CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Departamento de Computación

Mecanismo de remodelación semi-plástica adaptativa para interfaces colaborativas

Tesis que presenta

Roberto Enrique Alberto Lira

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dra. Sonia Guadalupe Mendoza Chapa

México, D.F. Enero 2010

Resumen

La heterogeneidad de dispositivos de cómputo, así como el progreso de las redes de comunicación y la miniaturización de dichos dispositivos, han permitido imaginar al usuario como: "un ente que evoluciona en un entorno variable y que utiliza de manera oportunista diversos dispositivos de cómputo para satisfacer sus múltiples necesidades en todo momento y en cualquier lugar". Esta heterogeneidad de dispositivos requiere que la interfaz de usuario de sus aplicaciones se adapten dinámicamente a cambios en el contexto de uso, el cual se define en términos del usuario, del entorno y de la plataforma. La propiedad de plasticidad es la capacidad de una interfaz de usuario de soportar variaciones en las características físicas de la plataforma (e.g., tamaño de la pantalla) y del entorno (e.g., interacción cara a cara vs interacción distribuida), preservando un conjunto de criterios de calidad (e.g., usabilidad y continuidad).

En el dominio de las aplicaciones mono-usuario, la propiedad de plasticidad ha sido estudiada principalmente mediante la definición de algunos conceptos y el desarrollo de algunos prototipos de laboratorio. A excepción de Flexclock, estos prototipos ofrecen pocos detalles de los mecanismos desarrollados para llevar acabo el proceso de adaptación plástica. En consecuencia, estas soluciones difícilmente pueden ser reutilizables en otros ámbitos porque distan mucho de ser propuestas genéricas. Por otra parte, la propiedad de plasticidad no ha sido plenamente introducida en aplicaciones colaborativas, a pesar de la necesidad inminente de dotar a estas aplicaciones de la capacidad de adaptabilidad.

Esta problemática es abordada mediante el desarrollo de un mecanismo semi-plástico adaptativo, el cual remodela la interfaz de usuario de una aplicación colaborativa en respuesta a cambios en la plataforma. Mediante este mecanismo, el proceso de adaptabilidad plástica es realizado por la aplicación misma sin intervención del usuario. El mecanismo propuesto ha sido validado mediante el desarrollo del juego didáctico "serpientes y escaleras", cuyas características permiten que múltiples usuarios (e.g., niños de educación básica) interactúen desde dispositivos de cómputo heterogéneos (e.g., PC, PDA, laptop y pizarrón interactivo).

Este trabajo pretende aportar nuevos avances en el tópico de investigación de la adaptabilidad de aplicaciones (particularmente, en la plasticidad de interfaces de usuario) para el soporte del trabajo colaborativo. A partir de los resultados obtenidos, se puede imaginar lógicamente la definición de conceptos y mecanismos de plasticidad genéricos que podrían adaptarse a cualquier aplicación colaborativa.

Palabras claves: adaptación de interfaces de usuario, plasticidad por remodelación, mecanismo semi-plástico adaptativo, aplicaciones colaborativas, juego didáctico "serpientes y escaleras".

Abstract

The heterogeneity of computing devices as well as the progress of communication networks and the miniaturization of such devices, have allowed to imagine the user as "an entity that evolves within a variable environment and uses in an opportunist way several computing devices in order to satisfy his several needs anytime anywhere". This heterogeneity of devices requires the user interface of their applications to dynamically adapt to changes in the use context, which is defined in terms of the user, the environment and the platform. The plasticity property is the ability of a user interface to support variations in the physical characteristics of the platform (e.g., display size) and the environment (e.g., face to face interaction vs distributed interaction) while preserving a set of quality criteria (e.g., usability and continuity).

In the mono-user application domain, the plasticity property has been mainly studied by means of the definition of some concepts and the development of some laboratory prototypes. Excepting from FlexClock, these prototypes provide few details about the developed mechanisms to carry out the process of plastic adaptation. Consequently, these solutions are unlikely reusable in other contexts as they are far from being generic proposals. On the other hand, the plasticity property has not been fully put into collaborative applications, in spite of the imminent need to provide these applications with the adaptability capacity.

To cope with this problem, we have developed a semi-plastic adaptive mechanism that remodels the user interface of a collaborative application in response to changes in the platform. By means of this mechanism, the process of plastic adaptability is performed by the application itself without user intervention. The proposed mechanism has been validated by means of the development of the didactic game "snakes and ladders", whose main characteristics allow multiple users (e.g., elementary school students) to interact from heterogeneous computing devices (e.g., PC, PDA, laptop and interactive whiteboard).

This work aims to provide new advances in the research topic of application adaptability (particularly, in the user interface plasticity) for collaborative working support. From the obtained results, we can logically imagine the definition of generic concepts and mechanisms for plasticity, which could be adapted to any collaborative application.

Palabras claves: user interface adaptation, plastic remodeling, semi-plastic adaptive mechanism, collaborative applications, "snakes and ladders" didactic game.

Agradecimientos

A Dios, por darme la vida y la fuerza para concluir el presente trabajo.

A mi padre el Profr. Roberto Alberto Sánchez, porque en mis caídas siempre me ayudaste a levantarme. Tus sabios consejos me han permitido seguir un camino recto. Eres el mejor padre del mundo y un ser humano excelente. Gracias por darme tu apoyo siempre.

A mi madre la Profra. María Estela Lira de Alberto, por ser la mejor madre del mundo y una persona ejemplar. En todo momento te has encontrado a mi lado, dándome tu apoyo, tu amor, tu comprensión y tus cuidados. Gracias por ayudarme a mantener la esperanza en los momentos más difíciles.

A mi hermana la Lic. Nubia Marlene Alberto Lira, por el gran esfuerzo en ayudarme a vivir en esta inmensa ciudad. Eres una gran persona, la mejor hermana y una mujer admirable.

A mi hermano Carlos de Jesús Alberto Lira, por darle más alegría a mi vida. Tu presencia en todo este tiempo me ha hecho saber lo bien acompañado que estoy. Sabes que te admiro mucho.

A mi asesora de tesis la Dra. Sonia Guadalupe Mendoza Chapa, por el esmero, tiempo y paciencia que dedicó en la realización del presente trabajo.

Al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional y al Consejo Nacional de Ciencia y Tecnología, por su valioso apoyo a lo largo de mis estudios de posgrado.

A la Dra. María Lizbeth Gallardo López, por su valioso apoyo como persona y sus importantes aportaciones a esta tesis de maestría, sin las cuales hubiera sido imposible alcanzar los resultados obtenidos.

Al Dr. José Guadalupe Rodríguez García por dedicar parte de su valioso tiempo a la revisión y retroalimentación de esta tesis de maestría.

A mi alma mater la Universidad Juárez Autónoma de Tabasco a quien debo mi formación académica.

A mi gran amigo Víctor Alberto Gómez Pérez, por compartir conmigo los momentos que nos han permitido crecer durante siete años.

A mis amigos Abel Cortázar, Sra. Sofia Reza, Mtra. Hortensia Almaguer, Mtro. Rubén González, Sixto Rodríguez y al Ing. Enrique Pecero, por su valioso apoyo en todo momento.

A todos mis amigos, profesores y compañeros que a lo largo de mi vida han aportado cosas positivas.

Finalmente quiero decir que, durante mi formación pude apreciar lo fácil que es destruir, pero me doy cuenta que lo verdaderamente valioso y difícil, es construir. Como prueba tengo esta tesis que con tanto trabajo ha sido concluida.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1 Contexto de investigación

La gran variedad de dispositivos de cómputo fijos y móviles (que se presentan en diversos tamaños y con diferentes capacidades de procesamiento, memoria, almacenamiento y comunicación), así como el progreso de las redes de comunicación, imponen nuevos requerimientos a las aplicaciones, tales como la capacidad de adaptar su interfaz de usuario a diferentes contextos de uso. El concepto de **contexto de uso** [Calvary et al., 2004] se define como la terna $< usuario, entorno, plataforma >$ donde:

1. el **usuario** denota el arquetipo humano que tiene por objetivo utilizar la aplicación;
2. la **plataforma** se refiere al *hardware* y *software* disponible en un dispositivo de cómputo para sostener la interacción del usuario con la aplicación; la plataforma puede ser modelada en términos de los recursos computacionales que determinan la forma en que se procesa, transmite y presenta la información, así como la manera en que el usuario manipula dicha información;
3. el **entorno** describe las condiciones físicas y sociales donde tiene lugar la interacción, e.g., los objetos, las personas y los eventos que son periféricos a la tarea que se está desarrollando, pero que pueden tener un impacto en el comportamiento de la aplicación.

Como se muestra en la Figura 1.1, la presente tesis de maestría se inscribe en el dominio de investigación de dos importantes áreas de las Ciencias de la Computación: 1) el Trabajo Cooperativo Asistido por Computadora (TCAC) y 2) la Interacción Hombre-Máquina (IHM).

1.1.1 Trabajo Cooperativo Asistido por Computadora

El TCAC es un dominio multidisciplinario que estudia tanto los aspectos sociales de las actividades individuales y colectivas, como los aspectos tecnológicos de la información y de las comunicaciones, con el fin de asistir la colaboración entre personas físicamente distribuidas. En relación con el carácter multidisciplinario del TCAC, este trabajo de investigación toma principalmente la perspectiva tecnológica para analizar, diseñar e implementar aplicaciones colaborativas. Esta clase de aplicaciones provee entornos de trabajo computarizados que permiten

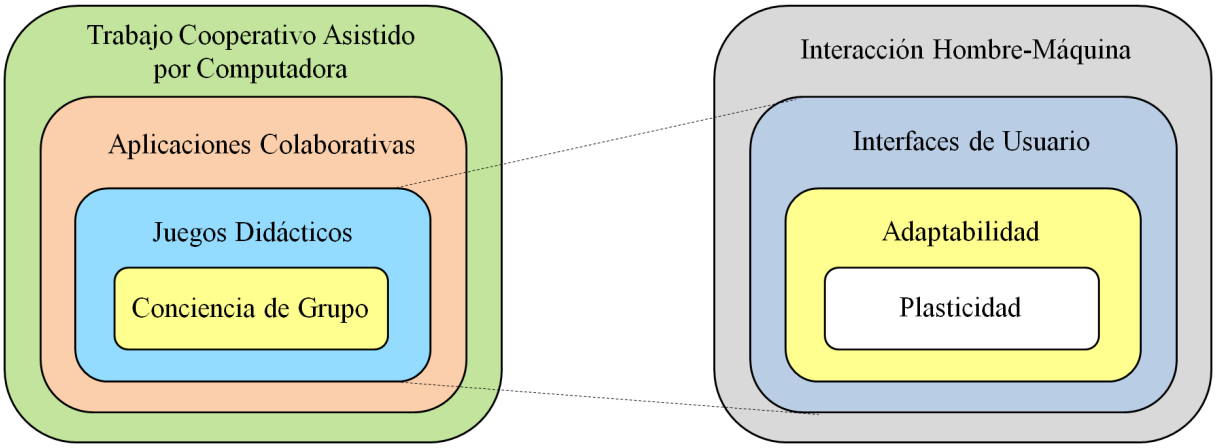


Figura 1.1: Contexto de investigación de la presente tesis de maestría

a los colaboradores intercambiar mensajes, compartir información y colaborar, aún cuando ellos no puedan reunirse físicamente en el mismo lugar y/o al mismo tiempo.

Cuando los colaboradores trabajan a distancia, requieren información sobre las actividades de sus colegas con el fin de crear un contexto para efectuar sus propias actividades. A este respecto, se han realizado importantes esfuerzos en el diseño de componentes gráficos (*widgets*) que proveen información de conciencia de grupo en entornos de interacción informal. Algunos ejemplos representativos de estos componentes gráficos fueron propuestos en aplicaciones de **juegos sociales**, tales como "Enigma Compartido" y "*Counter-Strike*".

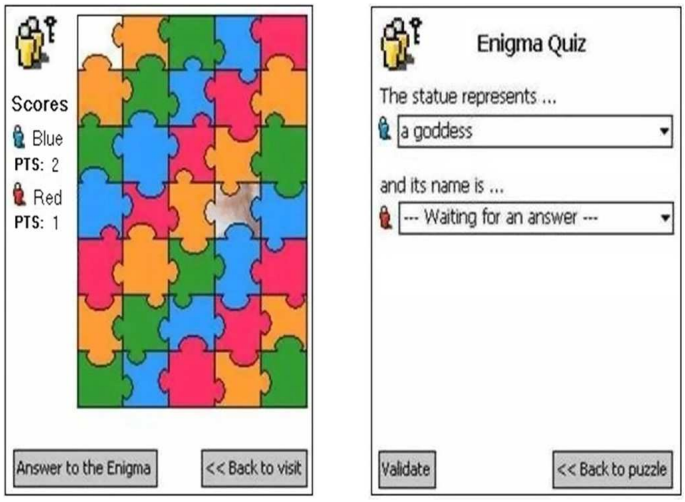


Figura 1.2: Interfaz de usuario del juego "Enigma Compartido" en una PDA

En particular, el juego "Enigma Compartido" [Dini et al., 2007] ofrece una serie de preguntas sobre un tópico que está asociado a una imagen de una obra de arte escondida en un rompecabezas. Este juego forma parte de la aplicación Cicero [Ciavarella and Paternò, 2004],

un sistema que sirve como guía de turistas en el museo de Marmol de Carrara[1]. La interfaz de usuario del juego "Enigma Compartido" en una PDA está compuesta de dos partes, accesibles una a la vez, debido al reducido tamaño de pantalla de la PDA (cf. Figura 1.2): a) la primera parte muestra la puntuación de los jugadores del grupo al que pertenece el usuario local, así como el rompecabezas que oculta la imagen de la obra de arte y b) la segunda parte de la interfaz de usuario muestra una serie de preguntas con sus posibles respuestas. En cambio, la interfaz de usuario del juego "Enigma Compartido" que se presenta en despliegues públicos de mayor dimensión y resolución (localizados en puntos estratégicos del museo), tiene como única función desplegar información de conciencia de grupo. Particularmente, dicha interfaz de usuario muestra tanto el estado de cada uno de los jugadores de los diferentes grupos como el rompecabezas (cf. Figura 1.3).

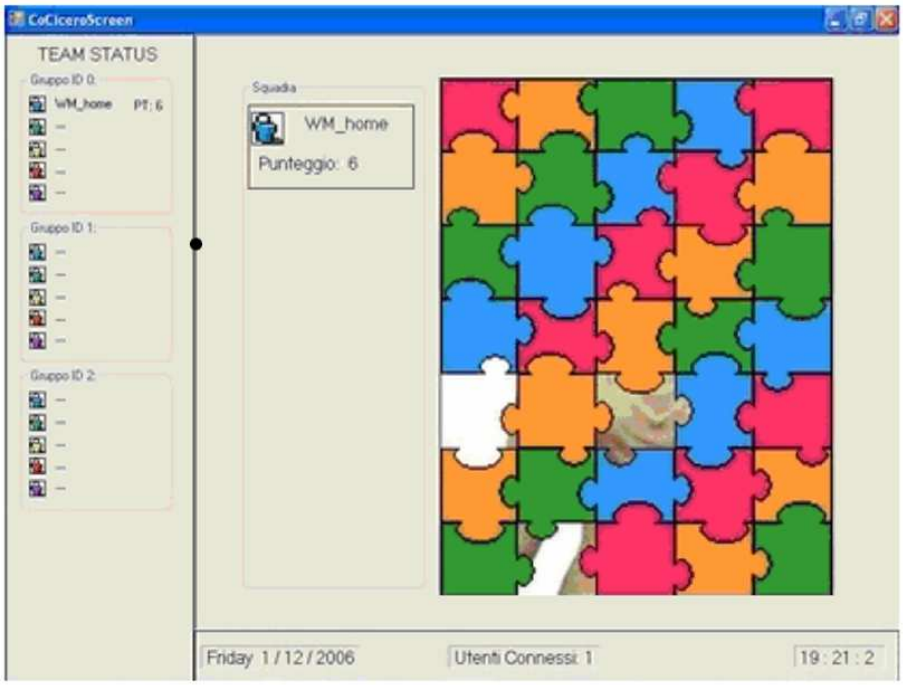


Figura 1.3: Interfaz de usuario del juego "Enigma Compartido" en un despliegue público

Una de las propuestas más populares en Internet es el juego colectivo *"Counter-Strike"*[2], el cual es un simulador de combate entre equipos de terroristas y antiterroristas. Su interfaz de usuario integra un conjunto de componentes gráficos que permiten tanto la interacción/colaboración entre los jugadores, como la provisión de información de conciencia de grupo. Estos componentes gráficos se distribuyen entre las dos vistas que forman la interfaz de usuario: a) la vista principal y b) la vista radar. La vista principal (cf. Figura 1.4) muestra, a cada jugador, los siguientes componentes gráficos:

- en la esquina superior izquierda se localiza un radar que muestra la ubicación y el campo de visión del jugador local,

[1]http://urano.isti.cnr.it:8880/museo/home.en.php
[2]http://es.wikipedia.org/wiki/Counter-Strike

Figura 1.4: Vista principal del juego *"Counter-Strike"* en una PC

- en la esquina superior derecha se encuentra una linterna que, cuando está activada, ilumina el camino del jugador local; y
- en la parte inferior de la pantalla, el jugador local puede ver la vida que le queda, su armadura, sus municiones restantes, su dinero y un reloj que muestra el tiempo restante antes del final de la ronda en curso.

Por su parte, la vista radar (cf. Figura 1.5) ofrece soporte de conciencia de grupo por medio de los siguientes componentes gráficos:

- el mapa del lugar donde se está llevando a cabo el combate,
- la ubicación de los jugadores de ambos equipos (cada equipo se diferencia por los colores azul y rojo); y
- la posición de las bombas.

Por su carácter violento, el juego colectivo *"Counter-Strike"* no es recomendable para niños, pero eventualmente podría ser utilizado para realizar prácticas de entrenamiento policiaco y militar. Cabe mencionar que este juego es accesible únicamente en PC.

1.1.2 Interacción Hombre - Máquina

Como ya se mencionó al inicio del presente capítulo, esta tesis de maestría también se inscribe en el dominio de investigación de la Interacción Hombre-Máquina (IHM), el cual se relaciona con: 1) el diseño, la implementación y la evaluación de aplicaciones de cómputo interactivas

Figura 1.5: Vista radar del juego *"Counter-Strike"* en una PC

que están enfocadas al uso humano y 2) el estudio de los principales fenómenos que rodean a dichas aplicaciones[3].

En el ámbito de la IHM, la interfaz de usuario constituye el artefacto tecnológico de una aplicación interactiva que posibilita una interacción amigable entre el usuario y la aplicación. La interfaz de usuario de una aplicación utiliza un conjunto de componentes (generalmente gráficos) para representar la información y las acciones disponibles al usuario. Comúnmente estas acciones se realizan mediante técnicas de manipulación directa [Darrell et al., 2002] e.g., *drag-and-drop* o *pick-and-drop*, que facilitan la interacción entre el usuario y la aplicación, utilizando los dispositivos de entrada/salida de la computadora[4], e.g., ratón o lápiz.

La representación gráfica se ha convertido en el estándar *"de-facto"* de las interfaces de usuario de aplicaciones computacionales. En consecuencia, el despliegue visual de aplicaciones en dispositivos que cuentan con grandes monitores (e.g., PC, laptop y pizarrón interactivo) ya es una tarea habitual de los desarrolladores de aplicaciones. Sin embargo, desde el punto de visto práctico, resulta no factible la transposición automática de una aplicación de PC hacia una PDA, debido a sus diferentes tamaños de pantalla.

El diseño inadecuado de aplicaciones accesibles desde dispositivos que ofrecen pequeñas pantallas (e.g., PDA y teléfono inteligente) puede generar serios problemas de usabilidad, e.g., la dificultad de uso o la pérdida de funcionalidad. En consecuencia, se requiere diseñar aplicaciones que posean intrínsecamente la propiedad de adaptabilidad, con el fin de poder ser accedidas transparentemente desde diversos dispositivos heterogéneos. Tres enfoques han sido propuestos para dotar a las aplicaciones de esta propiedad [Crease, 2001]:

1. **Reducción del tamaño de los componentes gráficos:** si se requiere que todos los componentes gráficos de una interfaz de usuario se encuentren en una sola vista, pero se

[3]http://en.wikipedia.org/wiki/Human-computer_interaction
[4]http://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario

dispone de un espacio limitado, entonces difícilmente podrán ser desplegados. La solución más obvia a esta problemática de espacio limitado consiste en reducir el tamaño de los componentes gráficos para que puedan tener cabida en la pantalla. Sin embargo, esta solución podría comprometer la usabilidad de la aplicación ya que, desde el punto de vista del usuario, la manipulación de los componentes gráficos podría resultar complicada.

Por otra parte, algunos componentes gráficos son más útiles al usuario que otros. En consecuencia, los componentes más importantes deberían abarcar un mayor espacio de la pantalla que los menos importantes. La reducción de tamaño de los componentes gráficos puede lograrse remodelando su presentación, e.g., substitución de un grupo de botones de radio por una lista desplegable o eliminación de un componente gráfico que esté duplicado en un menú.

2. **Multimodalidad:** consiste en aprovechar las ventajas de otras modalidades, además de la visual[5]. En consecuencia, la multimodalidad conlleva al uso de diferentes objetos de presentación, de acuerdo a los recursos que estén disponibles o que sean más convenientes para la interacción hombre-máquina. El empleo de modalidades alternativas ha demostrado ser eficaz, e.g., la incorporación de audio en los componentes gráficos de la interfaz de usuario [Crease et al., 2000] [Brewster, 1998] [Brewster and Cryer, 1999] para indicar algún error o atraer la atención del usuario.

3. **Plasticidad:** si la modalidad visual es la única opción realista y es imposible reducir el tamaño de los componentes gráficos para que puedan tener cabida en una pantalla, entonces la interfaz de usuario podría ser dividida en varias sub-secciones estructuradas a través de las cuales el usuario pueda navegar [Thevenin and Coutaz, 1999]. De esta manera, diferentes interfaces pueden ser construidas para diferentes plataformas utilizando modelos de la interfaz y de las tareas del usuario.

Cabe mencionar que a diferencia de la plasticidad, la multimodalidad no toma en cuenta el modelo de la interfaz de usuario ni las tareas en las que dicha interfaz pueda ser utilizada. En la multimodalidad, la adaptación se realiza en cada uno de los componentes gráficos, mas no en la interfaz de usuario completa como en el caso de la plasticidad. La presente tesis de maestría se enfoca en la solución por división/estructuración de la interfaz de usuario.

1.2 Planteamiento del problema

Al interactuar con aplicaciones que soportan tareas colaborativas, los miembros del grupo pueden estar reunidos en el mismo lugar, pero alternando entre trabajo colectivo e individual, o pueden encontrarse físicamente separados. En dichas condiciones de trabajo, la falta de un soporte adecuado de colaboración impide informar a cada usuario sobre las acciones que realizan los demás miembros del grupo en el espacio de trabajo compartido. Esta limitación sugiere desarrollar interfaces de usuario que informen oportunamente sobre la actividad de cada colaborador, con el fin de crear una conciencia grupal.

[5]En el mundo real se tienen cinco modalidades sensoriales y su combinación evita la saturación de información en un solo sentido del cuerpo humano.

Una interfaz mono-usuario provee componentes gráficos para permitir al usuario hacer uso de las funciones de una aplicación. Por el contrario, una interfaz de usuario colaborativa proporciona, además de los componentes ya mencionados, componentes gráficos adicionales para remplazar la información que se pierde cuando los miembros del grupo no interactúan cara a cara. En consecuencia, esta adición de componentes gráficos a las interfaces de usuario colaborativas implica una mayor dificultad en la administración del espacio de despliegue visual.

Los dispositivos de cómputo móviles que actualmente han aparecido en el mercado, cada vez cuentan con mayores capacidades de comunicación, procesamiento, memoria y almacenamiento, las cuales son suficientes para soportar algunas aplicaciones colaborativas. Estos avances tecnológicos permiten imaginar que los usuarios de una aplicación colaborativa pueden interactuar desde cualquier dispositivo de cómputo (fijo o móvil). Sin embargo, es evidente que la interfaz de usuario de una aplicación colaborativa no puede ser la misma sobre pantallas grandes y pequeñas.

Dadas las diferentes capacidades de los dispositivos de cómputo, se requiere que las interfaces de usuario de las aplicaciones se adapten a las características (e.g., tamaño de pantalla) de cada dispositivo. Con el fin de satisfacer este requerimiento, la propiedad de plasticidad permite que la interfaz de usuario de una aplicación se adapte a cambios en el contexto de uso, preservando la usabilidad y continuidad de interacción de la aplicación.

Sin embargo, la propiedad de plasticidad ha sido introducida a través de prototipos de laboratorio que revelan pocos detalles de los mecanismos que permiten llevar acabo el proceso de adaptación plástica. Esta limitante impide tener soluciones que puedan reutilizarse en otros ámbitos. Además, cabe señalar que en el dominio de los sistemas colaborativos la propiedad de plasticidad ha sido pobremente abordada, a pesar de la necesidad inminente de dotar a dichos sistemas de la capacidad de adaptación a cambios contextuales.

