process and a mechanism. In *In Proceedings of the Joint Conf. on Human-Computer Interaction IHM-HCI 2001*, pages 349–363, Lille, France. Springer-Verlag.

[Calvary et al., 2001b] Calvary, G., Coutaz, J., and Thevenin, D. (2001b). A unifying reference framework for the development of plastic user interfaces. In *Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pages 173–192, London, UK. Springer-Verlag.

[Calvary et al., 2007] Calvary, G., Coutaz, J., and Thevenin, D. (2007). Métamorphose des ihm et plasticité. pages 35–60.

[Carroll, 2003] Carroll, J. M. (2003). *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. Amsterdam, Oxford, Paris, San Francisco, USA.

[Coutaz and Calvary, 2012] Coutaz, J. and Calvary, G. (2012). Hci and software engineering for user interface plasticity. In *The Human-Computer Handbook - Fundamentals, Evolving Technologies and Emerging Applications*, pages 1195–1220. CRC Press Taylor and Francis Group.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Google, 2009] Google (2009). AsyncTask class overview. `http://developer.android.com/reference/android/os/AsyncTask.html`.

[Google, 2012] Google (2012). Network Service Discovery overview. `http://developer.android.com/training/connect-devices-wirelessly/nsd.html`.

[Google, 2014a] Google (2014a). Android Developers website. `http://developer.android.com/index.html`.

[Google, 2014b] Google (2014b). Platform versions usage. `http://developer.android.com/about/dashboards/index.html`.

[Hinckley, 2003] Hinckley, K. (2003). Bumping objects together as a semantically rich way of forming connections between ubiquitous devices.

[Hinckley et al., 2004] Hinckley, K., Ramos, G., Guimbretière, F., Baudisch, P., and Smith, M. (2004). Stitching: pen gestures that span multiple displays. In Costabile, M. F., editor, *Proceedings of the working conference on Advanced visual interfaces, AVI 2004*, pages 23–31, Gallipoli, Italy. ACM Press.

[IEEE, 2014] IEEE (2014). IEEE Standards Association. `http://standards.ieee.org`.

[IETF, 1981] IETF (1981). RFC Repository. `http://www.ietf.org/rfc.html`.

[Krumm, 2009] Krumm, J. (2009). *Ubiquitous Computing Fundamentals*. Chapman & HALL/CRC, 1st edition.

[Lucero et al., 2010] Lucero, A., Holopainen, J., and Jokela, T. (2010). Collaborative use of mobile phones for brainstorming. In de Sá, M., Carriço, L., and Correia, N., editors, *In Proceedings of the 12th Conference on Human Computer Interaction with Mobile Devices and Services*, pages 337–340, Lisbon, Portugal. ACM.

[Lucero et al., 2011] Lucero, A., Holopainen, J., and Jokela, T. (2011). Pass-them-around: Collaborative use of mobile phones for photo sharing. In Tan, D. S., Amershi, S., Begole, B., Kellogg, W. A., and Tungare, M., editors, *In Proceedings of the International Conference on Human Factors in Computing Systems*, pages 1787–1796, Vancouver, BC, Canada. ACM.

[Lucero et al., 2012a] Lucero, A., Holopainen, J., and Jokela, T. (2012a). Mobicomics: Collaborative use of mobile phones and large displays for public expression. In Churchill, E. F., Subramanian, S., Baudisch, P., and OHara, K., editors, *In Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*, pages 21–24, San Francisco, CA, USA. ACM.

[Lucero et al., 2012b] Lucero, A., Jokela, T., Palin, A., Aaltonen, V., and Nikara, J. (2012b). Easygroups: binding mobile devices for collaborative interactions. In Konstan, J. A., Chi, E. H., and Höök, K., editors, *In Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 2189–2194, Austin, TX, USA. ACM.

[Mandal et al., 2005] Mandal, A., Lopes, C. V., Givargis, T., Haghighat, A., Jurdak, R., and Baldi, P. (2005). Beep: 3D indoor positioning using audible sound. In *IEEE Consumer Communications and Networking Conference*.

[Mayrhofer and Gellersen, 2009] Mayrhofer, R. and Gellersen, H. (2009). Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Trans. Mob. Comput.*, 8(6):792–806.

[NVIDIA, 2010] NVIDIA (2010). Tegra Android Accelerometer. Whitepaper.

[Obraczka, 1998] Obraczka, K. (1998). Multicast Transport Protocols: A Survey and Taxonomy. *Communications Magazine, IEEE*, 36(1):94–102.

[Peng et al., 2009] Peng, C., Shen, G., Zhang, Y., and Lu, S. (2009). Point&connect: intention-based device pairing for mobile phone users. In Zielinski, K., Wolisz, A., Flinn, J., and LaMarca, A., editors, *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys 2009), Kraków*, pages 137–150, Poland. ACM.

[Pressman, 2002] Pressman, R. S. (2002). *Ingeniería del Software - Un enfoque práctico*. Mc Graw Hill, Madrid, España, 5th edition.

[Schneider et al., 2012] Schneider, D., Seifert, J., and Rukzio, E. (2012). Mobies: Extending mobile interfaces using external screens. In Rukzio, E., editor, *In Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, pages 4–6, Ulm, Germany. ACM.

[Talukder et al., 2010] Talukder, A. K., Ahmed, H., and Yavagal, R. R. (2010). *Mobile Computing - Technology, Applications and Service Creation*. Tata McGraw Hill, 2nd edition.

[Tandler et al., 2001] Tandler, P., Prante, T., Müller-Tomfelde, C., Streitz, N. A., and Steinmetz, R. (2001). Connectables: dynamic coupling of displays for the flexible creation of shared workspaces. In *Proceedings of the The 14th Annual ACM Symposium on User Interface Software and Technology*, pages 11–20, Orlando, Florida, USA. ACM.

[Thevenin and Coutaz, 1999] Thevenin, D. and Coutaz, J. (1999). Plasticity of user interfaces: Framework and research agenda. In Sasse, M. A. and Johnson, C., editors, *In Proceedings of INTERACT 99 - IFIP TC13 Seventh International Conference on Human-Computer Interaction*, Edinburgh, Scotland. IOS Press.

[Varshavsky et al., 2007] Varshavsky, A., Scannell, A., LaMarca, A., and de Lara, E. (2007). Amigo: Proximity-based authentication of mobile devices. In Krumm, J., Abowd, G. D., Seneviratne, A., and Strang, T., editors, *UbiComp 2007: Ubiquitous Computing, 9th International Conference, UbiComp 2007*, pages 253–270, Innsbruck, Austria. Springer.