1.3 Objetivos del proyecto

Objetivo General

El objetivo general de la presente tesis de maestría es diseñar e implementar un mecanismo de remodelación semi-plástica (i.e., sensible a cambios en la plataforma) y adaptativa (i.e., el proceso de adaptación es realizado por el sistema sin intervención del usuario) para interfaces de usuario de aplicaciones colaborativas. La aplicación de validación de este mecanismo de remodelación es el juego didáctico “Serpientes y Escaleras”, cuyos usuarios podrían interactuar concurrentemente desde diversos dispositivos heterogéneos (e.g., PC, PDA y pizarrón interactivo).

Adicionalmente, esta aplicación de validación podría contribuir a complementar las actividades escolares, principalmente en educación básica, ya que desde el punto de vista pedagógico el juego “Serpientes y Escaleras” permite: a) fomentar la participación de los niños, b) enseñar a respetar el turno de otros, c) reforzar materias escolares (e.g., matemáticas), d) desarrollar en los niños las capacidades de expresión, análisis, reflexión y juicio y e) ofrecer un acercamiento a la computación.

Objetivos Particulares

1. Diseñar e implementar un mecanismo que:

   (a) detecte las variaciones en el contexto de uso (principalmente en la plataforma),

   (b) identifique las soluciones candidatas para adaptar la interfaz de usuario,

   (c) seleccione una solución particular y

   (d) ejecute la nueva solución.

2. Comprender la lógica funcional del juego didáctico "Serpientes y Escaleras" para implementar la aplicación colaborativa correspondiente.

3. Diseñar e implementar diferentes "versiones" de los componentes gráficos, tanto del dominio de aplicación como de conciencia de grupo, que conformarán la interfaz de usuario de la aplicación colaborativa.

4. Diseñar e implementar un soporte de comunicación que permitirá a los jugadores (quienes pueden estar físicamente distribuidos) interactuar simultáneamente con la aplicación colaborativa desde dispositivos heterogéneos.

1.4 Organización de la tesis

La presente tesis de maestría está estructurada en cinco capítulos (cf. Figura 1.6). Después de haber presentado el contexto de investigación donde se ubica la presente tesis de maestría y de haber planteado el problema que se pretende resolver, así como los objetivos que se persiguen, se expone el estado del arte en el capítulo 2. Particularmente, se presenta una síntesis de la política de manejo de sesión denominada "protocolos sociales", la cual se utiliza para definir la coordinación entre los participantes de una sesión colaborativa. Enseguida, se describen las principales características plásticas de las aplicaciones más relevantes que están relacionadas con este trabajo de investigación. A partir de estas características se destacan, mediante un estudio comparativo, las principales ventajas y desventajas de las aplicaciones analizadas.

En el capítulo 3, se presenta un diseño orientado a objetos del mecanismo ReSAIC (Remodelación Semi-plástica y Adaptativa para Interfaces Colaborativas). En primer lugar, se describen los casos de uso de dicho mecanismo. Posteriormente, se explica su funcionamiento general, así como sus principales componentes. Enseguida, se detallan las arquitectura de comunicación y de distribución del mecanismos ReSAIC, las cuales establecen respectivamente la forma de realizar el intercambio de mensajes entre sus componentes y la ubicación física de estos últimos. Finalmente, se presenta el análisis y el diseño de la aplicación de validación "Serpientes y Escaleras".

En el capítulo 4, se describen los detalles técnicos de la implementación tanto del mecanismo ReSAIC como de la aplicación de validación "Serpientes y Escaleras". En primer lugar, se justifica la selección del lenguaje de programación. A continuación, se presentan los algoritmos que describen el funcionamiento del mecanismo ReSAIC. Posteriormente, se explican tanto las principales tareas que realiza el núcleo funcional de la aplicación de validación como los pasos

del proceso de implementación de sus componentes gráficos. Finalmente, con base en la aplicación de validación se presentan las pruebas que demuestran la funcionalidad del mecanismo ReSAIC.

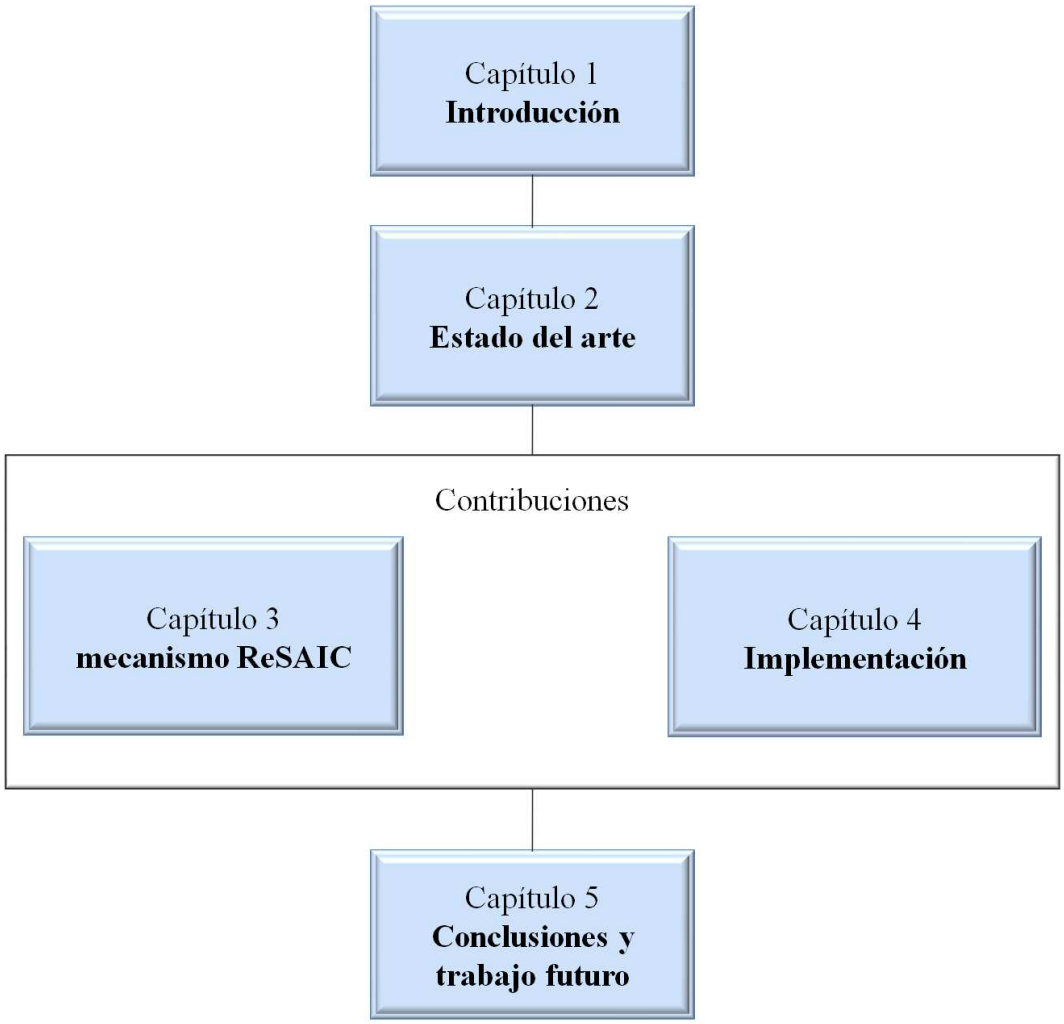


Figura 1.6: Organización del documento

Finalmente, en el capítulo 5 se presentan las conclusiones de este trabajo de investigación, así como algunas ideas de trabajo futuro que podrían mejorar el mecanismo de remodelación propuesto en cuanto a su capacidad plástica, su funcionalidad básica y su soporte de colaboración.

Capítulo 2

Estado del Arte

En el presente capítulo se describen y analizan los trabajos más representativos de la plasticidad de interfaces de usuario. Este capítulo está estructurado en cuatro partes. En primer lugar, se ofrece una breve descripción de la política de manejo de sesión denominada "protocolos sociales", la cual se utiliza en el desarrollo del presente trabajo para definir la coordinación entre los participantes de una sesión colaborativa (cf. sección 2.1). Seguidamente, se proporcionan los conceptos básicos de la plasticidad de interfaces de usuario en el dominio de los sistemas mono-usuario; en particular, se introducen los pasos del proceso de adaptación plástica, así como los tipos de plasticidad que han sido identificados en la literatura científica (cf. sección 2.2). Posteriormente, se describen los trabajos relacionados con el mecanismo de adaptación propuesto en esta tesis (cf. sección 2.3). Finalmente, se realiza un análisis comparativo de dichos trabajos relacionados con base en los conceptos introducidos en la sección 2.1 (cf. sección 2.4).

2.1 Políticas de manejo de sesión

Las políticas de manejo de sesión son reglas que gobiernan el comportamiento de un sistema, en términos de las condiciones en las que pueden ser invocadas las acciones definidas sobre los recursos compartidos. Particularmente, estas políticas definen [García et al., 2007]: a) acciones repetidas, b) acciones futuras o c) acciones que se relacionan con el cumplimiento de una condición.

Un sistema colaborativo requiere soluciones dinámicas y adaptables a su comportamiento. Las políticas de manejo de sesión ofrecen soluciones factibles para la gestión de acciones concurrentes en un sistema colaborativo, ya que permiten especificar dinámicamente estrategias adaptables que pueden ser modificadas fácilmente, sin necesidad de cambiar el código fuente o detener el sistema.

El dominio de investigación del Trabajo Cooperativo Asistido por Computadora se ha centrado principalmente en el estudio de políticas de coordinación, las cuales facilitan la interacción entre los colaboradores por medio de recursos compartidos, mientras evitan conflictos que conduzcan a la inconsistencia de dichos recursos. Ejemplos de este tipo de políticas son: el control de concurrencia [Ellis and Gibbs, 1989], *floor control* [Dommel and Garcia-Luna-Aceves, 1997] y el control de acceso [Shen and Dewan, 1992].

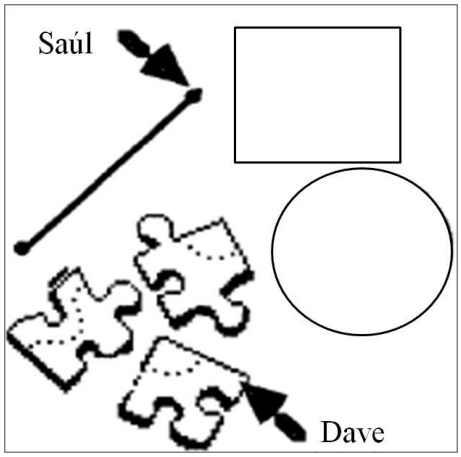


Figura 2.1: Herramienta de dibujo

Particularmente, en ausencia de una política de coordinación, los sitios son libres de intercambiar eventos, lo que ocasiona: a) posibles inconsistencias debido a que las acciones ocurren en desorden o b) competencias por recursos que solo una persona a la vez debe acceder [Greenberg and Marwood, 1994]. Aunque estos casos parecen inaceptables, podrían volverse estrategias razonables en algunas situaciones. En particular, las inconsistencias podrían no ser importantes o las personas podrían ser capaces de conciliar y reparar respectivamente sus propias acciones y conflictos.

2.1.1 Algunas inconsistencias no son importantes

Algunos tipos de inconsistencias en el espacio compartido pueden ser completamente aceptables para las personas. Por ejemplo, suponga que los usuarios Saúl y Dave están creando un objeto *bitmap* (e.g., bosquejo de las piezas de un rompecabezas) mediante la herramienta mostrada en la Figura 2.1, donde cada uno puede pintar o borrar pixeles con una brocha fina de dibujo. Sin embargo, puede ocurrir un conflicto de orden cuando uno de los usuarios dibuja algunos puntos al mismo tiempo que otro los borra (cf. Figura 2.2). Las imágenes resultantes diferirán en pocos pixeles en ambos sitios.

La Figura 2.3 muestra un acercamiento sobre 9 pixeles de un *bitmap*. Saul y Dave comienzan con la misma imagen (cf. recuadro superior de la Figura 2.3). Después, Saul dibuja un trazo diagonal, mientras que Dave borra siguiendo un trazo vertical. Las diferentes vistas locales del *bitmap* se muestran en los recuadros centrales. A continuación, las operaciones son transmitidas y ejecutadas en los otros sitios, generando imágenes finales inconsistentes (cf. recuadro inferior de la Figura 2.3).

Una diferencia de algunos cuantos pixeles en una gran imagen probablemente no importará a los usuarios, especialmente si emplean el espacio compartido para elaborar ideas, más que para producir un documento [Tang, 1991] [Greenberg et al., 1992], e.g., el editor de dibujos GroupSketch [Roseman and Greenberg, 1996] ignora la concurrencia [Greenberg and Bohnet, 1991] [Greenberg et al., 1992] y ningún usuario ha notado las pequeñas inconsistencias ocasionales en las imagenes. Sin embargo, esta afirmación no es verdadera en toda situación. Si las diferencias se vuelven evidentes (e.g., si se utiliza una brocha gruesa) o si la integridad de los datos en una

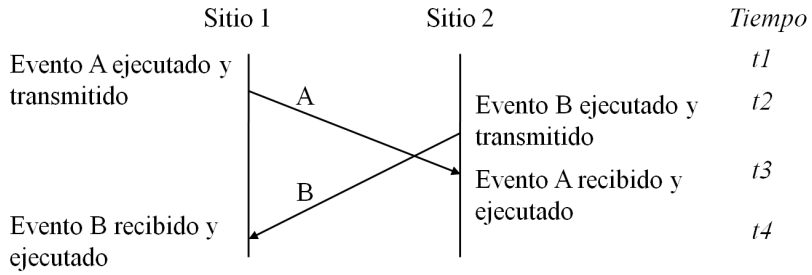


Figura 2.2: Conflicto de orden que origina una pérdida en la integridad de los datos

imagen debe ser mantenida (e.g., si el documento final es importante), entonces será necesario utilizar alguna política de control de concurrencia.

2.1.2 Las personas concilian sus propias acciones

Las personas siguen naturalmente **protocolos sociales** para mediar sus interacciones, e.g., la toma de turnos en conversaciones y la forma en que manejan objetos físicos compartidos. En una herramienta de edición o de dibujo compartido, los colaboradores perciben lo que sus colegas están haciendo y usualmente no efectúan ninguna acción que pudiera interferir con ellos. Stefik Bobrow et at. [Stefik et al., 1987] notaron este comportamiento en la manera en que las personas utilizaban COLAB, un cuarto de reunión cara a cara soportado por computadora. Sin importar si los colaboradores están utilizando un pizarrón blanco compartido real o colaborativo, sería descortés hacer garabatos sobre las marcas de otro, mientras las está escribiendo, u ocuparse en un juego de tira y afloja con un crayón. Por supuesto, existen situaciones en las que pueden ocurrir conflictos, tales como:

- Interferencia accidental debido a que una persona no se dio cuenta de lo que otro estaba haciendo;
- Efectos colaterales de una acción que tienen consecuencias en otras personas (e.g., la introducción de texto en un punto puede ocasionar la re-paginación del documento, afectando la vista de otra persona);
- Cambios intencionales de control (interrupciones o luchas de poder).

En algunas aplicaciones colaborativas, los conflictos de concurrencia pueden ser raros debido a que las personas negocian entre ellas. Cuando aparecen ligeras inconsistencias, estas podrían no ser problemáticas desde el punto de vista práctico. A fin de cuentas, si las personas notan conflictos o problemas, con frecuencia son suficientemente capaces de reparar sus efectos negativos y considerarlos como parte del dialogo natural. De esta manera, el rol de la computadora consiste en proveer suficiente retroacción de los recursos compartidos para soportar las habilidades naturales de las personas. En este caso, si se utiliza una política de control de concurrencia, esta solo se volvería un medio para evitar o resolver conflictos que rara vez ocurren [Stefik et al., 1987].

La herramienta de reunión cara a cara COLAB utiliza retroacción visual para mostrar los objetos de la pantalla que están ocupados [Bernstein et al., 1987]. Si un objeto es seleccionado por

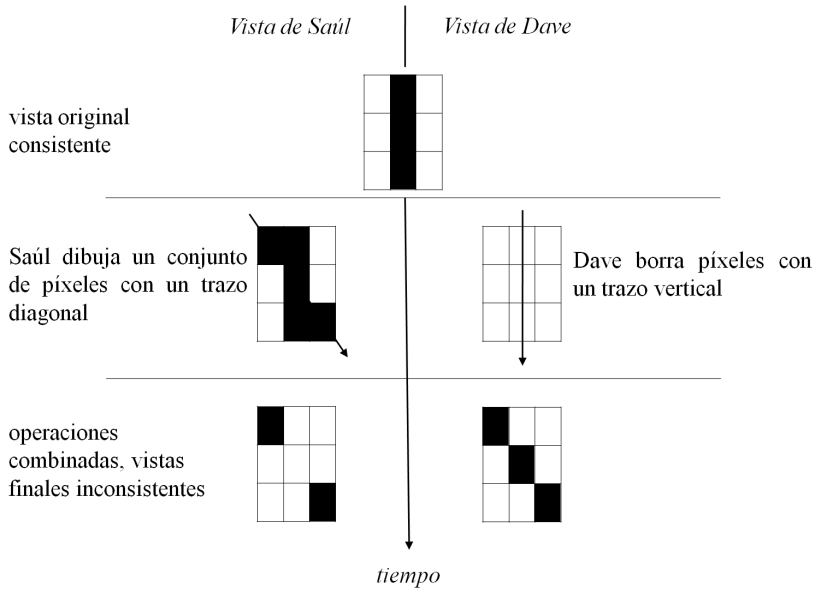


Figura 2.3: Vista de un dibujo que muestra una pequeña región de 3x3 pixeles. Dos usuarios están pintando conjuntamente un *bitmap* sin embargo, debido a eventos fuera de orden, se produce una inconsistencia en un pixel. Finalmente, no se cumplen los objetivos ni se satisface a los usuarios.

un participante, dicho objeto será mostrado en color gris para indicar a otros que no lo toquen. TRIVOLI es un sistema de dibujo compartido que soporta pequeños grupos [Moran et al., 1992]. Sus diseñadores también notaron que los conflictos son eventos raros y usualmente traen consecuencias menores. Incluso, se preocuparon por la concurrencia. A causa de la simplicidad en la manipulación de los dibujos compartidos, los diseñadores decidieron tratar los trazos de dibujo como "objetos inmutables". Si un objeto está sujeto a operaciones concurrentes, se produce un nuevo objeto para que cada usuario trabaje en su propia copia. Esta no es solo una solución técnica ya que ambos sitios están consistentes, sino también una solución humana ya que los usuarios pueden ver ambas copias y reparar el dibujo si es necesario.

2.2 Plasticidad de interfaces de usuario

La plasticidad de interfaces de usuario es una propiedad de los sistemas interactivos, que se inspira en la propiedad de los materiales que les permite expandirse y contraerse bajo restricciones naturales, sin romperse. Aplicada a interfaces de usuario, la plasticidad es la capacidad de adaptarse a cambios en el contexto de uso, preservando un conjunto de criterios de calidad como la usabilidad y la continuidad de interacción [Calvary et al., 2007] [Calvary et al., 2001b]. Particularmente, la usabilidad de un sistema interactivo se evalúa contra un conjunto de propiedades seleccionadas en las fases tempranas del proceso de desarrollo. Por lo tanto, se considera que una interfaz de usuario preserva su usabilidad si las propiedades seleccionadas se mantienen dentro de un rango predefinido de valores cuando ocurre la adaptación a cambios contextuales.

Calvary et al. identifica dos principales percutores de la plasticidad [Calvary et al., 2007]: redistribución y remodelación. La **redistribución** se refiere a la dispersión de la interfaz de

usuario en diferentes dispositivos. Cuatro tipos de redistribución son posibles: 1) de **centralizada** a **centralizada**, la cuál conserva el estado centralizado de una interfaz de usuario, e.g., migración total de una aplicación de una PC a una PDA; 2) de **centralizada** a **distribuida**, que desmiembra una interfaz de usuario con el fin de hacerla pasar de un estado centralizado a un estado distribuido, e.g., el sitio Web Sedan-Bouillon [Balme et al., 2005] puede dividirse o duplicarse parcialmente entre una PC y una PDA cuando el primer dispositivo detecta la cercanía del segundo; 3) de **distribuida** a **centralizada**, la cual reconcentra la interfaz de usuario en un único dispositivo; y 4) de **distribuida** a **distribuida**, que cambia el estado de distribución de la interfaz de usuario, e.g., una aplicación que se está ejecutándo simultaneamente en una PC y una PDA se podría redistribuir a una PC, a una PDA y a un pizarrón interactivo.

Con el fin de ilustrar la plasticidad por redistribución, considere la metáfora del pintor (cf. Figura 2.4) que es una aplicación de la técnica de manipulación directa *"pick and drop"* desarrollada por el Laboratorio de Ciencias de la Computación de Sony [Rekimoto, 1997]. La técnica *"pick-and-drop"* es un concepto extendido de la técnica *"drag and drop"* para un entorno computacional que utiliza un lápiz como dispositivo de entrada. Desde el punto de vista de un usuario inmerso en este tipo de entorno, la utilización de un lápiz le ofrece ventajas porque le permite manipular varios dispositivos computacionales empleando un mismo dispositivo de entrada.

La técnica de *"pick and drop"* permite transportar un objeto virtual de una computadora a otra mediante un artefacto físico. De esta manera, el usuario tiene la sensación de "levantar" dicho objeto virtual del despliegue de una computadora (tal como si fuera un objeto físico) y de "soltarlo" en otro despliegue. Particularmente, la aplicación de la metáfora del pintor (cf. Figura 2.4) emplea: 1) un pizarrón interactivo (como lienzo) donde el usuario puede dibujar y 2) un dispositivo móvil de tipo *palmtop* (como paleta de pintura de óleo) donde el usuario puede seleccionar un tipo de brocha, un color, una imagen o simplemente un conjunto de letras.

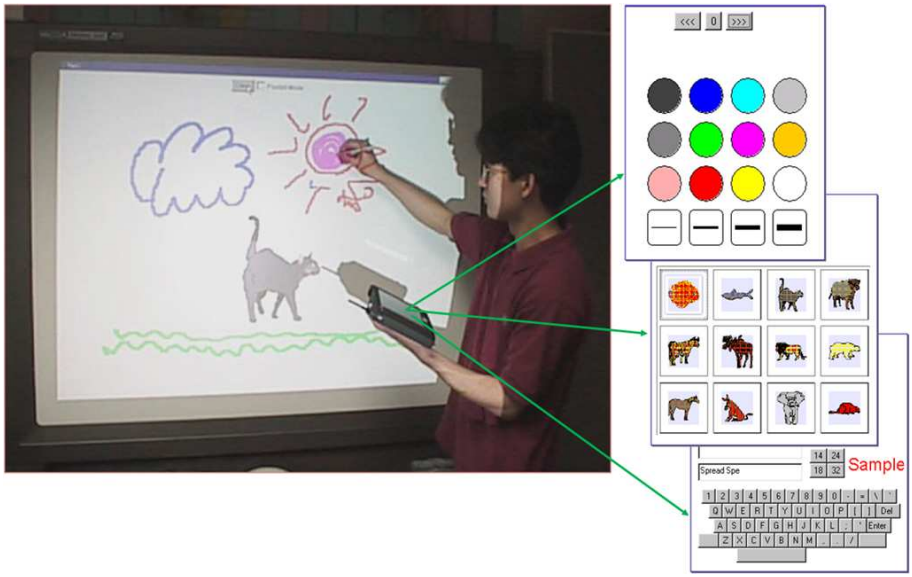


Figura 2.4: Metáfora del pintor mediante la técnica *"pick and drop"*

Por otra parte, la **remodelación** actúa sobre la presentación de los conceptos del dominio y de las tareas del usuario. Las investigaciones referentes a la plasticidad de interfaces de usuario

comenzaron precisamente por la remodelación. Uno de los primeros resultados fue la identificación de los niveles de abstracción donde la adaptación puede tener lugar [Thevenin, 2001]. Se aprovechó el modelo de la arquitectura ARCH para distinguir cinco niveles de abstracción: 1) el núcleo funcional (NF), 2) el adaptador del núcleo funcional (ANF), el controlador de diálogo (CD) y las presentaciones lógica y física (PL y PF, respectivamente) [Phillips, 1999]. En aquellos tiempos, los investigadores consideraban un solo cuadro gráfico (i.e. la pantalla de la PC). A partir de entonces, otras modalidades (e.g., la vocal) pueden ser consideradas [Berti and Paternò, 2005]. En consecuencia, se vuelve pertinente precisar si el conjunto de modalidades de un sistema será preservado o no durante la remodelación. A este respecto, se distinguen tres tipos de remodelación:

- **intra-modal:** la modalidad se preserva, e.g., de gráfico a gráfico,
- **inter-modal** o **transmodal:** la modalidad cambia, e.g. de gráfico a vocal, y
- **multimodal:** las modalidades se combinan, e.g., gráfica y vocal.

Con el fin de ilustrar la plasticidad por remodelación, considere un teclado numérico virtual [Calvary et al., 2002] que puede sustituir al teclado físico, cuando este último no está disponible. La entrada de números al sistema puede cambiar automáticamente del teclado físico al teclado virtual y viceversa, durante la ejecución del sistema.

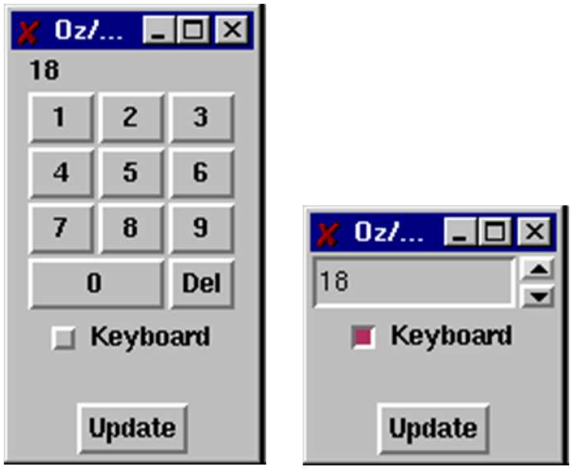
El teclado numérico virtual remodela su interfaz de usuario como consecuencia de la presencia o ausencia del teclado físico. Cuando este está ausente se muestra la interfaz de usuario de la Figura 2.5.a, la cual está compuesta de trece botones que tienen la finalidad de formar la cantidad numérica deseada. Por el contrario, cuando el teclado físico está presente se muestra la interfaz de usuario de la Figura 2.5.b, la cual está formada de los siguientes componentes gráficos: a) dos íconos (un triángulo hacia arriba y otro hacia abajo) para seleccionar el número requerido, b) un botón para activar/desactivar el teclado físico y c) un botón para actualizar el despliegue.

2.2.1 Usabilidad

La Organización Internacional para la Estandarización (ISO) ofrece dos definiciones de usabilidad en sus especificaciones [Bevan, 1999]: ISO/IEC 9126 y ISO/IEC 9241.

1. **ISO/IEC 9126**: la usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido y atractivo para el usuario, en condiciones específicas de uso. Esta definición hace énfasis en los atributos internos y externos de un producto, los cuales contribuyen a su funcionalidad y eficiencia. La usabilidad depende no sólo del producto sino también del usuario. Por esta razón, un producto no es en ningún caso intrínsecamente usable, ya que sólo tendrá la capacidad de ser utilizado en un contexto específico por usuarios particulares. La usabilidad no puede ser valorada estudiando un producto de manera aislada [Bevan and Macleod, 1994].

2. **ISO/IEC 9241**: usabilidad es la eficacia, eficiencia y satisfacción con la que un producto permite que un usuario particular alcance sus objetivos en un contexto de uso específico. Esta definición se centra en el concepto de calidad de uso, i.e., se refiere a cómo el usuario realiza con efectividad tareas particulares en escenarios específicos.



a) b)

Figura 2.5: Interfaz de usuario del teclado numérico virtual cuando el teclado físico está: a) ausente y b) presente

A partir de la conceptualización propuesta por ISO, se puede obtener los tres principios en los que se basa la usabilidad:

1. **Facilidad de aprendizaje:** se refiere a la facilidad con la que nuevos usuarios pueden tener una interacción efectiva con la aplicación. Este principio está relacionado con la predicibilidad, la sintetización, la familiaridad, la generalización de conocimientos previos y la consistencia.

2. **Flexibilidad:** hace referencia a la variedad de posibilidades en las que el usuario y el sistema pueden intercambiar información. También abarca la posibilidad de diálogo, la multiplicidad de vías para realizar una tarea, la similitud con tareas precedentes y la optimización del intercambio de información entre el usuario y el sistema.

3. **Robustez:** es el nivel de apoyo al usuario que facilita el cumplimiento de sus objetivos. Este principio está relacionado con la capacidad de observar al usuario, de recuperar información y de ajustar la tarea al usuario.

De las definiciones anteriores se concluye que la usabilidad se refiere al grado en el que el diseño de un sistema facilita o dificulta su manejo al usuario.

2.2.2 Proceso de adaptación plástica

El proceso de adaptación plástica consiste en los siguientes cinco pasos [Calvary et al., 2001b]:

1. **Detección de variaciones en el contexto de uso:** capta el entorno físico en el que se ejecuta un sistema.

2. **Identificación de las soluciones candidatas:** identifica las interfaces de usuario candidatas que sean adecuadas al nuevo contexto de uso. Dos técnicas de identificación son

susceptibles de implementación: a) cálculo en tiempo de ejecución (*on the fly*); y b) selección de un conjunto precalculado de interfaces de usuario.

3. **Selección de una solución particular:** usa alguna estrategia de resolución de problemas para elegir la interfaz de usuario más adecuada.

4. **Transición entre estados:** realiza la transición de la interfaz de usuario actual hacia la nueva solución seleccionada. Aunque la transición entre estados ha sido analizada desde los primeros desarrollos en interfaces hombre-máquina (e.g., uso de animación gráfica para transferir la carga cognitiva a nivel perceptivo), este tema de investigación está abierto [Robertson et al., 1991].

5. **Ejecución de la nueva solución:** ejecuta la nueva interfaz de usuario. Dos técnicas de ejecución han sido identificadas, en donde la interfaz de usuario puede: a) ser ejecutada desde el principio o b) preservar el estado del sistema.

La adaptación de una interfaz de usuario puede ser a la plataforma, al entorno o ambos (cf., Figura 2.6). En este sentido se pueden distinguir dos tipos de plasticidad:

- **Semi-plasticidad**: sólo realiza la adaptación al entorno o a la plataforma y;
- **Plasticidad completa**: lleva a cabo la adaptación tanto al entorno como a la plataforma.

A su vez la adaptación puede ser realizada por la aplicación, el usuario o ambos. A este respecto, se distinguen tres tipos de plasticidad:

- **Adaptativa**: la aplicación es capaz de manejar el proceso de adaptación plástica sin intervención humana,
- **Adaptable**: el usuario realiza manualmente el proceso de adaptación plástica y;
- **Mixta**: cubre una combinación de intervenciones tanto del usuario como de la aplicación.

Con base en esta clasificación de plasticidad, el interés de nuestra investigación se centra únicamente en la adaptación a la plataforma desempeñada por la aplicación, lo que origina un tipo de semi-plasticidad adaptativa (cf. Figura 2.6). Aunque dicha clasificación de plasticidad no contempla la adaptación al usuario, esta ha sido tomada en cuenta en el análisis comparativo de las aplicaciones descritas en el presente capítulo. Finalmente, es necesario precisar que no se abordaron los tipos de plasticidad completa, ya que requieren un estudio más profundo que podría tomar varios años.

2.3 Trabajos relacionados

En el presente capítulo se analizan cinco aplicaciones representativas del estado del arte. La primera es una aplicación de control de calefacción, la cual se relaciona con el presente trabajo en cuanto a que incorpora un mecanismo semi-plástico adaptativo, ya que el proceso de adaptación es desempeñado por la aplicación en función de la plataforma. La segunda

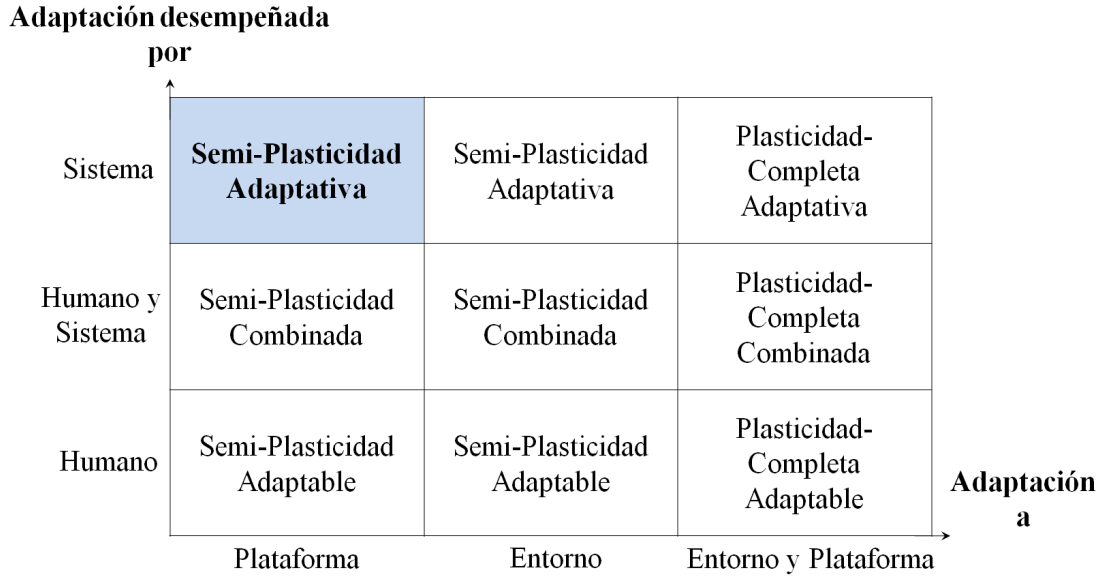


Figura 2.6: Tipos de plasticidad

aplicación, *Flexclock*, también incorpora un mecanismo semi-plástico adaptativo, cuyo principal aporte al presente trabajo es dar una orientación en el diseño del algoritmo de adaptación plástica.

La tercer aplicación, *CICERO*, ofrece a los visitantes de un museo una visita guiada interactiva y un conjunto de juegos colaborativos que estimulan tanto la cooperación, como el aprendizaje de los visitantes. La cuarta aplicación descrita es *UbiDraw*, un editor de dibujo que reubica los componentes gráficos de su interfaz de usuario de acuerdo a la tarea desempeñada por el usuario. Finalmente, la aplicación *CamNote* tiene la capacidad de migrar su interfaz de usuario de una *PC* a una *Pocket PC* mientras preserva la continuidad de interacción con el usuario.

Cabe señalar que de las cinco aplicaciones descritas en la presente sección, únicamente *CICERO* ofrece un soporte para la colaboración entre usuarios. Las demás aplicaciones son monousuario.

2.3.1 Control de calefacción

La aplicación de "Control de Calefacción" [Coutaz and Calvary, 2008] propuesta por FEC (*French Electricity Company*) se utiliza para consultar y modificar la temperatura de una habitación en particular. Dicha aplicación puede ser accedida por medio de diversos dispositivos de cómputo para poder controlar la calefacción: a) desde la casa mediante un dispositivo montado en la pared o una PDA conectada a una red inalámbrica, b) desde la oficina, a través de la Web, empleando una PC o c) desde cualquier lugar haciendo uso de un teléfono móvil o un reloj. Las Figuras 2.7 y 2.8 muestran las diferentes versiones de la interfaz de usuario de esta aplicación, considerando dos habitaciones:

- En la Figura 2.7.a, se muestra que el tamaño de la pantalla de una PC es suficientemente grande como para mostrar el estado de los termostatos de ambas habitaciones.

- En las Figuras 2.7.b, 2.7.c y 2.7.d, se observa que el tamaño de la pantalla de una PDA permite la presentación de un único termostato a la vez. Por lo tanto, una función de navegación es necesaria para acceder al segundo termostato. Particularmente, en la Figura 2.7.b, se aprecia que la tarea de navegación se apoya en un botón, mientras que en las Figuras 2.7.c y 2.7.d, se utilizan respectivamente una lista desplegable e hiperligas para interactuar con la aplicación. Finalmente, en la Figura 2.7.e, la pantalla del reloj es tan pequeña que la interfaz de usuario se limita a mostrar la temperatura de la sala, el lugar más importante de la casa. Las opciones de ajuste de temperatura y las tareas de navegación (i.e., cambiar de una vista a otra) no están disponibles.

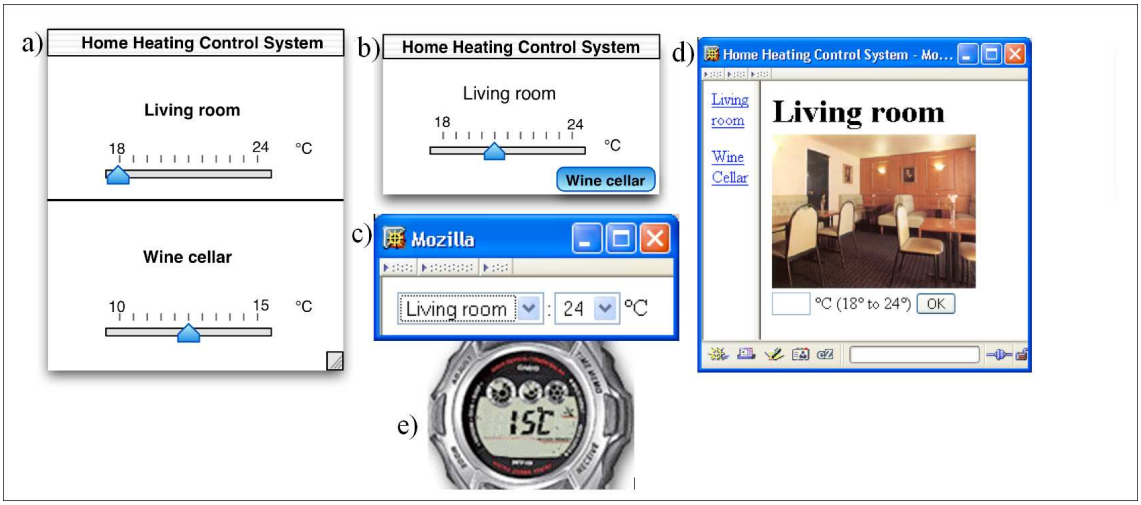


Figura 2.7: Aplicación de "Control de Calefacción": a) en la pantalla de una PC, la temperatura de ambas habitaciones está disponible en un solo vistazo; b), c) y d) en la pantalla de una PDA, únicamente la temperatura de una habitación se muestra a la vez, pero el estado de la aplicación es modificable y navegable a través de un botón (b), una lista desplegable (c) o hiperligas (d) y e) en la pantalla de un reloj, sólo la temperatura de la habitación principal es observable

La Figura 2.8 muestra la secuencia de pasos para ajustar la temperatura de una habitación mediante un teléfono celular habilitado con *WAP* (*Wireless Application Protocol*) [Forum, 1998]: 1) un usuario selecciona la opción "sala" (*le salon*, cf. Figura 2.8.a), 2) en respuesta, la aplicación muestra la temperatura actual de esta habitación (cf. Figura 2.8.b) y 3) finalmente, el usuario modifica la temperatura de la sala mediante la función "editar" (*donner ordre*, cf. Figura 2.8.c). Con la finalidad de recordar al usuario su ubicación actual en el espacio de navegación, se añadió el nombre correspondiente a cada vista de la aplicación.

2.3.2 *Flexclock*

La aplicación *Flexclock* [Grolaux et al., 2002] es un reemplazo del reloj del sistema UNIX, cuyo objetivo es mostrar la hora y la fecha en diferentes formatos de acuerdo al tamaño de la ventana. Estos formatos van desde una simple cadena *hh:mm:ss* hasta una vista más compleja compuesta de un reloj análogico, un reloj digital y un calendario mensual. *Flexclock* ofrece 17 posibles vistas (cf. Figura 2.9), las cuales existen de manera simultánea i.e., desde la perspectiva

a) b) c)

Figura 2.8: Interfaz de usuario de la aplicación de "Control de Calefacción" en un teléfono celular: a) el usuario selecciona la opción "sala", b) la aplicación indica la temperatura actual de este cuarto y c) el usuario modifica dicha temperatura

de la aplicación no hay diferencia entre la vista visible y las vistas ocultas al usuario. La aplicación *Flexclock* se caracterizan por:

1. **lo que muestra en las vistas**: por medio del lenguaje de programación *Oz* y de la herramienta *QTk* se define el estado de las vistas, i.e., componentes gráficos y geometría.

2. **cómo actualiza la hora en las vistas**: un procedimiento que toma el tiempo como parámetro se encarga de actualizar las vistas cada segundo.

3. **cómo escoge una vista**: cada vez que un usuario cambia el tamaño de la ventana de *Flexclock*, se activa un mecanismo basado en eventos, el cual aplica una regla de selección. Cada vista define los mínimos ancho y alto (en pixeles) requeridos para ser mostrada. Esta información es suficiente para que dicho mecanismo seleccione, entre todas las vistas, aquella que puede desplegarse en un espacio disponible.

El contenido de una ventana cambia de acuerdo a su tamaño. El alto y el ancho de dicha ventana determinan su presentación. Cada reajuste del tamaño de la ventana dispara la identificación de la mejor región, i.e., el área en donde una vista puede encajar. Como varias regiones candidatas pueden ser identificadas (cf. Figura 2.10 #1), es necesario seleccionar solo una región con ayuda de la siguiente fórmula:

$$d = min_{x,y}\|(d_1, d_2), (x, y)\|_2 = min_{x,y}\sqrt{(d_1 - x)^2 + (d_2 - y)^2} \tag{2.1}$$

donde (d_1, d_2) son las coordenadas de la esquina superior derecha de la ventana en el plano y (x, y) son las coordenadas de la esquina superior derecha de las regiones candidatas. La distancia d representa la distancia en Norma-2[1] entre la esquina superior derecha de la ventana

[1]La Norma-2 o norma euclidiana es una norma para medir distancias, la cual considera que la distancia más corta entre dos puntos es la recta que los une.

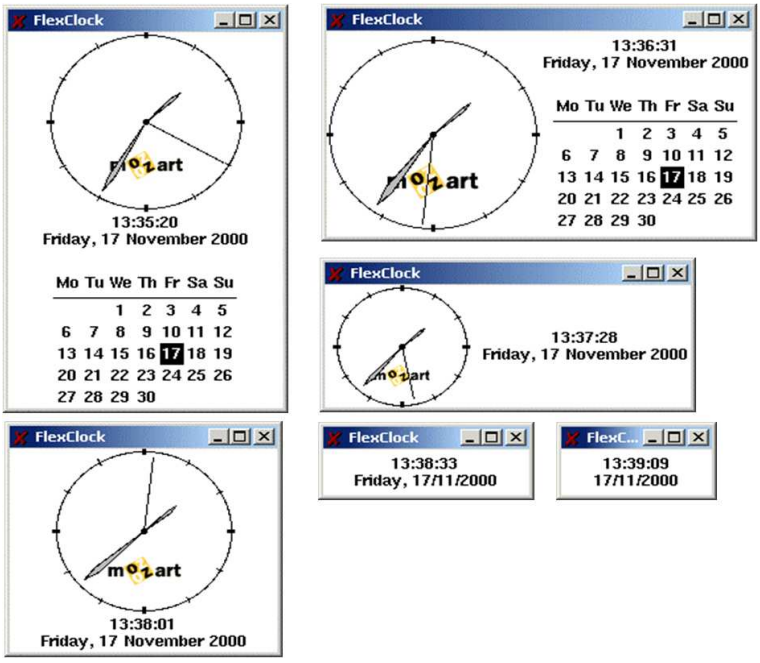


Figura 2.9: Algunas posibles vistas de *Flexclock*

y la esquina superior derecha de las regiones candidatas. Existe un umbral mínimo en la esquina inferior derecha del plano (cf. Figura 2.10 #2) debajo del cual es imposible mostrar la ventana con algún contenido.

Para poder adaptarse, cada vista define un tamaño mínimo, pero no un máximo. En consecuencia, los componentes gráficos se autoconfiguran de acuerdo al gestor de geometría utilizado, e.g., el reloj digital puede centrarse en el espacio destinado a él, mientras que el reloj análogico puede desplegarse lo más grande posible. Así, si el usuario cambia el tamaño de la ventana de tal forma que siempre se muestre la misma vista, esta tomará ventaja del tamaño disponible tanto como sea posible.

En la Figura 2.11 se muestra un plano dividido en dos regiones por una línea diagonal que inicia en la coordenada (0,0). Con base en la formula 2.1, cualquier ventana cuya esquina superior derecha se encuentre en la región superior de la diagonal, se asociará a una configuración de vista vertical (cf. Figura 2.11 #1), mientras que cualquier ventana cuya esquina superior derecha se encuentre en la región inferior de la diagonal, se asociará a una configuración de vista horizontal (cf. Figura 2.11 #2).

2.3.3 *Cicero*

En la última década, las aplicaciones móviles de guías de museos han incentivado numerosos trabajos tales como los que proponen Abowd, Cheverst, Oppermann et al. [Abowd et al., 1997] [Cheverst et al., 2000] [Oppermann and Specht, 2000]. Una visita al museo es a menudo una experiencia individual. Incluso las guías electrónicas y los kioskos interactivos no están diseñados para promover la interacción entre usuarios e incrementar sus experiencias. Por lo tanto, la incorporación de juegos en aplicaciones móviles de guías de museos provee un medio para alentar la colaboración entre usuarios.

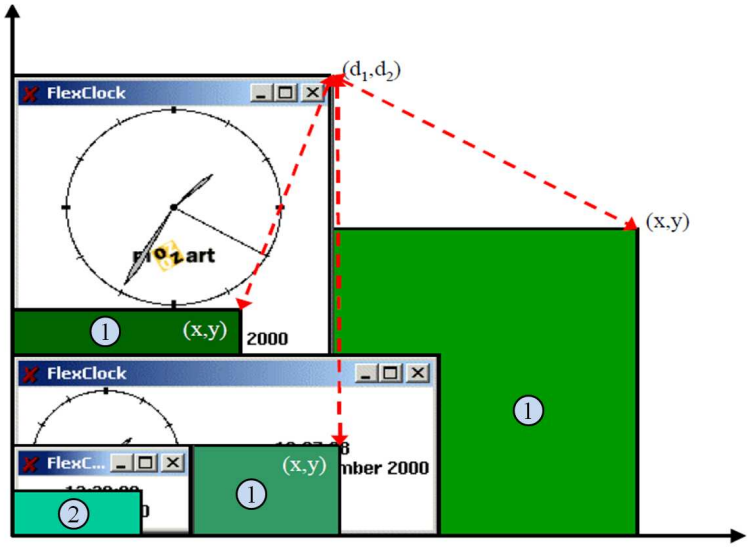


Figura 2.10: Regiones candidatas para la nueva presentación antes del reajuste

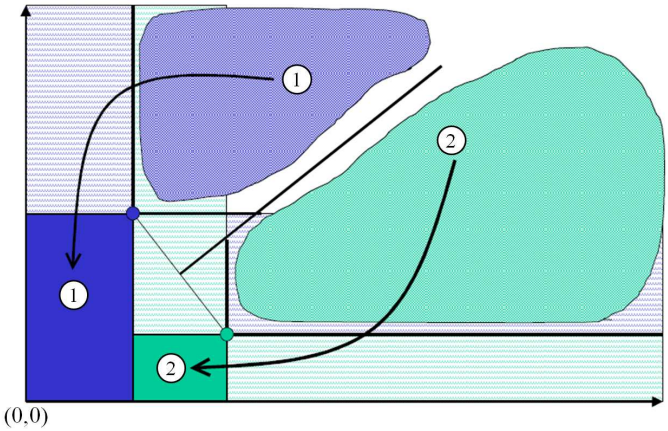


Figura 2.11: Configuraciones vertical y horizontal de las vistas

Los autores de los proyectos *Equator* [Brown et al., 2003] y *Sotto Voce* [Grinter et al., 2002] notaron que la información sobre la ubicación de los individuos es esencial en una visita colectiva a un museo para mantener la conciencia de grupo. Este requerimiento ha sido considerado en *Cicero* [Ciavarella and Paternò, 2004], una aplicación que provee información de las piezas de arte del múseo de Mármol de Carrara. El escenario típico de uso consiste en que los usuarios se muevan libremente por el museo e interactúen a través de dispositivos móviles. Adicionalmente, *Cicero* [Bell et al., 2006] [Danesh et al., 2001] utiliza despliegues públicos para mostrar la localización del visitante y las diferentes maneras de explotar juegos individuales y colectivos.

Los módulos principales de la arquitectura del software de *Cicero* son: 1) la aplicación para la PDA, 2) la aplicación para el despliegue público y 3) el protocolo de comunicación que permite la interacción entre los dispositivos de cómputo utilizados por los visitantes del museo. La aplicación para la PDA se componen de cuatro capas, cada una provee servicios a las demás:

1. La **capa de núcleo** implementa estructuras de información útiles para las capas adyacentes. En particular, esta capa contiene un administrador de concurrencia de señales IRDA (señales infrarrojas que detectan cuando un usuario entra a una sección del museo).

2. La **capa de comunicación** actualiza la información que se muestra en los despliegues públicos, poniendo énfasis en el estado de los jugadores. Además, esta capa implementa algoritmos de distribución, los cuales permiten a los jugadores que utilizan una PDA organizarse en equipos. Esta capa es utilizada por las capas de visitas y de juegos.

3. La **capa de visitas** soporta la presentación del mapa del museo y los componentes gráficos de interacción. Cada obra de arte está asociada a un icono que identifica a su tipo (e.g., escultura o pintura) y está posicionada en el mapa de acuerdo a su localización física. Mediante la selección de un icono, los usuarios pueden recibir información detallada acerca de la obra de arte correspondiente. Además, la capa de visitas permite a los usuarios recibir ayuda, acceder a videos sobre las obras de arte, cambiar parámetros de audio (e.g., ajustar el volúmen) y obtener información sobre las obras visitadas por otros jugadores.

4. La **capa de juegos** tiene la finalidad de extender la aplicación de guía de museo. Los juegos se definen mediante representaciones basadas en XML para permitir modificaciones fáciles y extensiones. Esta capa utiliza tanto los servicios de la capa de núcleo como los de la capa de comunicación para informar a todos los jugadores sobre la actualización del puntaje.

Un dispositivo puede monitorear un equipo de jugadores sin necesidad de conocer las direcciones IP de los demás dispositivos. Cuando los dispositivos reciben un mensaje, verifican si está dirigido a ellos: en caso positivo envían una respuesta, de lo contrario el mensaje se descarta. La comunicación entre los dispositivos se realiza de manera unidireccional (i.e., de los dispositivos móviles a los despliegues públicos) mediante el protocolo TCP.

Cicero ofrece una aplicación llamada *CoCicero* [Laurillau and F. Paternò, 2004] que integra cinco juegos individuales y un juego colaborativo que permite la interacción entre los visitantes del museo. Los objetivos de integrar un conjunto de juegos a las aplicaciones que sirven de guía a los visitantes del museo son: 1) mostrar a dichos visitantes un nuevo ambiente, 2) hacer las visitas al museo más interactivas y 3) estimular el aprendizaje de los visitantes.

La Figura 2.12 ilustra la forma en que se extendió la guía virtual para ofrecer juegos colaborativos, así como los recursos necesarios para soportar esta clase de juegos. Los visitantes pueden organizarse en equipos (máximo cinco) para resolver los problemas propuestos en los juegos individuales, que son accesibles desde una PDA. Cada visitante está asociado a un nombre y un color. El monto total de puntos ganados por el equipo de cada jugador es visualizado en la esquina superior izquierda de la interfaz de usuario (cf. Figura 2.12). No todas las obras de arte tienen un juego asociado: si algún juego está disponible, se mostrará un icono etiquetado con una interrogación "?". Inicialmente, este icono es blanco pero si alguien entra al juego y contesta la pregunta correctamente, entonces el icono se volverá verde. Por el contrario, si la respuesta es incorrecta, el icono se volverá rojo. Esta aplicación también provee información sobre las obras visitadas por otros jugadores. Por último, existe un menú gráfico localizado en la parte inferior de la interfaz de usuario donde se pueden realizar acciones adicionales, como acceder al juego "Enigma Compartido", cuyo objetivo es motivar la interacción y cooperación

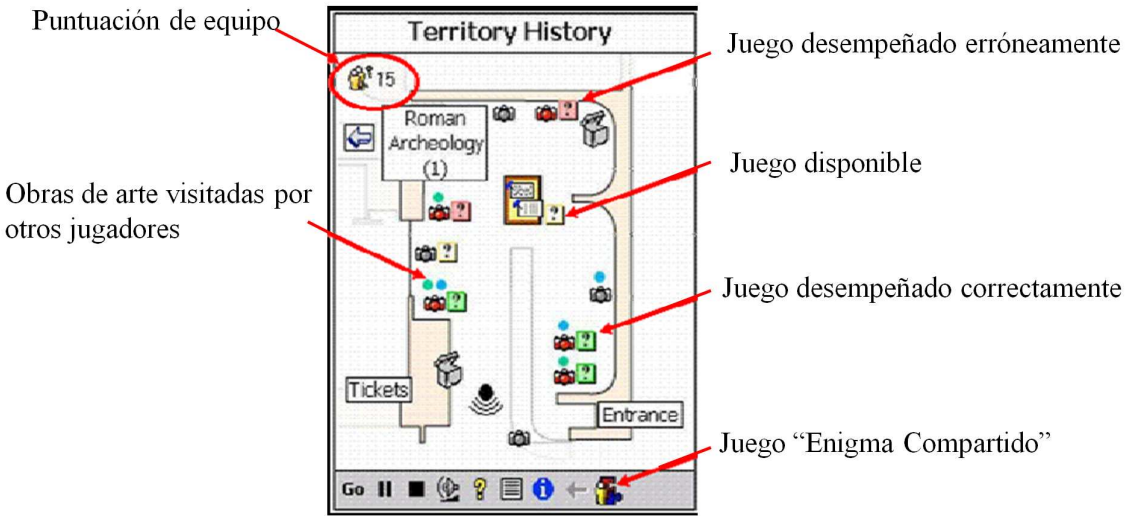


Figura 2.12: Soporte cooperativo para juegos en *CoCicero*

entre los participantes, quienes deben encontrar las respuestas a varias preguntas para resolver conjuntamente un enigma.

En particular, el juego "Enigma Compartido" contiene una serie de preguntas sobre un tópico asociado a una imagen de una obra de arte, la cual está ocultada por un rompecabezas. Cuando un jugador resuelve un problema propuesto en un juego individual, una pieza del rompecabezas se remueve para mostrar la sección cubierta de la imagen y facilitar la respuesta de la pregunta principal del juego colaborativo.

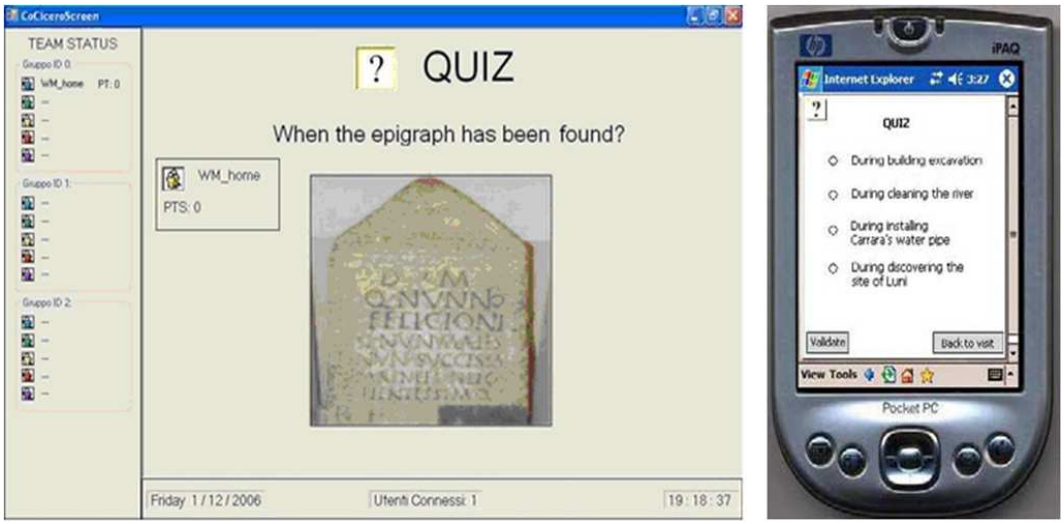


Figura 2.13: *CoCicero* en modo distribuido: despliegue público (izquierda) y PDA (derecha)

La interfaz de usuario del juego "Enigma Compartido" para una PDA se divide en dos partes visualizables de manera secuencial: (cf. Figura 1.2) la primera parte muestra la puntuación de los jugadores y la imagen escondida en el rompecabezas, en tanto que la segunda parte muestra las preguntas con sus posibles respuestas. Un juego de *CoCicero* puede ser mostrado de dos

maneras distintas (seleccionables a través de la interfaz de usuario para una PDA):

- **modo normal**: en la PDA, la interfaz de usuario no cambia, mientras que en el despliegue público se muestra tanto el estado de cada uno de los jugadores de los diferentes equipos, como el rompecabezas que oculta la imagen de la obra de arte (cf. Figura 1.3); la presentación del juego en un despliegue público se utiliza para enfocar la atención de múltiples usuarios;

- **modo distribuido**: la interfaz de usuario se divide en dos partes, de manera que las opciones de respuesta se muestran solo en la PDA, mientras que la pregunta y la imagen de una obra de arte se observa en el despliegue público (cf. Figura 2.13); el resultado de la respuesta del usuario se presenta solo en la PDA.

2.3.4 *UbiDraw*

UbiDraw fue desarrollado utilizando tanto el ambiente *Mozart* [Grolaux et al., 2001] como la herramienta gráfica *QTk*. El ambiente *Mozart* es por definición multi-plataforma, ya que permite que el código de una aplicación se ejecute de manera similar en *Linux, Windows* o *Mac*. Por su parte, la herramienta gráfica *QTk* se apoya del lenguaje de programación *Oz* para el desarrollo de aplicaciones en el ambiente *Mozart*.

UbiDraw proporciona cuatro componentes de dibujo: "Archivos" (*Fichiers*), "Dibujo" (*Dessin*), "Opciones" (*Options*) y "Retocar" (*Retoucher*) agrupados en una barra de menú (cf. Figura 2.14). Cada componente ofrece una barra de herramientas que puede ser mostrada en diferentes lugares de la ventana principal de *UbiDraw*, dependiendo del tamaño de dicha ventana. El conjunto de iconos que contiene cada barra de herramientas puede ser presentado en tres formas diferentes de acuerdo a su estatus:

1. **oculto**: ningún icono de la barra de herramientas es visible;

2. **arreglo vertical**: todos los iconos se presentan en una barra de herramientas vertical; y

3. **arreglo horizontal**: todos los iconos se muestran en una barra de herramientas horizontal.

La Figura 2.14 representa gráficamente estas tres posibles vistas de *UbiDraw*. Particularmente, la Figura 2.14.a muestra las barras de herramientas de "Archivos" y "Dibujo", mientras las barras de herramientas de "Opciones" y "Retocar" están ocultas, con el fin de maximizar el área disponible para el lienzo. La Figura 2.14.b presenta la barra de herramientas "Retocar" en forma vertical, en tanto que la Figura 2.14.c muestra las barras de herramientas "Opciones" y "Retocar" en forma horizontal. Cada barra de herramientas no necesariamente muestra todos sus iconos: su contenido puede ir de vacio (cuando su estatus es **oculto**) a completo (cuando todos los iconos son visibles de manera **vertical** u **horizontal**).

Para determinar el contenido de una barra de herramientas, *Ubidraw* utiliza un sistema de escalas de prioridad, el cual regula la presentación de los iconos con base en tres prioridades: 1) el último icono seleccionado, 2) la preferencia del usuario en un icono particular y 3) la cantidad de *clicks* realizados sobre un icono. Además, entre mayor prioridad tenga un icono, mayor tiempo será visible al usuario. De esta manera, *Ubidraw* puede determinar el tiempo

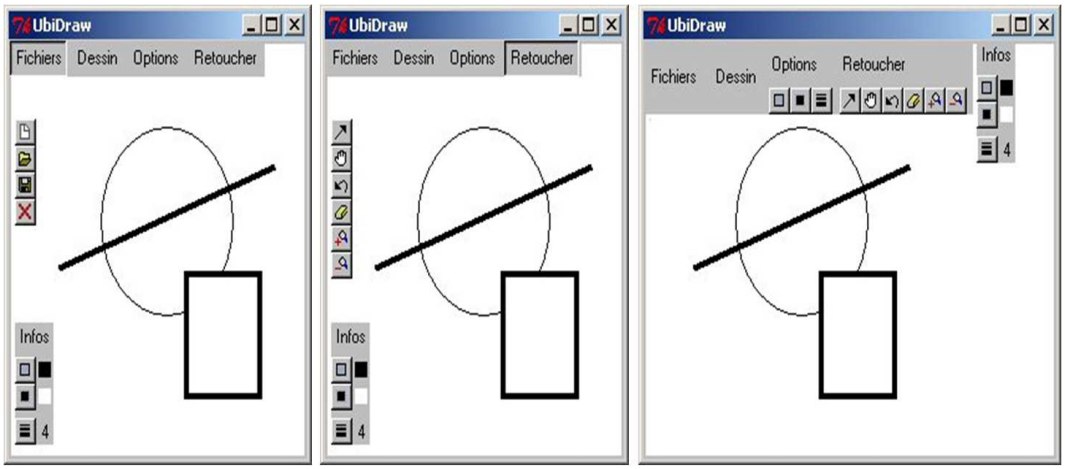


Figura 2.14: Tres vistas posibles de las barras de herramientas de *UbiDraw*

que mostrará una determinada vista de la interfaz de usuario. La Figura 2.15 reproduce una situación antes (cf. Figura 2.15.a) y después (cf. Figura 2.15.b) de aumentar horizontalmente la ventana principal de *Ubidraw*.

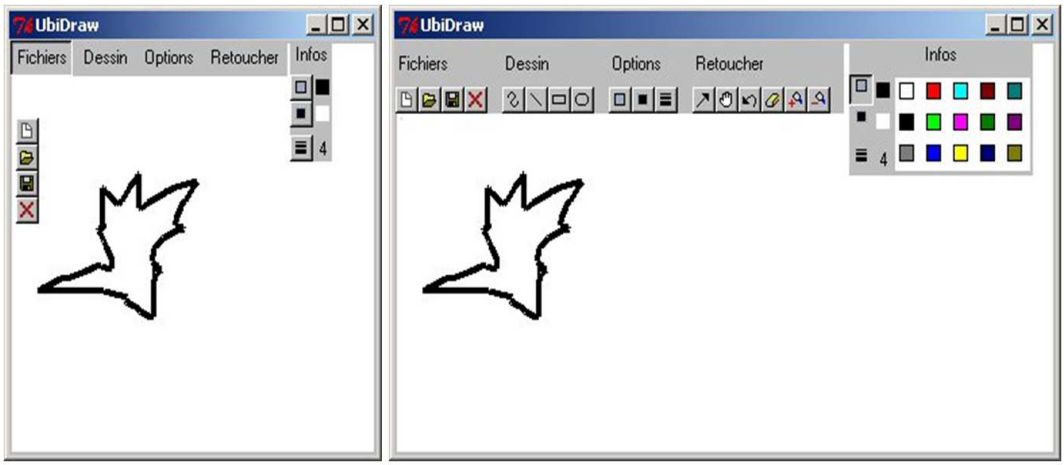


Figura 2.15: Interfaz de usuario de *Ubidraw* antes y después de aumentar la ventana principal

La aplicación *UbiDraw* está conformada de tres clases principales (cf. Figura 2.16): 1) la clase *GUI*, 2) la clase *UndoList* y 3) la clase *Dataprocess*. La clase *GUI* implementa una interfaz de usuario específica; la clase *UndoList* guarda un registro del historial de acciones realizadas; y la clase *Dataprocess* utiliza objetos de dibujo.

La interfaz de usuario de *Ubidraw* consiste principalmente de un lienzo (*customCanvas*), el cual integra un conjunto de *UbiWidgets* (i.e., componentes de dibujo y sus correspondientes barras de herramientas). El lienzo (*customCanvas*) muestra uno de los tres estados (i.e., oculto, vertical y horizontal) de cada componente *UbiWidget*, dependiendo de la clase *ContextWatcher* [Calvary et al., 2001a] [Demeure et al., 2007] [Gram and Cockton, 1996], como a continuación

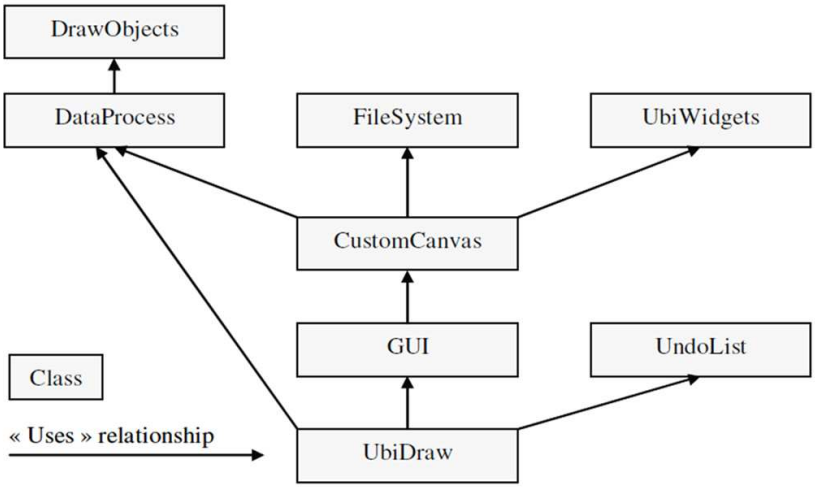


Figura 2.16: Arquitectura de software de *UbiDraw*

se explica.

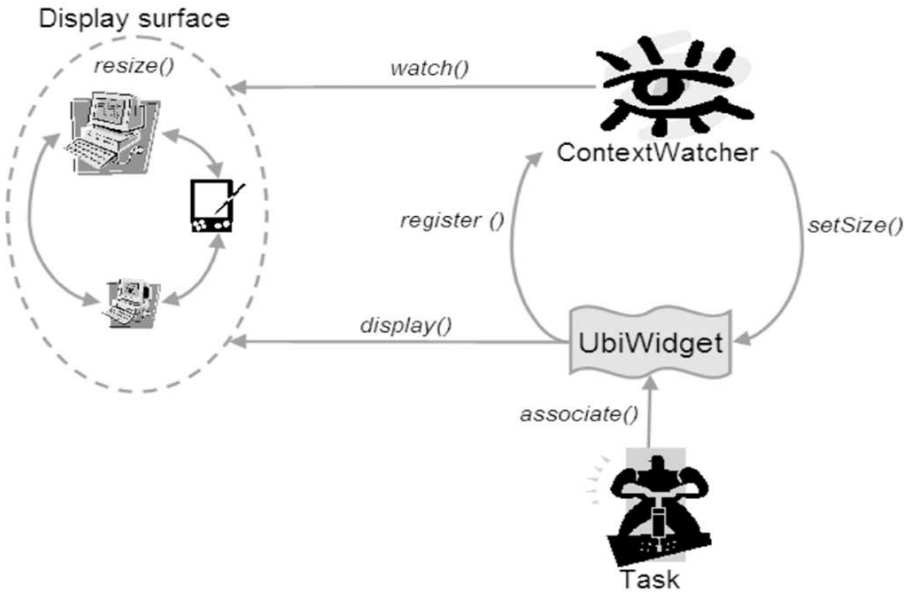


Figura 2.17: Mecanismo de adaptación plástica de *Ubidraw* en tiempo de ejecución

El componente central del mecanismo de adaptación es el componente *UbiWidget*, el cual contiene las clases *ContextWatcher* y *UbiWidget*. La clase *ContextWatcher* es responsable de la ubicación de todos los componentes gráficos que conforman la interfaz de usuario y la clase *UbiWidget* genera componentes gráficos plásticos.

Cada componente *UbiWidget* se registra en la clase *contextWatcher*, la cual le asigna un contenido inicial (cf., Figura 2.17). Dependiendo de su estatus, cada componente *UbiWidget* se muestra o bien permanece oculto. Si el contexto de uso cambia, i.e., si el tamaño de la ventana principal de *UbiDraw* se reajusta, la clase *contextWatcher* será notificada. En respuesta, esta clase realiza algunos cálculos con el fin de proponer tanto un nuevo estatus, como un nuevo

contenido para cada componente *UbiWidget*.

Para realizar el proceso de adaptación plástica de *UbiDraw* en tiempo de ejecución se realizan tres pasos:

1. **reconocimiento de la situación**: consiste en reconocer el nuevo contexto de uso, detectando e identificando los cambios que tienen lugar; cuando el tamaño de la ventana cambia, el escuchador de *UbiDraw* disparará el cálculo de una reacción;

2. **cálculo de una reacción**: consiste en identificar y seleccionar la reacción candidata; una posible reacción de *UbiDraw* consiste en recalcular el diseño de la interfaz de usuario; y

3. **ejecución de reacciones**: consiste en los pasos "preparar", "ejecutar" y "cerrar la reacción".

UbiDraw aplica instantáneamente una reacción (e.g., reajuste de los componentes gráficos) en respuesta a una acción (e.g., aumentar o disminuir el tamaño de la ventana). La adaptación siempre resulta de la iniciativa del usuario (e.g., si el usuario reajusta una ventana o utiliza una plataforma diferente). Consecuentemente, no se ha tomado ninguna precaución particular en la ejecución de la reacción ni se requiere incorporar un paso inicial, ya que la interfaz de usuario se encarga de activar la adaptabilidad después de que ocurre un cambio significativo en el contexto de uso.

La clase *contextWatcher* contiene un método llamado "observar", el cual percibe cualquier cambio en el tamaño del lienzo y en respuesta aplica la presentación apropiada. Para calcular la transformación más adecuada, la clase *contextWatcher* necesita tres datos de cada componente *UbiWidget* registrado: su contenido mínimo, su contenido máximo y el rango de la tarea que apoya (el rango establece un mecanismo de prioridad). La clase *contextWatcher* ordena los componentes *UbiWidgets* de acuerdo a su nivel de rango; por lo tanto, se mostrará primero el componente gráfico con el rango más alto. El algoritmo de ubicación siempre intentará poner el número máximo de componentes gráficos en el lienzo (*CustomCanvas*).

La clase *contextWatcher* comunica a cada componente *UbiWidget* su contenido actual y su localización en el lienzo (*CustomCanvas*). A partir de este momento, el componente *UbiWidget* puede dibujarse a sí mismo. La selección de la presentación apropiada está a cargo del método "mostrar" de la clase *GUI*.

2.3.5 *CamNote*

CamNote (CAMALEON Note) es un visor de diapositivas similar a *Microsoft Power Point*. Sin embargo, *CamNote* se puede ejecutar simultáneamente en una computadora personal y en una computadora *Pocket PC*. La demostración del funcionamiento de *CamNote* consiste en migrar el control remoto que se muestra en una *PC* a una *Pocket PC*, mientras se preserva la continuidad de interacción. Cuando la migración ocurre, se requiere una remodelación del control remoto para acomodar varios recursos de interacción, computación y comunicación. *CamNote* está formado de tres componentes (cf. Figura 2.18):

- **un visor de diapositivas**: permite la inserción de videos translucidos (conocidos como pixeles espejos [Morikawa and Maesako, 1998] [Vernier et al., 1999]);

- **un editor o visor de notas**: permite la inserción de comentarios en cada diapositiva y;

- **un control remoto**: permite la navegación a través de la presentación.

El visor de diapositivas y el editor están disponibles solo para *PCs*, en tanto que el control remoto está disponible tanto para *PC* como para *PocketPC*. En la esquina superior derecha de la Figura 2.18 se observa una ventana que muestra el visor de diapositivas, mientras que en la parte central se observa una ventana que puede ser rotada, la cual integra el control remoto para navegar a través de la presentación, el visor de notas personales y la imagen de video del expositor.

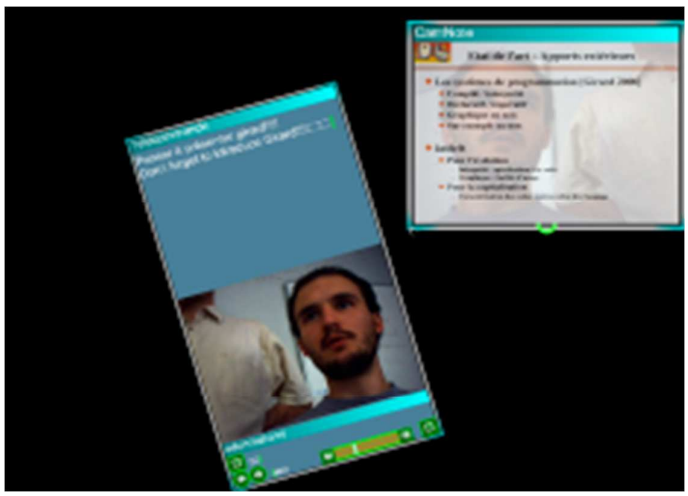


Figura 2.18: Interfaz de usuario de *CamNote* centralizada en una *PC*

Para proveer un sentido de interacción continua se utiliza un mecanismo de transición que se describe a continuación. La interfaz de usuario de *CamNote* mostrada en una *PC* puede rotar, i.e., la ventana no necesita ser paralela a los bordes de la pantalla de la *PC*. Este efecto de rotación de la interfaz de usuario se utiliza para crear la sensación de una interacción continua cuando el control remoto migra a la *Pocket PC* (cf. Figura 2.19): si el usuario aumenta el tamaño de la ventana del visor de diapositivas, entonces el control remoto empieza a desaparecer de la *PC* (cf. Figura 2.20) y aparecer en una *Pocket PC* (cf. Figura 2.21). De manera análoga, si la ventana del visor de diapositivas se reduce, la interfaz de usuario automáticamente regresa a la configuración que se muestra en la Figura 2.18.

Si una *Pocket PC* está disponible cuando el usuario está aumentando la ventana, entonces el control remoto migrará a dicho dispositivo (cf. Figura 2.21). Se requiere una remodelación de la interfaz de usuario para que pueda acomodarse en varias fuentes de interacción, computación y comunicación. Generalmente, el video ya no es observable en esta configuración del control remoto.

2.4 Análisis comparativo

En esta sección se realiza un análisis comparativo de los diferentes trabajos estudiados en el presente capítulo. Para realizar dicho análisis, se parte de los conceptos introducidos en la sección 2.1 (i.e., plasticidad de interfaces de usuario, usabilidad y proceso de adaptación

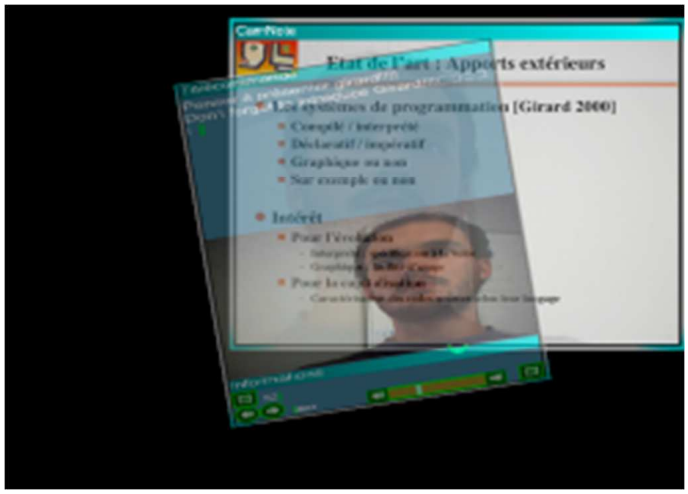


Figura 2.19: El control remoto empieza a desaparecer de la *PC* cuando el usuario aumenta la ventana del visor de diapositivas

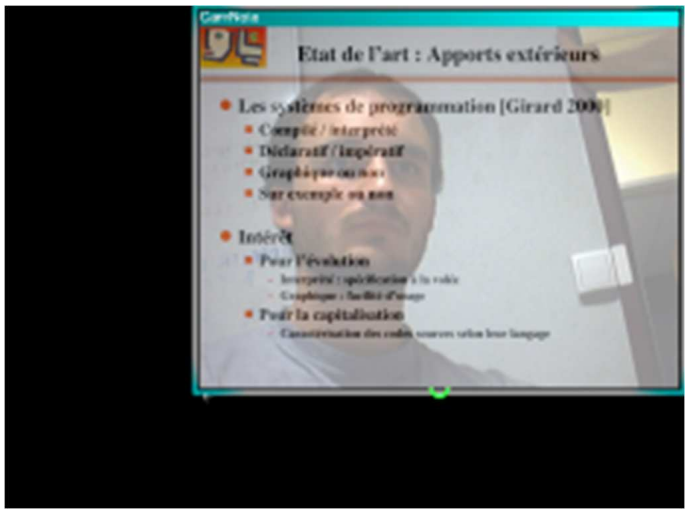


Figura 2.20: Interfaz de usuario centralizada de *CamNote* cuando el control remoto desapareció de la *PC*

plástica). En la Figura 2.22 se muestran las características plásticas más relevantes (columna izquierda) de las aplicaciones descritas a lo largo del capítulo (fila superior). Por medio de este análisis, se evidencian algunas características que no han sido contempladas por las aplicaciones estudiadas y que motivaron el desarrollo del presente trabajo de tesis de maestría.

En la Figura 2.22 primeramente se examina el **tipo de aplicación**, i.e., si es mono-usuario o colaborativa. Las aplicaciones de "Control de Calefacción", *FlexClock*, *UbiDraw* y *CamNote* son mono-usuario, ya que solo pueden ser utilizadas por un único usuario a la vez. En cambio, *CICERO* es una aplicación colaborativa, ya que soporta la interacción entre personas en tiempo real a través de sus aplicaciones de juegos colaborativos.

El **contexto de uso** se refiere a la adaptación de la interfaz de usuario a tres aspectos: a) el usuario, b) la plataforma y c) el entorno (cf. sección 2.1). Las cinco aplicaciones analizadas

Figura 2.21: Control remoto en la *Pocket PC*

se adaptan a la plataforma. Particularmente, *Flexclock* y *UbiDraw* pueden ser ejecutadas en diversos sistemas operativos (e.g., *Windows, Linux* y *Mac*), mientras que las aplicaciones de "Control de Calefacción", *CICERO* y *CamNote* son accesibles desde diversos dispositivos de cómputo (i.e., *PDAs*, despliegues públicos, *PCs*, *Pocket PCs* y teléfonos celulares).

Por otra parte, *Flexclock*, *UbiDraw* y *CamNote* también se adaptan al usuario porque la remodelación de su interfaz gráfica depende de la tarea realizada por el usuario (i.e., la interfaz gráfica cambia de presentación cuando el usuario reajusta el tamaño de la ventana). En la documentación referente a las aplicaciones de "Control de Calefacción" y *CICERO* no se especifica qué o quién desempeña el proceso de adaptación plástica. En *Flexclock*, *UbiDraw* y *CamNote* la **adaptación** es desempeñada por el sistema. Con base en los tipos de plasticidad mostrados en la Figura 2.6, *Flexclock*, *UbiDraw* y *CamNote* son clasificados como aplicaciones semi-plásticas adaptativas.

Todas las aplicaciones analizadas implementan la **remodelación** de tipo intra-modal, i.e., preserva la modalidad de gráfico a gráfico.

Por otra parte, las aplicaciones de "Control de Calefacción", *Flexclock* y *Ubidraw* soportan únicamente el **tipo de redistribución** "centralizada a centralizada", ya que su interfaz de usuario no está distribuida en más de un dispositivo. En cambio, *CICERO* y *CamNote* pueden redistribuir sus interfaces de usuario en más de un dispositivo de cómputo.

Las **interfaces de usuario** de las aplicaciones de "Control de Calefacción" y *CICERO* están prefabricadas para los diversos dispositivos de cómputo (i.e., no son generadas en tiempo de ejecución). *FlexClock* incorpora un conjunto de diecisiete interfaces de usuario previamente diseñadas, pero además genera interfaces de usuario en tiempo de ejecución, ya que aún cuando *Flexclock* elige una interfaz de usuario de las diecisiete posibles, esta aplicación puede modificar el tamaño y la alineación de sus componentes gráficos (e.g., agrandar el reloj analógico o centrar el reloj digital). Por su parte, *UbiDraw* genera sus interfaces de usuario en tiempo de ejecución, dependiendo del reajuste de la ventana principal. *CamNote* no especifica si sus interfaces de usuario están prefabricadas o son generadas en tiempo de ejecución.

Características de la aplicación	Control de calefacción	Flexclock	CICERO	UbiDraw	CamNote
Tipo de sistema	mono-usuario	mono-usuario	colaborativo	mono-usuario	mono-usuario
Adaptación a (contexto de uso)	plataforma	usuario y plataforma	plataforma	usuario y plataforma	usuario y plataforma
Adaptación desempeñada por	no especifica	sistema	no especifica	sistema	sistema
Tipo de remodelación	intra-modal	intra-modal	intra-modal	intra-modal	intra-modal
Tipo(s) de redistribución	centralizada a centralizada	centralizada a centralizada	centralizada a centralizada y centralizada a distribuida	centralizada a centralizada	centralizada a centralizada y centralizada a distribuida
Generación de interfaces de usuario	prefabricadas	prefabricadas y generadas en tiempo de ejecución	prefabricadas	generadas en tiempo de ejecución	no especifica
Preservación de usabilidad	si	si	si	si	si

Figura 2.22: Características plásticas de las aplicaciones analizadas

La aplicación de "Control de Calefacción" si conserva la **usabilidad**, puesto que los usuarios pueden consultar y modificar la temperatura de los cuartos de una casa desde una *PC*, una *PDA* o un teléfono celular. Similares a esta aplicación son *CamNote* y *CICERO*. *FlexClock* también preserva la usabilidad porque aún cuando el usuario haya cambiado las dimensiones del área de despliegue, esta aplicación continúa proporcionando la fecha y hora actual. Finalmente, *UbiDraw* es similar a *Flexclock*.

Capítulo 3

Mecanismo de remodelación semi-plástica adaptativa para interfaces colaborativas

En el presente capítulo se detalla un diseño orientado a objetos del mecanismo ReSAIC (Remodelación Semi-plástica y Adaptativa para Interfaces Colaborativas). Primeramente, se describen los casos de uso de dicho mecanismo (cf. sección 3.1). Después, se explica el funcionamiento general de ReSAIC mediante sus vistas estática y dinámica (cf. sección 3.2) así como sus principales componentes (cf. sección 3.3). Posteriormente, se presenta la arquitectura de comunicación del mecanismo ReSAIC, la cual establece la forma de llevar a cabo el intercambio de mensajes cuando sus objetos interactúan en una red (cf. sección 3.4). Enseguida, se detalla la arquitectura de distribución del mecanismo ReSAIC, la cual define la repartición de sus objetos entre los sitios implicados en un proceso de interacción (cf. sección 3.5). Finalmente, se presenta el análisis y el diseño de la aplicación de validación "Serpientes y Escaleras", que incluye la descripción de la lógica del juego, las principales clases que la conforman y la presentación de las diversas versiones de su interfaz de usuario (cf. sección 3.6).

3.1 Casos de uso del mecanismo ReSAIC

Como se mencionó en el capítulo 1, el objetivo principal de esta tesis de maestría consiste en diseñar e implementar un mecanismo semi-plástico adaptativo, llamado ReSAIC (Remodelación Semi-plástica Adaptativa para Interfaces Colaborativas), el cual adapta la interfaz de usuario de una aplicación colaborativa a las características físicas de diversas plataformas, sin intervención humana. Para lograr dicho objetivo se identificaron los casos de uso que se describen a continuación (cf. Figura 3.1).

Se tiene un conjunto de usuarios, i.e., personas que, provistas de un dispositivo de cómputo (e.g., PC, PDA o laptop) desean emplear una aplicación colaborativa en un momento determinado y en espacios geográficos distintos. Como se mencionó antes, los usuarios emplean dispositivos de cómputo heterogéneos, por lo tanto surge la necesidad de adaptar el tamaño de la interfaz de usuario a cada uno de los dispositivos, con el propósito de que cada usuario interactúe adecuadamente con la aplicación colaborativa. Para lograr lo anterior, el usuario solicita `ejecutar el núcleo funcional` de la aplicación, el cual a su vez solicita `ejecutar adaptador` para emplear la opción de `seleccionar una interfaz de`

`usuario`. Con base en esta selección, el núcleo funcional se da a la tarea de `desplegar una interfaz de usuario` adaptada al tamaño de pantalla del dispositivo.

Este caso de uso lo realizan todos y cada uno de los usuarios que desean interactuar con una aplicación colaborativa. De manera que, luego de que se despliega la interfaz de usuario adecuada, es necesario `actualizarla` cada que se produzca un evento relevante en el espacio de trabajo compartido, con la finalidad de que los usuarios estén informados de lo que acontece. El núcleo funcional es el encargado de `procesar operaciones`, las cuales se refieren a las acciones realizadas por el usuario `local` (aquel que ejecuta un núcleo funcional a partir de su propio dispositivo) y por los usuarios `remotos` (aquellos que interactúa a distancia con el usuario local mediante su propio dispositivo).

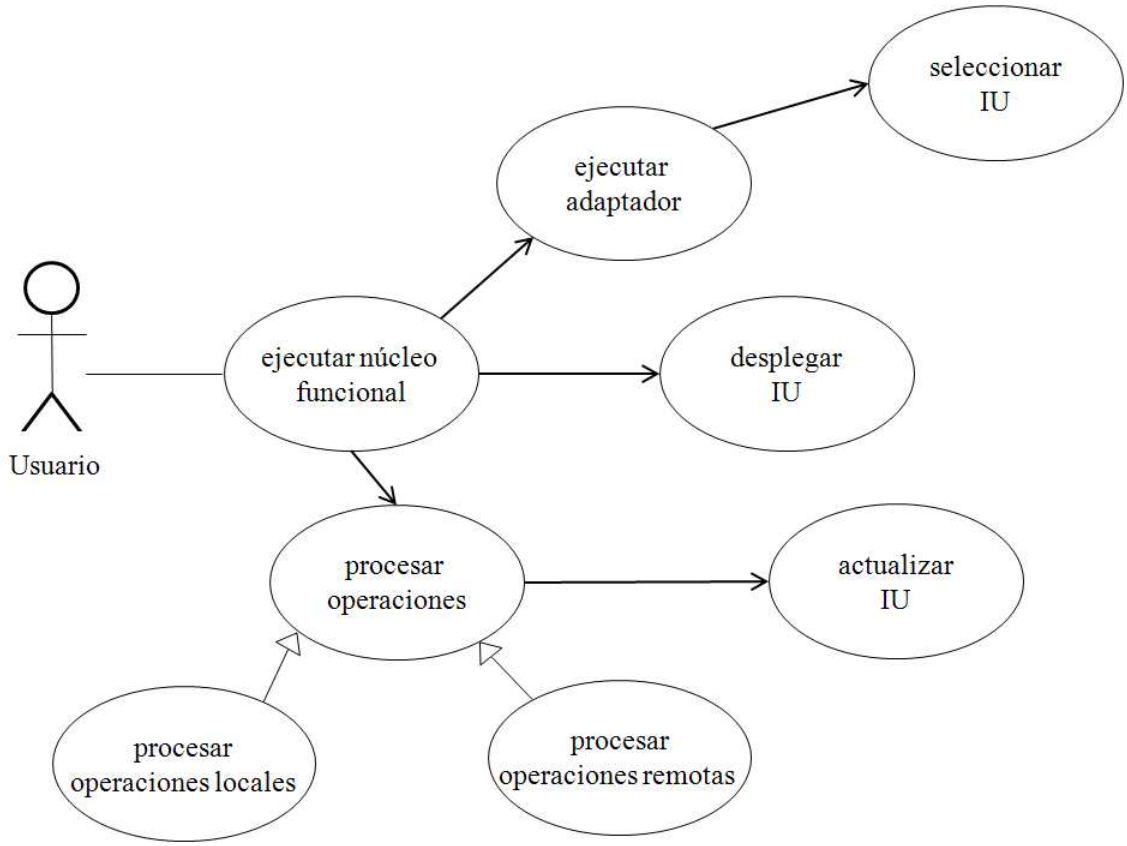


Figura 3.1: Casos de uso del mecanismo ReSAIC

3.2 Vistas estática y dinámica del mecanismo ReSAIC

Con el propósito de identificar a los principales componentes del mecanismo ReSAIC, en la presente sección se muestra en primer lugar la vista estática de dicho mecanismo mediante un diagrama de clases y en segundo lugar se expone la vista dinámica de mismo mecanismo por medio de un diagrama de colaboración.

Vista estática

Para facilitar la interacción entre un grupo de usuarios y una aplicación colaborativa, se identificaron las siguientes clases (cf. Figura 3.2):

1. Clase `Adaptador`: permite seleccionar la interfaz de usuario adecuada al tamaño de pantalla de un dispositivo de cómputo.

2. Clase `Interfaz de usuario`: es el medio visual adaptado al tamaño de pantalla del dispositivo de cómputo que permite llevar a cabo la interacción entre varios usuarios de una aplicación colaborativa.

3. Clase `Núcleo funcional`: es el componente principal de una aplicación colaborativa, cuya finalidad es procesar operaciones tanto locales (respecto al dispositivo de cómputo de un usuario) como remotas (operaciones que ha realizado otro usuario). Toda operación procesada en el núcleo funcional se reflejarán en la interfaz de usuario.

4. Clase `Intermediario`: proyecta la participación de un usuario sobre la aplicación colaborativa.

5. Clase `Coordinador`: organiza la interacción entre los usuarios de una aplicación colaborativa.

6. Clase `Vigía`: es un demonio que está a la espera de que se conecte un objeto de la clase `Intermediario`. Este demonio también crea un objeto de la clase `Coordinador`, el cual establece una comunicación directa con dicho objeto de la clase `Intermediario`.

Considere las siguientes definiciones: sea `aplicación colaborativa` un objeto de la clase `Aplicación Colaborativa`, `mecanismo ReSAIC` un objeto de la clase `Mecanismo ReSAIC`, `adaptador` un objeto de la clase `Adaptador`, `núcleo funcional` un objeto de la clase `Núcleo funcional`, `intermediario` un objeto de la clase `Intermediario`, `vigía` un objeto de la clase `Vigía` y `coordinador` un objeto de la clase `Coordinador`.

Entre las clases `Usuario` y `Aplicación colaborativa` existe una relación de agregación con cardinalidad de `n...1`, ya que un usuario puede hacer uso de una aplicación colaborativa. Por otro lado, una aplicación colaborativa puede ser empleada por varios usuarios.

Las clases `Aplicación colaborativa` y `Núcleo funcional` mantienen una relación de composición con cardinalidad de `1...n`, porque durante la ejecución de una aplicación colaborativa pueden ser creadas varias instancias de su núcleo funcional, una por cada usuario. Por otra parte, un núcleo funcional es exclusivo de una aplicación colaborativa.

Entre las clases `Aplicación colaborativa` e `Interfaz de usuario` se define una relación de composición con cardinalidad de `1...n`, debido que una aplicación colaborativa puede desplegar varias instancias de su interfaz de usuario (una por cada usuario). Por otro lado, una interfaz de usuario forma parte solamente de una aplicación colaborativa.

Las clases `Usuario` y `Núcleo funcional` tienen una relación de asociación con cardinalidad de `1...1`, ya que un usuario solo puede ejecutar, en su dispositivo de cómputo, una

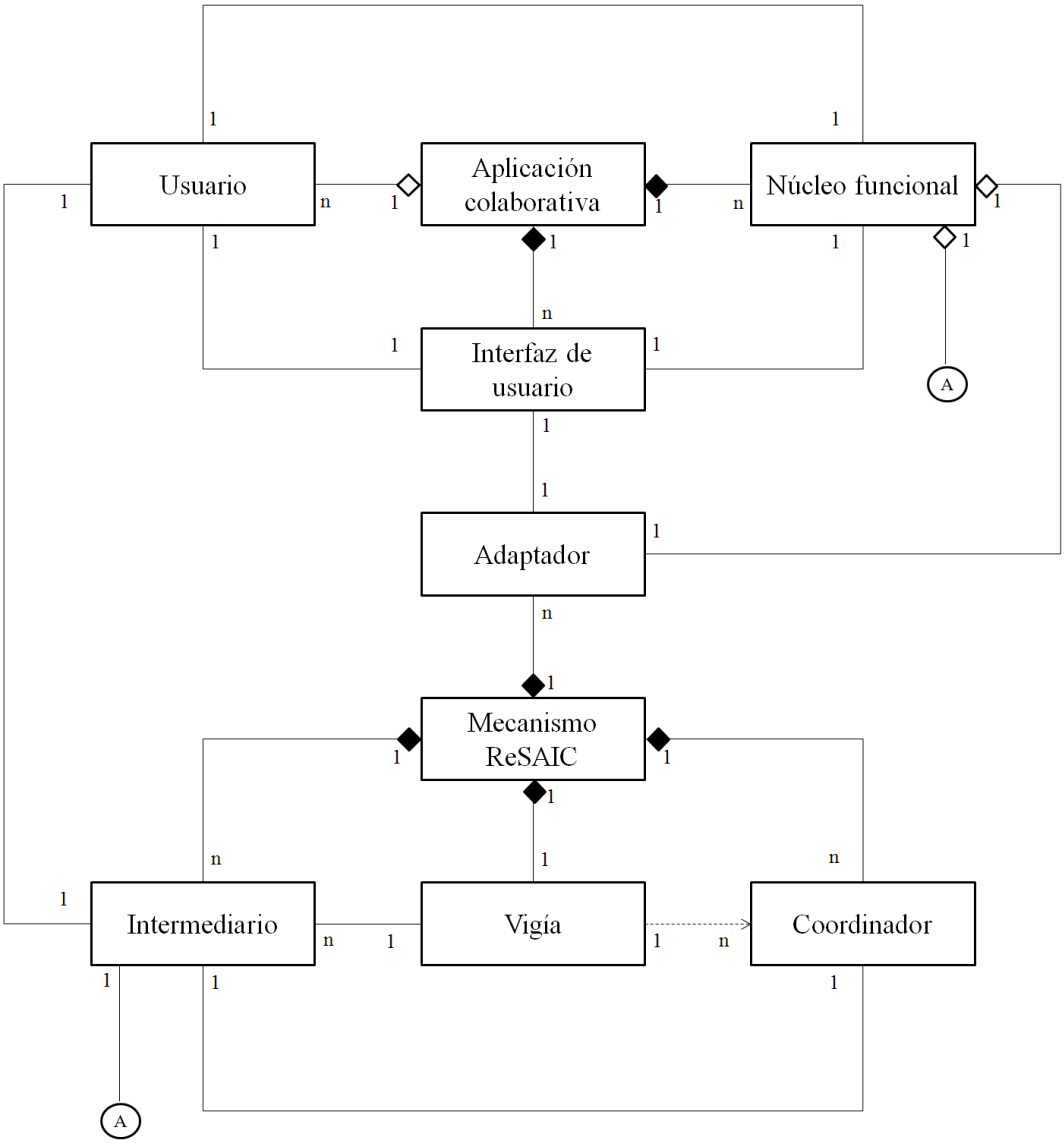


Figura 3.2: Diagrama de clases del mecanismo ReSAIC

instancia del núcleo funcional. Por otra parte, una instancia del núcleo funcional únicamente provee servicio a un usuario.

Entre las clases `Núcleo funcional` e `Interfaz de usuario` se define una relación de asociación con cardinalidad de `1...1`, porque una instancia del núcleo funcional despliega y actualiza solamente una instancia de la interfaz de usuario. Por otro lado, una interfaz de usuario es actualizada únicamente por un núcleo funcional.

Las clases `Usuario` e `Interfaz de usuario` mantienen una relación de asociación con cardinalidad de `1...1`, debido un usuario solo interactúa con una intancia de la interfaz de usuario. Por otra parte, una interfaz de usuario es exclusiva de un usuario.

Entre las clases `Núcleo funcional` y `Adaptador` existe una relación de agregación con cardinalidad de `1...1`, ya que una instancia del núcleo funcional utiliza un único adaptador para seleccionar una interfaz de usuario. Por otro lado, un adaptador solo puede ser

empleado por una instancia del núcleo funcional.

Entre las clases `Núcleo funcional` e `Intermediario` existe una relación de asociación con cardinalidad de `1...1`, porque una instancia del núcleo funcional solo utiliza un intermediario para obtener acciones remotas o enviar acciones locales. Por otro lado, un adaptador únicamente puede ser empleado por una instancia del núcleo funcional.

Las clases `Interfaz de usuario` y `Adaptador` tienen una relación de asociación con cardinalidad es de `1...1`, debido a que un adaptador selecciona, de un conjunto de interfaces de usuario preestablecidas, una sola interfaz de usuario adecuada al tamaño de pantalla de un dispositivo de cómputo. Por otro lado, una interfaz de usuario es elegida por un único adaptador.

Entre las clases `Adaptador` y `Mecanismo ReSAIC`, se establece una relación de composición con cardinalidad de `n...1`, ya que un adaptador forma parte del mecanismo ReSAIC. Por otro lado, el mecanismo ReSAIC puede contener varios adaptadores.

Las clases `Mecanismo ReSAIC` e `Intermediario`, tienen una relación de composición con cardinalidad de `1...n`, porque el mecanismo ReSAIC puede contener varios intermediarios. Por otro lado, un intermediario forma parte del mecanismo ReSAIC.

Entre las clases `Vigía` y `Mecanismo ReSAIC`, se encuentra una relación de composición con cardinalidad de `1...1`, debido a que un vigía forma parte del mecanismo ReSAIC. Por otro lado, el mecanismo ReSAIC contiene únicamente un vigía.

Las clases `Intermediario` y `Vigía` tienen una relación de asociación con cardinalidad de `n...1`, ya que un intermediario solo puede comunicarse con un vigía. Por otro lado, un vigía puede establecer conexión con varios intermediarios.

Entre clases `Mecanismo ReSAIC` y `Coordinador`, se define una relación de composición con cardinalidad de `1...n`, porque el mecanismo ReSAIC puede contener varios coordinadores. Por otro lado, un coordinador forma parte del mecanismo ReSAIC.

Las clases `Vigía` y `Coordinador` mantienen una relación de instanciación con cardinalidad de `1...n`, debido a que un vigía puede crear varias instancias del coordinador. Por otro lado, un coordinador es instanciado por un vigía.

Entre las clases `Intermediario` y `Coordinador` existe una relación de asociación con cardinalidad de `1...1`, ya que un intermediario interactúa con un solo coordinador. Por otra parte, un coordinador interactúa con un único intermediario.

Las clases `Intermediario` y `Usuario` tienen una relación de asociación con cardinalidad de `1...1`, porque un intermediario proyecta la participación de un solo usuario. Por otro lado, un usuario está representado por un único intermediario.

Vista dinámica

Para ilustrar la vista dinámica del mecanismo ReSAIC se presenta el diagrama 3.3, el cual muestra la colaboración entre cuatro usuarios que utilizan diferentes dispositivos de cómputo, e.g., *laptop*, PDA vertical, PC y PDA horizontal. Suponga que el usuario que emplea la *laptop* es el primero que ingresa su nombre en la `interfaz de usuario` de una aplicación colaborativa. En respuesta, el `núcleo funcional` envía el id de dicho usuario al `intermediario`, el cual solicita establecer una conexión con el `vigía`. Seguidamente, el `vigía` crear un hilo `coordinador` C1 que se encarga de atender a dicho `intermediario`.

Por cada usuario que ingresa a la aplicación colaborativa, e.g., PDA vertical, PC y PDA horizontal se lleva a cabo el proceso mencionado en el párrafo anterior. De esta forma, el

`vigía` asigna los hilos `coordinador` C2, C3 y C4 para atender respectivamente a dichos usuarios.

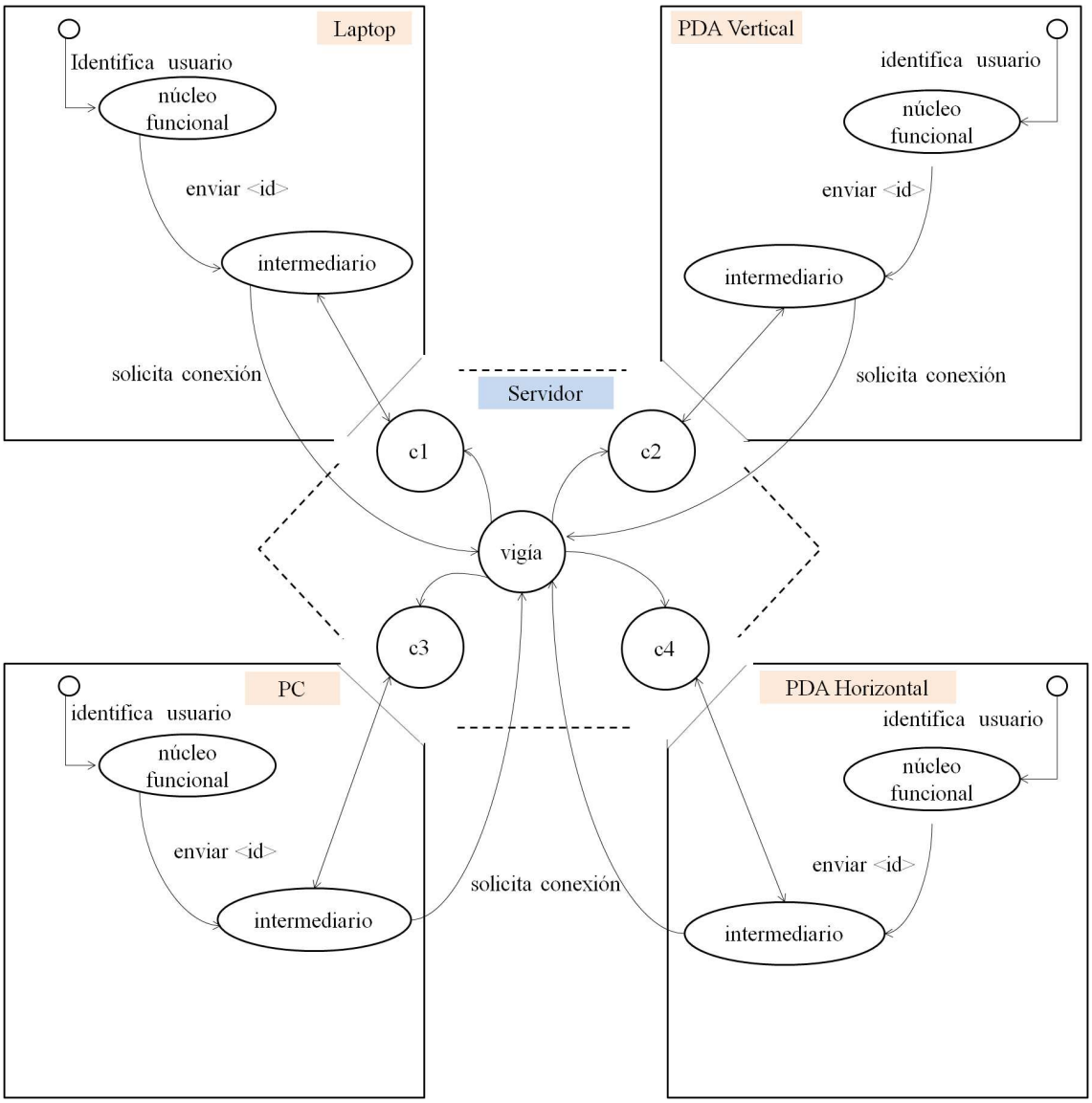


Figura 3.3: Diagrama de colaboración del mecanismo ReSAIC

Es necesario precisar que, aunque varios usuarios ingresen su nombre simultáneamente en la `interfaz de usuario` de una aplicación colaborativa, todos los `intermediarios` tendrán asignado un `coordinador`, ya que las solicitudes de conexión llegan a una cola de espera reservada por el `vigía`. Dicha cola es de tipo *FIFO*, i.e., el primero en entrar, es el primero en salir.

3.3 Componentes del mecanismo ReSAIC

Los componentes principales que integran el mecanismo ReSAIC siguen una arquitectura cliente/servidor. Si bién la arquitectura será analizada en la sección 3.4, la Figura 3.4 permite mostrar la distribución de los componentes según el modelo cliente-servidor:

1. el *software* del cliente (cf. Figura 3.4 #1) contiene las clases `Núcleo funcional` e `Intermediario`, la cual forma parte del componente de **soporte de colaboración**; así mismo contiene el componente de **adaptación plástica**, i.e., la clase `Adaptador` y un conjunto de interfaces de usuario de una aplicación colaborativa; este *software* se ejecuta en el dispositivo de cómputo de cada usuario que participa en la aplicación colaborativa en cuestión;

2. el *software* del servidor (cf. Figura 3.4 #2) contiene las clases `Vigía` y `Coordinador`; este *software* está hospedado en una PC que funge como servidor.

A continuación se describen los componentes principales del mecanismo ReSAIC: a) la adaptación plástica (cf. letra A de la Figura 3.4) y b) el soporte de colaboración (cf. letra B de la Figura 3.4). En esta descripción se justifican las principales características de cada uno de estos componentes.

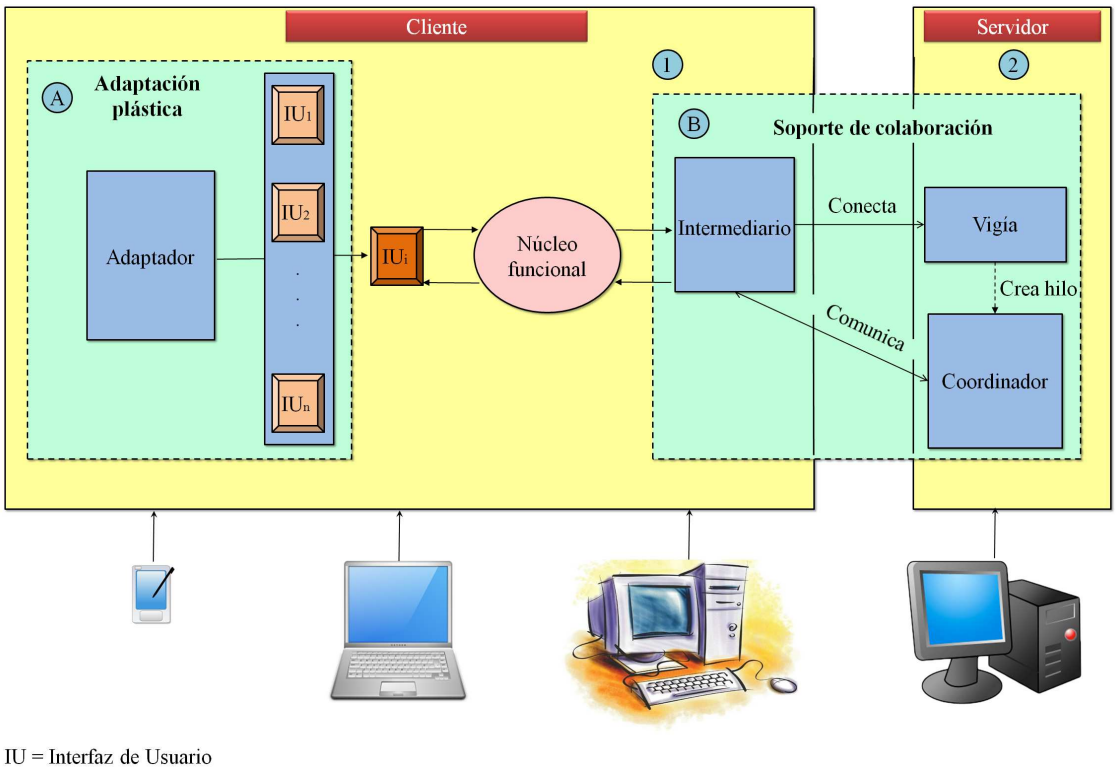


Figura 3.4: Diagrama de componentes del mecanismo ReSAIC

3.3.1 Adaptación plástica

Para definir la estructuración de la adaptación plástica, se parte del algoritmo de cinco pasos descrito en la sub-sección 2.1.2 del capítulo 2: 1) detección de las variaciones en el contexto de uso, 2) identificación de las soluciones candidatas, 3) selección de una solución particular, 4) transición entre estados y 5) ejecución de la nueva solución. Sin embargo, en la presente estructuración hemos omitido el cuarto paso de dicho algoritmo, denominado "transición entre estados", porque no consideramos el cambio de una solución a otra. Esta omisión implica que la interfaz de usuario seleccionada al inicio de una sesión colaborativa no se remodelará ni se redistribuirá ni tampoco migrará de un dispositivo de cómputo a otro durante dicha sesión.

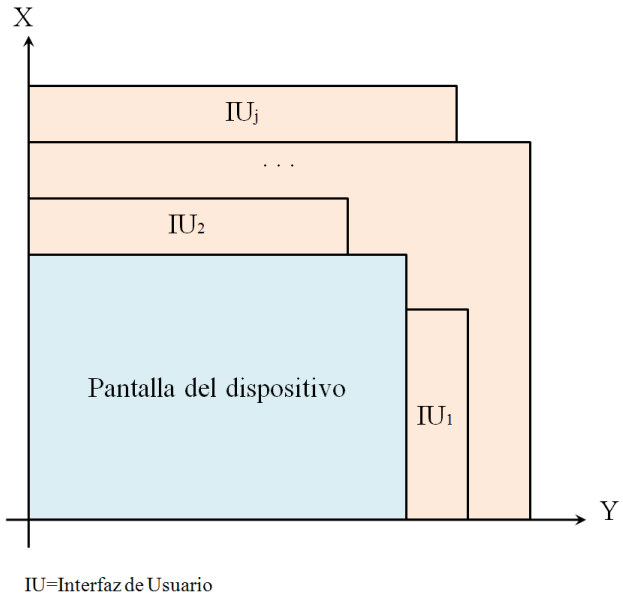


Figura 3.5: Interfaces de usuario inadecuadas para la pantalla de un dispositivo de cómputo

El `contexto de uso` está definido por la terna $< usuario, entorno, plataforma >$ (cf. sub-sección 1.1 del capítulo 1). En el presente trabajo únicamente consideramos la adaptación de la interfaz de usuario de una aplicación colaborativa a diversas plataformas. Sin embargo, es necesario precisar que al ejecutarse dicha aplicación, la interfaz de usuario no varía su presentación durante una sesión colaborativa. Por lo tanto, la adaptación plástica se realiza mediante un algoritmo que consta de los siguientes cuatro pasos:

1. **Detección del contexto de uso**: obtiene las dimensiones (ancho y alto en pixeles) de la pantalla del dispositivo de cómputo en el que se ejecutará una aplicación colaborativa.

2. **Identificación de las soluciones candidatas**: identifica las interfaces de usuario de una aplicación colaborativa que son apropiadas al nuevo contexto de uso. Es evidente que las interfaces de usuario cuyas dimensiones sean mayores que las de la pantalla de un dispositivo de cómputo resultan inadecuadas (e.g., IU_1, IU_2, ..., IU_j de la Figura 3.5). En consecuencia, se consideran soluciones candidatas aquellas interfaces de usuario que tienen un ancho y un alto menores o iguales que los de la pantalla del dispositivo de

cómputo. Es necesario verificar, mediante un algoritmo de búsqueda, que exista al menos una interfaz de usuario que tenga dichas características. En caso de que no exista dicha interfaz de usuario, se desplegará un mensaje de error en la pantalla del dispositivo de cómputo.

3. **Selección de una solución particular**: como varias soluciones candidatas pueden ser identificadas (e.g., IU_1, IU_2, ..., IU_i de la Figura 3.6), esta etapa selecciona la interfaz de usuario más adecuada a desplegar. De todas las interfaces de usuario candidatas, se elige la que minimiza la distancia d en Norma-2[1] entre el ancho x y el alto y de la pantalla del dispositivo de cómputo y el ancho x_i y el alto y_i de la interfaz de usuario:

$$d = min_{x,y}\sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{3.1}$$

   En caso de que existan dos o más interfaces de usuario que cumplen con esta regla, se mostrará la primera que se detecte.

4. **Ejecución de la nueva solución**: despliega la interfaz de usuario más adecuada al nuevo contexto de uso. El algoritmo de adaptación plástica retorna la identificación que corresponde a la interfaz de usuario seleccionada. Posteriormente, dicha identificación permite que un objeto encargado de la gestión de gráficos muestre la interfaz de usuario correspondiente.

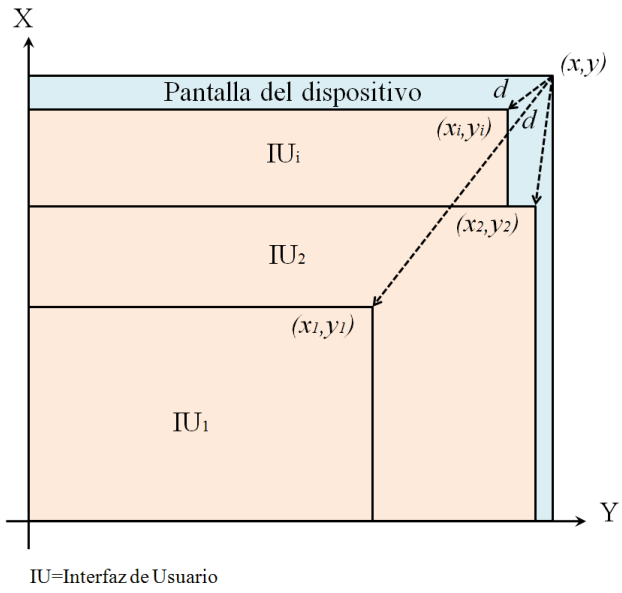


Figura 3.6: Interfaces de usuario candidatas para la pantalla de un dispositivo de cómputo

Para realizar el proceso de adaptación plástica se utiliza la Norma-2, la cual permite seleccionar de manera eficiente la interfaz de usuario que se desplegará en la pantalla de un dispositivo

[1]La Norma-2 o norma euclidiana es una norma para medir distancias, la cual considera que la distancia más corta entre dos puntos es la recta que los une.

de cómputo. Inicialmente se tenía contemplada otra opción, que consiste en calcular la superficie de las interfaces de usuario, con la finalidad de mostrar aquella que contara con un área menor o igual que la de la pantalla del dispositivo de cómputo. Sin embargo, el cálculo de áreas no es una solución adecuada en todos los casos. Por ejemplo, en la Figura 3.7 se muestra una interfaz de usuario que tiene un área de $54,000\ px^2$ y un dispositivo de cómputo con un área de $120,000\ px^2$. Evidentemente, la interfaz de usuario tiene un área menor que la del dispositivo de cómputo sin embargo, como se observa en esta figura, dicha interfaz no encaja en la pantalla de dicho dispositivo.

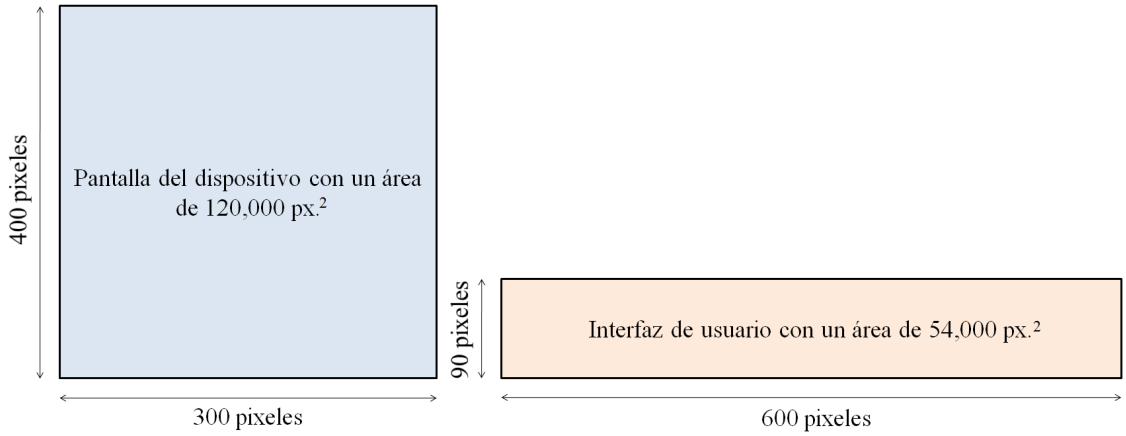


Figura 3.7: Interfaz de usuario con un área menor que la de la pantalla de un dispositivo

3.3.2 Soporte de colaboración

El soporte de colaboración tiene como principal objetivo permitir la interacción entre los usuarios de una aplicación colaborativa. Particularmente, este soporte está diseñado para aplicaciones colaborativas que tienen los siguientes requerimientos:

1. No aceptan el ingreso de nuevos usuarios, una vez iniciada una sesión colaborativa. Diseñamos un entorno básico de colaboración debido a que, en la presente tesis de maestría, los esfuerzos se concentran en el desarrollo de un mecanismo de adaptación.

2. Cada participante tiene asignado un turno para realizar la actividad propuesta por la aplicación colaborativa.

El soporte de colaboración está formado por las clases: i) `Intermediario`, ii) `Vigía` y iii) `Coordinador`. Los objetos de la clase `Intermediario` se comunican entre sí a través de los objetos de la clase `Coordinador`, con la finalidad de actualizar las interfaces de usuario de la aplicación colaborativa. De esta manera, los usuarios se mantienen informados de los acontecimientos más relevantes que ocurren en una sesión colaborativa.

Las funciones de un objeto de la clase `Intermediario` son:

1. Conectarse con un objeto de la clase `Vigía`;

2. Recibir la lista de todos los usuarios de la aplicación colaborativa;

3. Esperar la orden de un objeto de la clase `Coordinador` para informarle al usuario local que es su turno de participar en la aplicación colaborativa; esta funcionalidad permite satisfacer el segundo requerimiento enunciado al inicio de la presente sección;

4. Enviar los resultados de una acción realizada por el usuario local (e.g., mover una pieza en un tablero de ajedrez) a un objeto de la clase `Coordinador`; y

5. Recibir los datos de una acción realizada por el usuario remoto para actualizar la interfaz de usuario.

Las funciones de un objeto de la clase `Vigía` son:

1. Esperar a que un objeto de la clase `Intermediario` se conecte con él;

2. Impedir que se conecten más usuarios, representados por objetos de la clase `Intermediario`, una vez iniciada la aplicación colaborativa; esta funcionalidad permite satisfacer el primer requerimiento enunciado al principio de la presente sección;

3. Registrar cada objeto de la clase `Intermediario` que se conecta;

4. Asignar un turno a cada objeto de la clase `Intermediario` con base en la estrategia definida por la política "el que primero llega, es el primero que participa" (cf. sub-sección 2.1); esta funcionalidad permite satisfacer el segundo requerimiento; y

5. Crear un objeto de la clase `Coordinador` para cada objeto de la clase `Intermediario` que solicite una conexión.

Las funciones de un objeto de la clase `Coordinador` son:

1. Informar a todos los objetos de la clase `Intermediario` sobre quiénes participan en la aplicación colaborativa;

2. Notificar a cada objeto de la clase `Intermediario` su turno de participar en el momento que le corresponda tener el control de la aplicación;

3. Recibir de un objeto de la clase `Intermediario` los resultados de una acción realizada por un usuario (e.g., la nueva posición de una pieza en un tablero de ajedrez); y

4. Distribuir los resultados de una acción realizada por un usuario al resto de objetos de la clase `Coordinador`.

3.4 Arquitectura de comunicación del mecanismo ReSAIC

Los componentes de un sistema distribuido [Ardaiz et al., 2004] no solo están separados lógicamente sino también físicamente, por lo que requieren mecanismos de comunicación para poder interactuar. El software orientado a objetos de un sistema distribuido está diseñado de tal forma que todos los componentes, que requieran o proporcionen acceso a los recursos compartidos, estén implementados como objetos. Sin embargo, para que los objetos implicados en un misma tarea puedan interactuar, se necesita un soporte la transferencia de datos y la sincronización de operaciones.

La implementación de un sistema de paso de mensajes entre distintas computadoras requiere tanto una red de computadoras como sus respectivos protocolos de comunicación. El rendimiento global de un sistema distribuido tiene una dependencia directa de los mecanismos de comunicación utilizados para la interacción entre objetos remotos. El mecanismo más utilizado es el paso de mensajes, el cual recibe diversos nombres tales como canales, sockets o puertos. Para definir las reglas de comunicación entre objetos remotos, el desarrollador de aplicaciones puede recurrir a diversas arquitecturas, e.g., *peer to peer* y cliente-servidor.

El caso ideal para establecer una comunicación entre dos objetos remotos consiste en no depender de un servidor, ya que si este carece de un soporte adecuado para el manejo de fallos, pueden ocurrir graves consecuencias (e.g., pérdida de datos) cuando surja un problema (e.g., desconexión del servidor). Particularmente, en la arquitectura *peer to peer*, la comunicación entre nodos se lleva a cabo sin necesidad de utilizar un servidor dedicado. Sin embargo, es necesario crear un hilo[2] por cada dispositivo de cómputo que desee establecer una comunicación. La ejecución de múltiples hilos no representa ningún problema en un dispositivo de cómputo que cuenta con suficientes recursos tanto de memoria como de procesamiento. Sin embargo, en dispositivos con recursos restringidos (e.g., PDA) dicha ejecución puede ocasionar que la aplicación colaborativa resulte ineficiente para los usuarios.

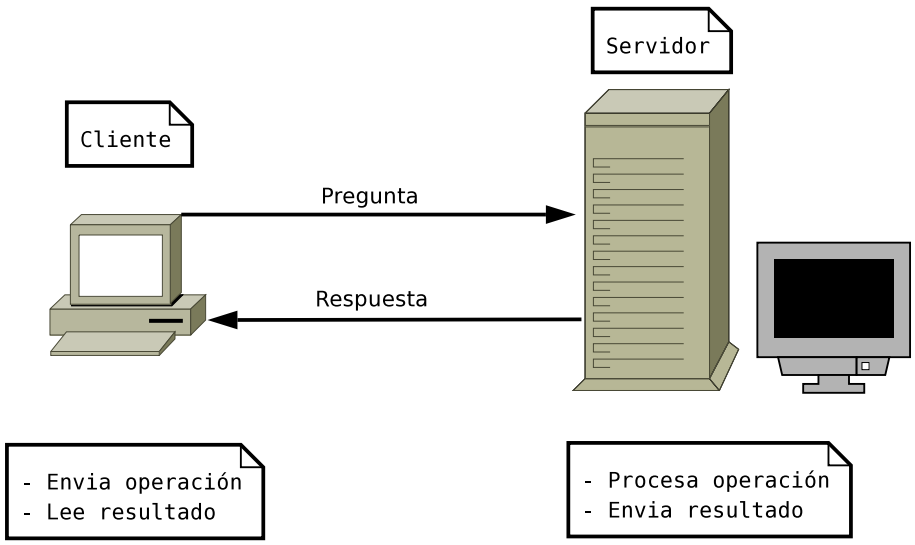


Figura 3.8: Modelo de comunicación cliente-servidor

[2]Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea, i.e., permite a una aplicación realizar varias tareas a la vez

En la presente tesis de maestría se emplean dispositivos de cómputo con recursos restringidos, e.g., PDA y PocketPC. Sin embargo, como se mencionó en el párrafo anterior, la arquitectura *peer to peer* es ineficiente cuando se utilizan dispositivos limitados. Con base en esta restricción, seleccionamos la arquitectura cliente-servidor.

Arquitectura cliente-servidor

La arquitectura cliente-servidor es un modelo de comunicación que permite a los usuarios finales obtener acceso a la información, en forma transparente, aún en entornos multiplataforma. Dicha arquitectura intenta ofrecer usabilidad, flexibilidad, interoperabilidad y escalabilidad. Su funcionamiento consiste en lo siguiente: el cliente envía un mensaje solicitando una determinada tarea a un servidor, el cual le envía uno o varios mensajes en respuesta (cf. Figura 3.8). En un sistema distribuido, cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras.

El cliente (conocido con el término de *front-end*) es el objeto que permite al usuario formular los requerimientos para enviarlos al servidor. El cliente maneja todas las funciones relacionadas con la manipulación y el despliegue de datos. Por lo tanto, el cliente está desarrollado sobre plataformas que permiten construir interfaces gráficas de usuario. Las funciones que lleva a cabo el cliente se resumen en los siguientes puntos: a) administrar la interfaz de usuario, b) interactuar con el usuario, c) procesar la lógica de la aplicación, d) generar operaciones, e) recibir resultados del servidor y f) formatear resultados.

El servidor (conocido con el término *back-end*) está encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. El servidor maneja todas las funciones relacionadas con la mayoría de las reglas de la aplicación. Las funciones que lleva a cabo el objeto servidor se resumen en los siguientes puntos: a) aceptar los requerimientos que hacen los clientes, b) procesar operaciones, c) formatear datos para trasmitirlos a los clientes y d) procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

En la presente investigación se utiliza una variante del modelo cliente-servidor, según la cual el servidor funge como coordinador de los clientes. Más precisamente, la creación de hilos para atender las peticiones de los objetos de la clase `Intermediario` es desempeñada por el único objeto de la clase `Vigía`, el cual está hospedado en una PC (cf. Figura 3.9). Dicho objeto queda a la espera de que se conecte un objeto de la clase `Intermediario`. Al conectarse un objeto de esta última clase, el objeto de la clase `Vigía` crea un hilo para establecer una comunicación directa con él. Acto seguido, el objeto de la clase `Vigía` continúa a la espera de solicitudes de conexión por parte de otros objetos de la clase `Intermediario`.

Medios de comunicación

Existen dos medios de comunicación de red: cableado e inalámbrico. En el presente trabajo de desarrollo se utiliza el medio inalámbrico, debido a que este se encuentra disponible para múltiples dispositivos móviles (e.g., PDA, *laptop*) sin requerir ningún gasto en infraestructura. Cualquier dispositivo móvil que tenga acceso a la red puede conectarse desde cualquier punto de acceso dentro de un rango suficientemente amplio de espacio. Sin embargo, dentro del medio inalámbrico existen varias tecnologías, e.g., Wi-Fi y Bluetooth.

La tecnología Wi-Fi [Kindberg et al., 2008] es la mejor opción para conectar una PC o un

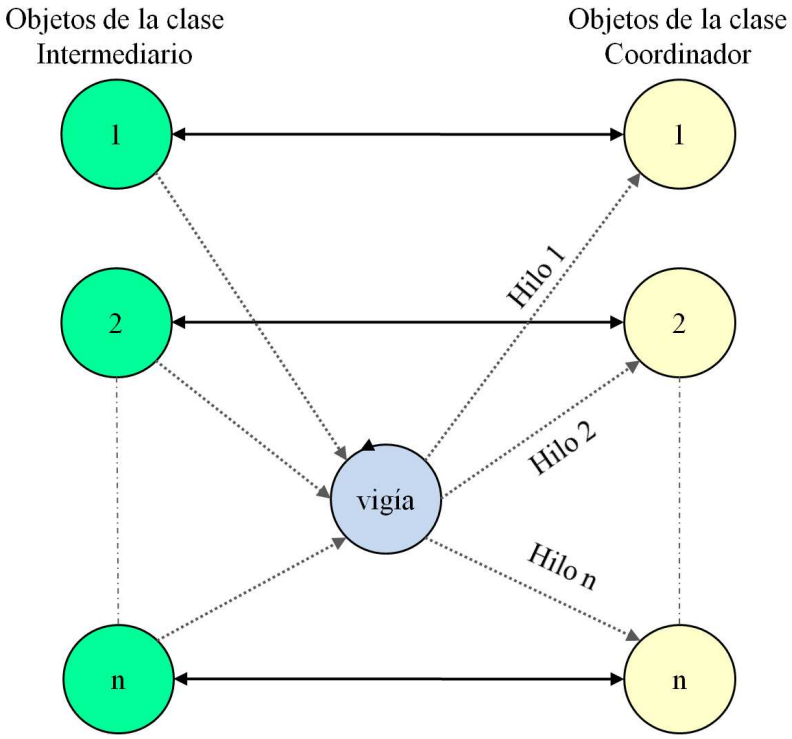


Figura 3.9: Servidor concurrente basado en múltiples hilos

PDA a una red LAN Ethernet porque cuenta con mayores velocidad de transmisión y cobertura de radio. Por otra parte, la tecnología Bluetooth ofrece menores velocidad de transmisión y ancho de banda para la transmisión de datos y está orientada a aplicaciones *peer to peer*. Por esta razón, la mejor opción de comunicación inalámbrica para este trabajo de desarrollo es la tecnología Wi-Fi.

3.5 Arquitectura de distribución del mecanismo ReSAIC

Una arquitectura de distribución define la repartición de los objetos de un sistema entre los sitios implicados en un proceso de interacción. Las arquitecturas de distribución difieren principalmente en tres aspectos: 1) la representación de cada objeto compartido en los sitios participantes; 2) el número de instancias de cada clase que están presentes en el sistema y 3) la posible movilidad de cada objeto entre dichos sitios.

Como se mencionó en la sección 3.2, el mecanismo ReSAIC consiste de un conjunto de clases, las cuales pueden ser reutilizadas para el desarrollo de aplicaciones con un nivel básico de interacción. Una parte de dicho mecanismo está localizado en una PC, la cual desempeña el papel de servidor. Sin embargo, la mayor parte del mecanismo ReSAIC se encuentra en un directorio, llamado *mecanismo*, el cual está hospedado en los diversos dispositivos de cómputo que utilizan los usuarios para interactuar en una sesión colaborativa. Dicho directorio contiene las siguientes clases:

1. Clase `Adaptador`: los métodos de esta clase tienen la finalidad de permitir el despliegue

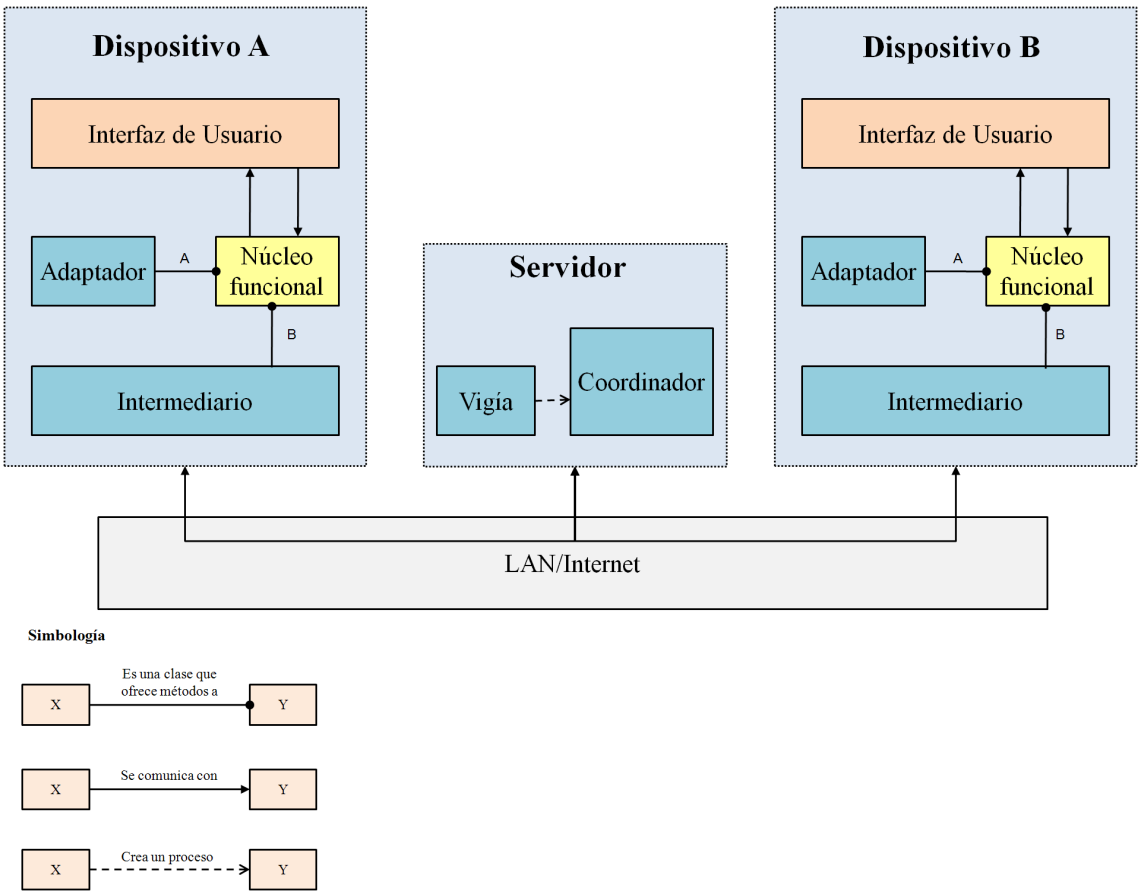


Figura 3.10: Arquitectura de distribución del mecanismo ReSAIC

de la interfaz de usuario adecuada a un determinado dispositivo de cómputo.

2. Clase `Intermediario`: los métodos de esta clase tienen el objetivo de proyectar la participación de un usuario en una aplicación colaborativa.

Métodos de la clase `Adaptador` (cf. letra A de la Figura 3.10):

- Método `detecta_dispositivo()`: obtiene las dimensiones, i.e., ancho y alto en pixeles, de la pantalla de un dispositivo de cómputo.
- Método `selecciona_interfaz(entero vector2D, entero x, entero y)`: devuelve el identificador (i.e., un número entero) de la interfaz de usuario que mejor se adapta a la pantalla de un dispositivo de cómputo. A este método es necesario pasarle como parámetros: 1) un arreglo de dos dimensiones en donde se incluyan los tamaños (i.e., ancho y alto en pixeles) de todas las interfaces de usuario disponibles y 2) las dimensiones de la pantalla del dispositivo de cómputo.

Métodos de la clase `Intermediario` (cf. letra B de la Figura 3.10):

- Método `conectar (cadena nodo)`: permite la conexión con el servidor; el parámetro nodo indica la dirección IP del sitio en el que se ejecuta dicho servidor.

- Método `pasar_idUsuario(cadena)`: se utiliza para que el núcleo funcional pase el identificador de un usuario a un objeto de la clase `Intermediario`. La estructura de dicho identificador se deja al criterio del programador de aplicaciones.

- Método `obtenerListaUsuarios()`: devuelve los identificadores de todos los usuarios presentes en una aplicación colaborativa.

- Método `obtenerDatos()`: obtiene los datos que envía un objeto de la clase `Coordinador`. Es necesario que el programador de aplicaciones verifique si estos datos corresponden a una acción realizada por un usuario remoto o bien a una indicación de turno de participación.

- Método `pasarAccion(cadena)`: permite pasar a un objeto de la clase `Intermediario` una cadena que contiene información sobre una acción realizada en la interfaz de usuario por el usuario local. La estructura de dicha cadena se deja al criterio del programador de aplicaciones.

Los métodos anteriormente descritos son invocados por el desarrollador del núcleo funcional de una aplicación colaborativa. De igual forma, se deja al desarrollador la tarea de diseñar e implementar múltiples versiones de la interfaz de usuario de dicha aplicación.

Interacción entre los componentes del mecanismo ReSAIC

Considere las siguientes definiciones: sea `tabla hash` un objeto de la clase `Tabla hash`, `dimension` un objeto de la clase `Dimension`, `adaptador` un objeto de la clase `Adaptador`, `intermediario` un objeto de la clase `Intermediario`, `vigía` un objeto de la clase `Vigía`, `coordinador` un objeto de la clase `Coordinador` y `núcleo funcional` un objeto de la clase `Núcleo funcional`.

En la Figura 3.11, se observa la secuencia de pasos para llevar a cabo la fase de adaptación plástica (cf. sub-sección 3.2.1 del capítulo 3):

1. un `usuario` ejecuta el `núcleo funcional` de una aplicación colaborativa en su dispositivo de cómputo;

2. el `núcleo funcional` invoca al método `detecta_dispositivo()` del `adaptador`;

3. en respuesta, el `adaptador` solicita un objeto `dimension`, mediante el cual se obtiene el ancho y el alto de la pantalla del dispositivo de cómputo;

4. el `adaptador` obtiene la `dimension`;

5. el `adaptador` retorna a la `dimension` al `núcleo funcional`;

6. el `núcleo funcional` invoca al método `selecciona_interfaz(vector2D,x, y)` del `adaptador`, el cual toma como parámetros los tamaños de las `interfaces de usuario` (vector2D) y el tamaño de la pantalla del dispositivo de cómputo (x e y);

7. el `adaptador` selecciona, con base en la `Norma-2`, la `interfaz de usuario` que mejor se adecúa al dispositivo de cómputo;

8. el `adaptador` retorna el identificador (i.e., un número entero) de dicha `interfaz de usuario` al `núcleo funcional`; y

9. finalmente, el `núcleo funcional` despliega la `interfaz de usuario` en la pantalla del dispositivo de cómputo.

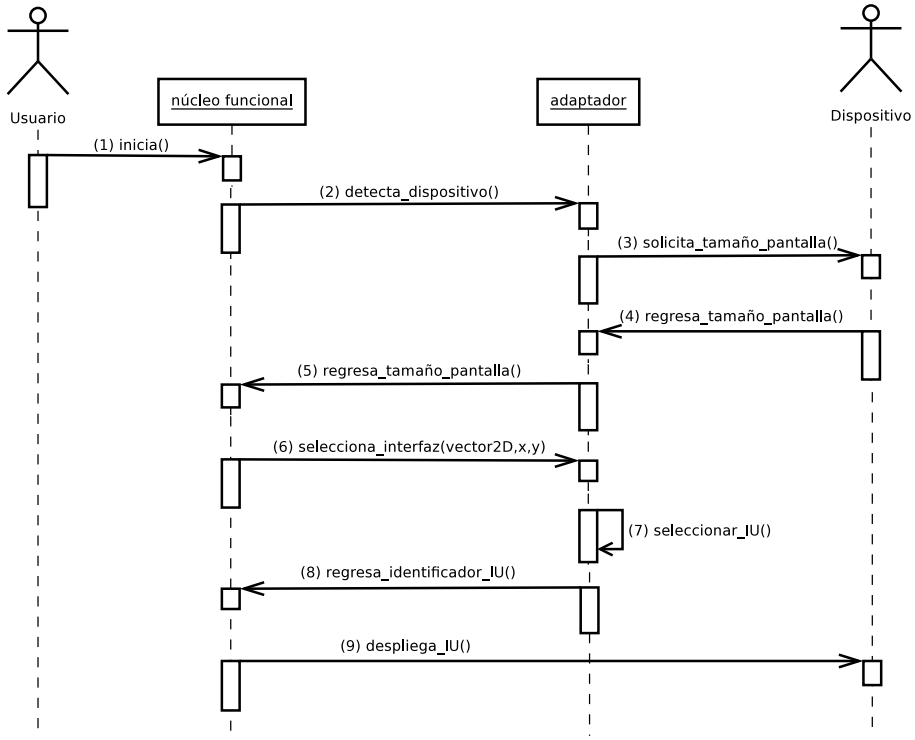


Figura 3.11: Diagrama de secuencias de la fase de adaptación plástica

Una vez que finalizó la fase de adaptación plástica, la siguiente fase consiste en permitir que los usuarios ingresen a la aplicación colaborativa. En la Figura 3.12 se muestra la secuencia de pasos para dar de alta a un usuario:

1. un `usuario` ingresa su nombre en la `interfaz de usuario` de la aplicación colaborativa;

2. la `interfaz de usuario` pasa el nombre del usuario al `núcleo funcional` al dar *click* sobre el botón “Entrar”;

3. el `núcleo funcional` genera un *id* para identificar al `usuario` (e.g., color y nombre del usuario);

4. el `núcleo funcional` invoca al método `conectar(nodo)` del `intermediario`, el cual recibe como parámetro la dirección IP del servidor;

5. en respuesta, el `adaptador` establece una conexión con el `vigía`;

6. el `núcleo funcional` invoca al método `pasar_idUsuario(cadena)` del `intermediario`, el cual recibe como parámetro el identificador del `usuario`;

7. el `intermediario` envía este identificador al `vigía` a través de un *socket*;

8. el `vigía` registra dicho identificador en una `tabla hash`; y

9. el `vigía` crea un `coordinador`, el cual establece una comunicación directa con el `intermediario`.

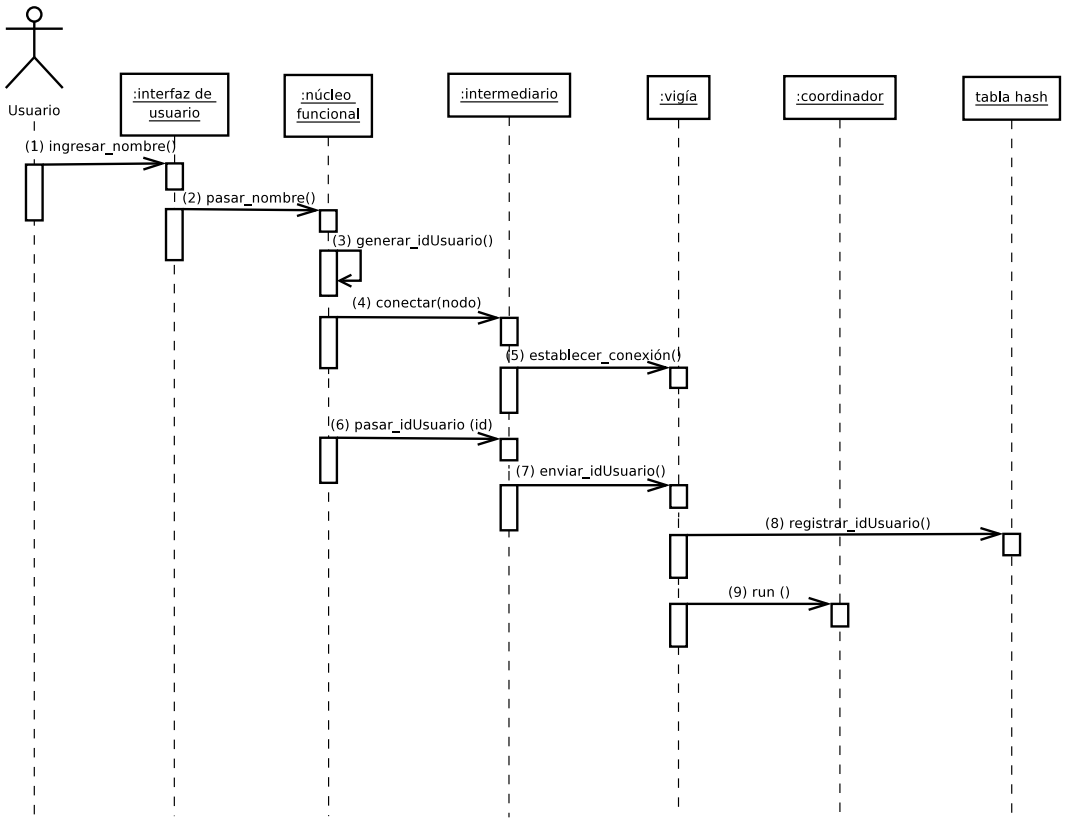


Figura 3.12: Diagrama de secuencias para dar de alta a un usuario

La clase `Coordinador` extiende la clase *Thread* con la finalidad de crear múltiples hilos. Los métodos de la clase `Coordinador` son:

- Constructor `Coordinador(`*`Socket`* `intermediario, entero id_hilo)`: inicializa las variables `in` y `out` que permiten llevar a cabo la comunicación con un `intermediario` e invoca al método *start()* de la clase *Thread*, el cual crea el hilo que ejecuta el código del método `run`.

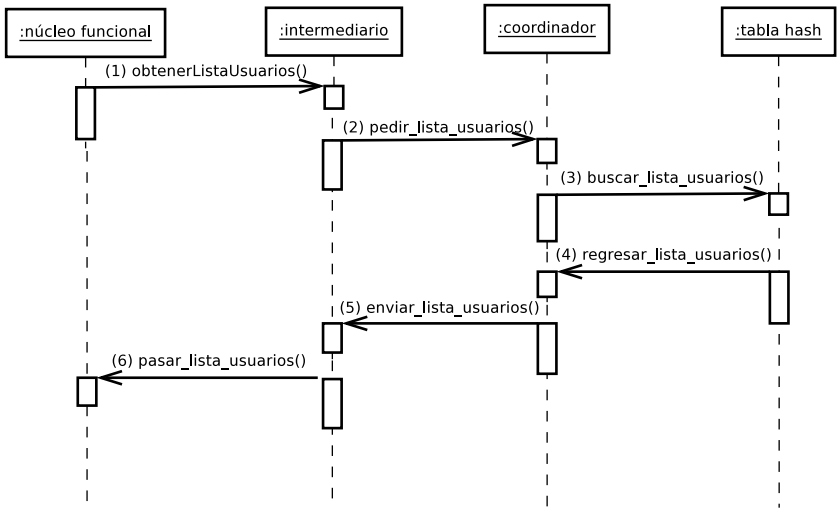


Figura 3.13: Diagrama de secuencias para obtener la lista de usuarios

- Método `run()`: permite coordinar las acciones de un usuario, i.e., le informa su turno de participación, así como las acciones realizadas por otros usuarios.

La Figura 3.13 muestra la secuencia de pasos que permiten obtener la lista de usuarios que participan en la aplicación colaborativa:

1. el `núcleo funcional` invoca al método `obtenerListaUsuarios()` del `intermediario`;
2. en respuesta, el `intermediario` espera recibir, de parte de un `coordinador`, la lista de participantes que ingresaron a la aplicación colaborativa;
3. el `coordinador` busca, en la `tabla hash`, los identificadores de los usuarios que han ingresado a la aplicación colaborativa;
4. el `coordinador` obtiene, de la `tabla hash`, la lista de usuarios;
5. el `coordinador` espera a que ingresen los usuarios faltantes y posteriormente envía la lista de usuarios (uno por uno) al `intermediario`; y
6. el `intermediario` retorna la lista de usuarios al `núcleo funcional`.

Al terminar la fase de ingreso de usuarios a la aplicación colaborativa, inicia el proceso de participación. La Figura 3.14 muestra la secuencia de pasos para que los usuarios interactúen con dicha aplicación:

1. el `núcleo funcional` invoca al método `obtenerDatos()` del `intermediario`;
2. en respuesta, el `intermediario` queda a la espera de recibir datos de un `coordinador` a través de la variable de comunicación `in`;

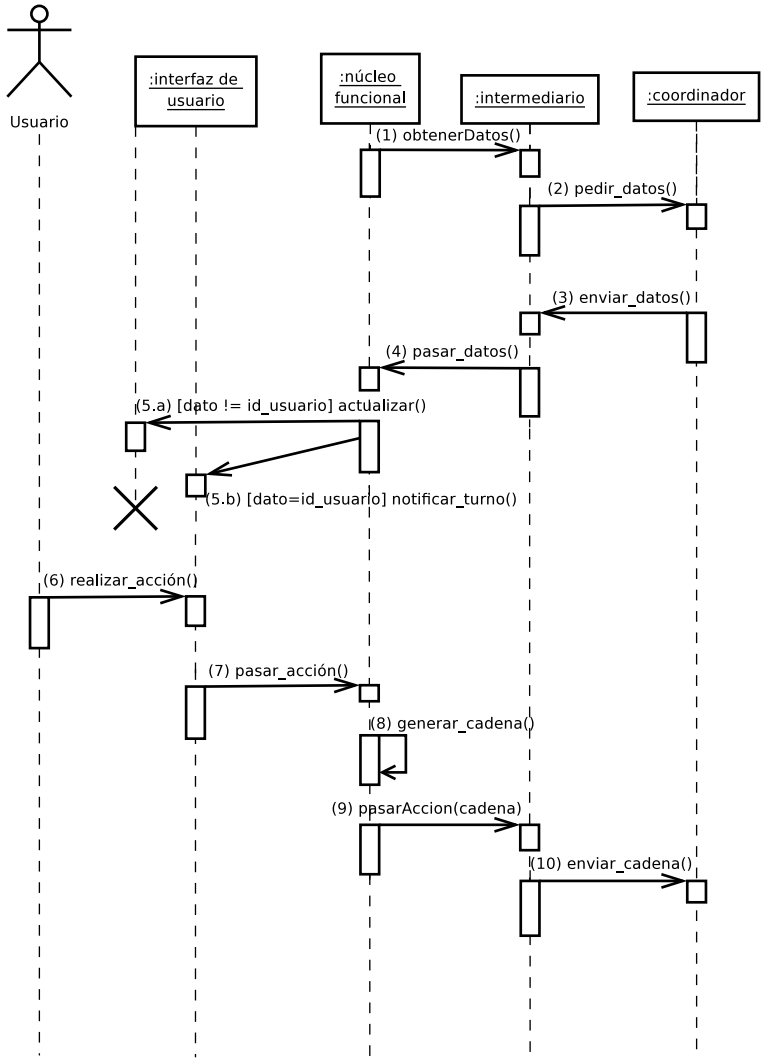


Figura 3.14: Diagrama de secuencias para la interacción entre los usuarios

3. el `coordinador` envía al `intermediario` un `dato`, el cual puede referirse al identificador del `usuario` (i.e., id usuario) o a una cadena que contiene información de una acción realizada por un usuario remoto (e.g., mover una pieza de ajedrez);

4. el `intermediario` retorna este `dato` al `núcleo funcional`;

5. el `núcleo funcional` compara dicho `dato`;

   (a) si `dato` es una cadena que contiene información de una acción realizada por un usuario remoto, el `núcleo funcional` muestra el efecto de dicha acción en la `interfaz de usuario` del usuario local;

   (b) si `dato` es el identificador del `usuario` significa que es su turno de participar, por lo tanto el `núcleo funcional` despliega un aviso en su `interfaz de usuario`;

6. si se cumple el punto 5.b, el `usuario` realiza una acción en la `interfaz de usuario`;

7. la `interfaz de usuario` pasa dicha acción al `núcleo funcional`;

8. el `núcleo funcional` genera una cadena que contiene los datos necesarios para informar a los usuarios remotos de la acción realizada por el `usuario` local;

9. el `núcleo funcional` invoca al método `pasarAccion(cadena)`, el cual recibe como parámetro la cadena generada en el paso anterior; y

10. en respuesta, el `intermediario` envía esta cadena al `coordinador` correspondiente, el cual la distribuye al resto de los `coordinadores` para que estos la difundan a los `intermediarios`.

Para demostrar la correcta interacción entre los objetos del mecanismo ReSAIC, se propone emplear el núcleo funcional de la aplicación "Serpientes y Escaleras". Se eligió este juego didáctico como aplicación de validación porque cuenta con las características propuestas en el soporte de colaboración del mecanismo ReSAIC.

3.6 Análisis y diseño de la aplicación "Serpientes y Escaleras"

La presente sección está estructurada en tres partes. En primer lugar, se describe la lógica del juego "Serpientes y Escaleras" y se justifica la elección de este juego colaborativo como aplicación de validación del mecanismo ReSAIC (cf. sub-sección 3.3.1). Enseguida, se describen las principales clases que definen las políticas utilizadas para llevar a cabo la coordinación entre los usuarios (cf. sub-sección 3.3.2). Finalmente, se muestran las versiones de la interfaz de usuario de la aplicación "Serpientes y Escaleras" para diversos dispositivos de cómputo (e.g., laptop y PDA en posiciones vertical y horizontal, cf. sub-sección 3.3.3).

3.6.1 Descripción del juego Serpientes y Escaleras

También conocido como "Juego de la Escalera", es esencialmente un entretenimiento para la enseñanza de valores, que representa simbólicamente el camino a lo largo de la vida para alcanzar el éxito por un buen comportamiento. El juego clásico consta de un tablero que tiene 100 casillas. Algunas casillas representan valores éticos mediante una escalera, la cual permite subir a otras casillas para acortar el camino a la meta. Sin embargo, también existen casillas que representan peligros por medio de una serpiente, la cual obliga a descender en el tablero.

Este viaje virtuoso está regido no por la voluntad ni el esfuerzo, sino por el azar ya que se juega con dados que, en última instancia, determinan dónde y cómo circular por el camino. El azar resulta paradójico si se trata de inculcar el esfuerzo por seguir las enseñanzas de una vida recta y moral. Sin embargo, el objetivo del juego es mostrar de forma simulada lo que sucedería en caso de no actuar en forma correcta. En este sentido, el aprendizaje consiste en sobreponerse a las caídas, en seguir jugando para reparar el daño y en continuar hasta alcanzar la meta[3].

[3] http://sepiensa.org.mx/contenidos/2006/p_serpientesyescaleras/serpientescaleras1.html

Por turno, cada jugador lanza un dado y avanza tantas casillas en el tablero como indique dicho dado. Si el jugador llega a una casilla que muestra la cola de una serpiente, entonces deberá regresar a la casilla en donde se encuentra la cabeza de esta serpiente. Por el contrario, si el jugador llega a una casilla que muestra la base de una escalera, entonces deberá avanzar a la casilla en donde termina esta escalera. El jugador que llegue primero a la casilla del tablero que contiene el número máximo, ganará el juego. Los demás jugadores pueden continuar jugando para clasificarse en segundo y tercer lugar. En caso de que un jugador se encuentre a pocas casillas de la meta, pero al lanzar los dados obtenga un número mayor del que necesita para ganar, entonces deberá retroceder tantas casillas como la cantidad que obtuvo de más.

¿Por qué un juego didáctico?

Desde punto de vista de la formación educativa, un juego didáctico es un medio a través del cual los participantes adquieren un cierto conocimiento a partir de sus interacciones con el mismo. Un juego didáctico combina el factor necesario de motivación para captar la atención de los participantes, con los objetivos del aprendizaje que subyacen en las actividades realizadas en el juego.

La incorporación de contenidos didácticos en juegos por computadora ha permitido evolucionar los recursos con los que cuentan los docentes para realizar su profesión. Los juegos por computadora, dentro de un aula escolar, se han convertido en un medio de enseñanza atractivo para los alumnos [González et al., 2007]. Existen numerosos estudios que abogan por los beneficios de los juegos por computadora como excelentes herramientas educativas. A partir de estos estudios, se pueden extraer las siguientes conclusiones:

- **éxito escolar:** los alumnos que utilizan juegos por computadora incrementan notablemente su capacidad de compresión lectora;
- **habilidades cognitivas:** los juegos por computadora proponen ambientes de aprendizaje basados en el descubrimiento y en la creatividad;
- **atención y concentración:** debido a su naturaleza lúdica, los juegos por computadora incrementan la concentración de los alumnos para resolver un problema concreto; y
- **motivación:** los juegos por computadora suponen un mecanismo de estimulo para los participantes, que facilita el proceso de aprendizaje y aumenta considerablemente la asistencia a clase.

En resumen, un juego didáctico por computadora propone un medio donde el aprendizaje se obtiene como resultado de las actividades realizadas en el juego. De esta manera, el conocimiento se adquiere a través del contenido del juego, en tanto que las habilidades cognitivas se desarrollan como resultado de la propia acción de jugar.

3.6.2 Mecanismo de coordinación

En la presente tesis de maestría se optó por utilizar las políticas de manejo de sesión denominadas “protocolos sociales” (cf. sub-sección 3.1) y “el primero que llega, es el que primero participa”, con el fin de facilitar la coordinación entre usuarios de igual jerarquía. Estas políticas no

bloquean las acciones de un usuario cuando no es su turno de participación, ya que el respeto de turno es de índole ético. Particularmente, estas políticas permiten definir la estructura del grupo, ya que:

- **establecen el control de una sesión**: determinan quién autoriza el registro de un usuario, cómo se realiza la interacción entre los usuarios y cómo se define el turno de participación de cada usuario y;
- **soportan cambios en tiempo de ejecución**: determinan cómo se modifica el rol de un usuario, los permisos del rol y/o la política actual.

En la Figura 3.15 se muestra el diagrama de clases del mecanismo de coordinación empleado en la aplicación "Serpientes y Escaleras". La clase `Estructura del grupo` está definida en un momento dado por una política específica, la cual establece cómo se organiza el trabajo colaborativo. Particularmente, en la aplicación "Serpientes y Escaleras", la estructura del grupo es "desestratificada", ya que todos los jugadores tienen los mismos derechos/obligaciones.

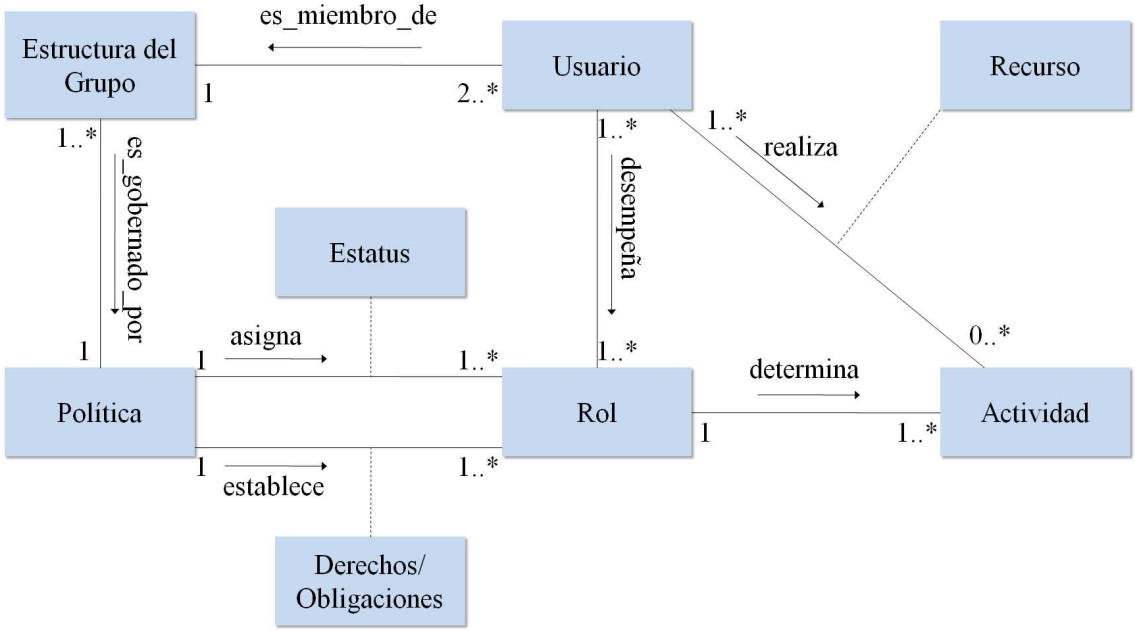


Figura 3.15: Diagrama de clases del mecanismo de coordinación

La clase `Política` define un conjunto de `Roles`, a cada uno de los cuales le asigna tanto un estatus como derechos/obligaciones. Para que la política funcione correctamente, cada rol siempre debe ser desempeñado por al menos un usuario. En específico, la política que gobierna el turno de participación de los usuarios de la aplicación "Serpientes y Escaleras" es "el primero que llega, es el primero que participa". De los n usuarios presentes en una sesión colaborativa solo un usuario a la vez puede tomar el rol de "jugador protagonista", quien tiene el control del juego, mientras que los $n-1$ jugadores toman el rol de "jugador espectador".

La clase `Estatus` define la línea de autoridad de un rol, de acuerdo a su posición dentro de la organización del grupo, con el fin de establecer de manera clara una estructura jerárquica.

El estatus también determina, en un instante dado, quién es el usuario que controla o dirige una sesión colaborativa. Particularmente, la interfaz de usuario de la aplicación "Serpientes y Escaleras" muestra el estatus mediante un semáforo, el cual indica a cada usuario su turno de participación en el juego. Con base en la política de "protocolos sociales", la aplicación no impide sin embargo que un usuario realice una acción cuando no le corresponde.

La clase `Derechos/Obligaciones` indica los permisos y las responsabilidades de un usuario de acuerdo al estatus del rol que desempeña en un momento dado. Estos derechos/obligaciones se definen con base en el estatus, mas no con base en el usuario que los tiene asignados. Por tanto, la política no debe modificarse cuando los usuarios cambian de estatus dentro de la organización del grupo. Los derechos/obligaciones restringen las acciones de un usuario cuando interactúa con los demás usuarios, a fin de desempeñar las actividades colaborativas. De esta manera, un derecho de los usuarios de la aplicación "Serpientes y Escaleras" consiste en tener asignado un turno de participación en el juego, mientras que una obligación se basa en respetar el turno del usuario en curso.

La clase `Usuario` es una persona que desempeña uno o varios roles que le permiten llevar a cabo las actividades colaborativas de acuerdo con sus derechos/obligaciones y su estatus. Especificamente, el arquetipo de los usuarios de la aplicación "Serpientes y Escaleras" está representado por niños de educación primaria.

La clase `Actividad` está constituida por un conjunto de acciones desempeñadas por uno o más usuarios con el fin de lograr una meta común. Para llevar a cabo dicha actividad se utilizan los recursos necesarios. Así, las actividades de los usuarios de la aplicación "Serpientes y Escaleras" consisten en lanzar un dado y posicionarse en la casilla correspondiente del tablero del juego.

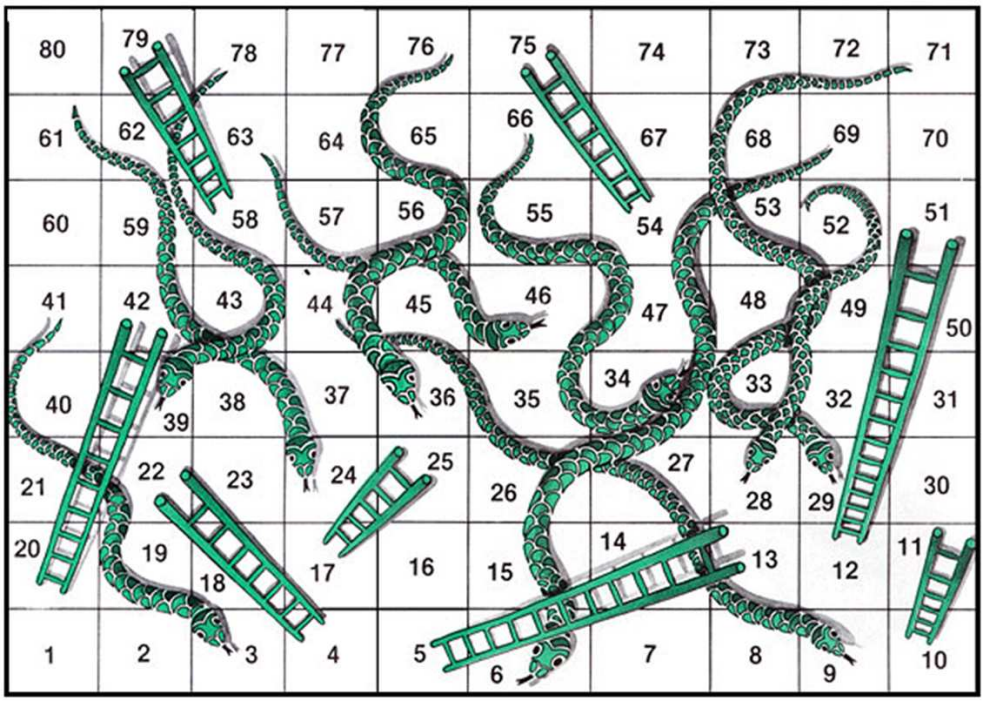


Figura 3.16: Tablero del juego "Serpientes y Escaleras"

Finalmente, la clase `Recurso` representa los objetos utilizados por los usuarios en la realización de sus actividades, con la finalidad de alcanzar los objetivos comunes establecidos en la estructura del grupo. En particular, los recursos de los que disponen los usuarios de la aplicación "Serpientes y Escaleras" son un dado y un tablero.

3.6.3 Interfaces de usuario de la aplicación "Serpientes y Escaleras"

Se diseñaron tres interfaces de usuario de la aplicación "Serpientes y Escaleras". La primera versión está diseñada para todo tipo de dispositivos con tamaño de pantalla de una PC o una laptop. La segunda versión está diseñada para PDAs que tienen un ancho menor o igual que 240 pixeles y un alto menor o igual que 320 pixeles. La tercera versión está diseñada para PDAs con las dimensiones anteriores (i.e., 240x320 pixeles), pero que pueden desplegar interfaces de usuario en posición horizontal. Con la finalidad de permitir la interacción entre los usuarios y la aplicación colaborativa desde dispositivos de cómputo con pantallas pequeñas se considera un tablero con ochenta casillas (cf. Figura 3.16).

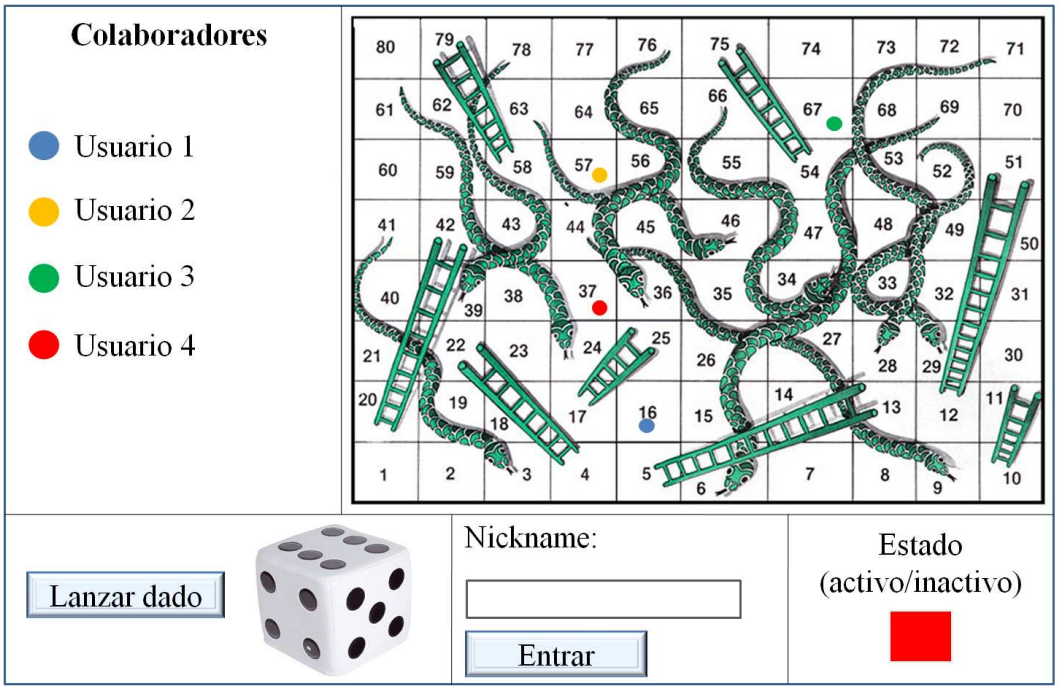


Figura 3.17: Interfaz de usuario de la aplicación "Serpientes y Escaleras" en una *laptop*

La Figura 3.17 muestra la interfaz de usuario de la aplicación "Serpientes y Escaleras" desplegada en una *laptop*. Esta interfaz de usuario está dividida en cinco partes:

1. La parte superior izquierda muestra los nombres de los usuarios presentes en la aplicación colaborativa. A cada usuario se le asigna un color con la finalidad de identificarlo.

2. La parte superior derecha muestra el tablero del juego, en el cual los usuarios deben posicionarse de acuerdo al número que obtienen al lanzar el dado. La ubicación de cada usuario en la casilla correspondiente se muestra mediante un circulo de color. Si un

usuario realiza un movimiento incorrecto en el tablero, la aplicación colaborativa no le permitirá avanzar.

3. La parte inferior izquierda está compuesta de un dado y un botón que permite simular el lanzamiento del dado.

4. La parte inferior central permite a los usuarios ingresar al juego mediante su respectiva contraseña.

5. La parte inferior derecha simula un semáforo, cuya función es notificar a los usuarios su turno de participación en el juego.

La Figura 3.18 muestra la interfaz de usuario de la aplicación "Serpientes y Escaleras" desplegada en una PDA (posición vertical). Esta interfaz de usuario presenta la misma información que la interfaz de usuario que se muestra en la *laptop* (cf. Figura 3.17). Sin embargo, a causa del tamaño de pantalla de la PDA, no toda la información puede ser desplegada simultáneamente. Para acceder a la información en la PDA, se utilizan pestañas colocadas en la parte superior de la interfaz de usuario, donde el jugador podrá elegir las opciones "Tablero" o "Colaboradores". Cuando el usuario selecciona la pestaña "Tablero" (cf. Figura 3.18.a), se muestra la parte de la interfaz de usuario que le permite lanzar los dados y avanzar en el tablero del juego. Cuando el usuario selecciona la pestaña "Colaboradores" (cf. Figura 3.18.b), se muestra la parte de la interfaz de usuario que le notifica al usuario su turno de participación en el juego.

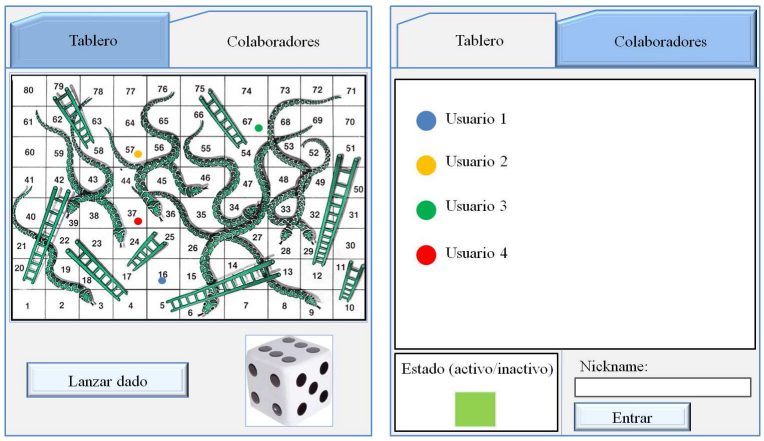


Figura 3.18: Interfaz de usuario de la aplicación "Serpientes y Escaleras" en una PDA (posición vertical)

Actualmente algunos dispositivos móviles con pantallas pequeñas tienen la capacidad de girar vertical a horizontalmente la interfaz de usuario de sus aplicaciones. La Figura 3.19 muestra la interfaz de usuario de la aplicación "Serpientes y Escaleras" ejecutada en una PDA en posición horizontal. Se observa que esta interfaz de usuario es similar a la que se muestra en la Figura 3.18. Sin embargo, algunos componentes gráficos cambiaron de posición, con la finalidad de aprovechar la orientación de pantalla de la PDA.

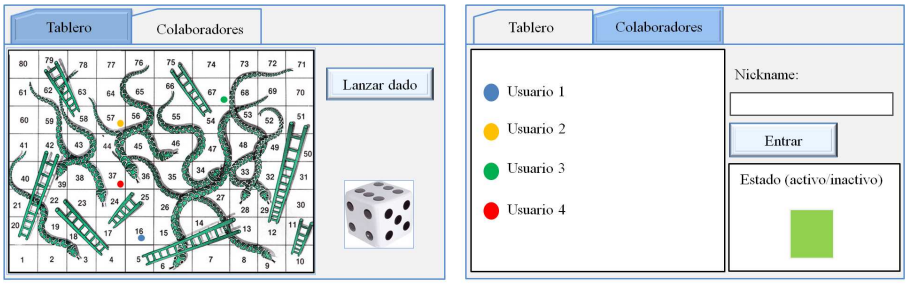


Figura 3.19: Interfaz de usuario de la aplicación “Serpientes y Escaleras” en una PDA (posición horizontal)

Capítulo 4

Implementación del mecanismo ReSAIC

El presente capítulo está estructurado en cinco partes. En primer lugar, se explican las razones por las cuales se eligió el lenguaje Java para implementar tanto el mecanismo ReSAIC como la aplicación de validación "Serpientes y Escaleras" (cf. sección 4.1). A continuación, se presentan los algoritmos que describen el funcionamiento del mecanismo ReSAIC (cf. sección 4.2). Enseguida, se explican tanto las principales tareas que realiza el núcleo funcional de esta aplicación de validación como los pasos del proceso de implementación de sus interfaces de usuario en una PC y una PDA (en el caso de este último dispositivo se consideran las vistas horizontal y vertical) (cf. secciones 4.3 y 4.4). Finalmente, se demuestra la funcionalidad del mecanismo ReSAIC con base en las pruebas realizadas a través de la aplicación de validación "Serpientes y Escaleras" (cf. sección 4.5).

4.1 Selección del lenguaje de programación

Una de las tareas fundamentales en la implementación de una aplicación es definir el lenguaje de programación más conveniente para desarrollarla. Particularmente, se busca un ambiente de desarrollo compatible con los dispositivos de cómputo utilizados. El *software* propuesto en la presente tesis de maestría se implementó con base en el paradigma de programación orientada a objetos [Cook, 1990], ya que los programas resultantes, a menudo, son más fáciles de entender, corregir y modificar. Es común que al referirse a la programación orientada a objetos se piense en Java, porque es el lenguaje de programación orientado a objetos que más se utiliza en el mundo. El lenguaje Java se ha convertido en el lenguaje de elección para implementar aplicaciones basadas en Internet y *software* para dispositivos que se comunican a través de una red. En la actualidad, Java se utiliza para: a) desarrollar aplicaciones empresariales a gran escala, b) mejorar la funcionalidad de los servidores Web y c) proporcionar aplicaciones para dispositivos domésticos (e.g., teléfonos celulares, radiolocalizadores y asistentes digitales personales).

Una de las características principales de Java es la independencia de la plataforma, i.e., los programas escritos en este lenguaje pueden ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "*write once, run everywhere*". Las primeras implementaciones de Java utilizaban una máquina virtual[1] interpretada para conseguir portabilidad. Sin embargo, los resultados

[1]La máquina virtual es un software que emula a una computadora y puede ejecutar programas como si fuese una computadora real

eran programas que se ejecutaban más lentamente que aquellos escritos en C o C++. Estos resultados hicieron que Java se ganara la reputación de lento en rendimiento. Empleando diversas técnicas, las implementaciones recientes de la máquina virtual de Java ejecutan programas considerablemente más rápido que las versiones antiguas. Sin embargo, dichas implementaciones de la máquina virtual siguen siendo mucho más lentas que otros lenguajes.

Un argumento en contra de lenguajes como C++ se refiere a que los programadores tienen la carga añadida de administrar la memoria dinámica de forma manual. En C++, el desarrollador puede asignar memoria en una zona conocida como *heap* (montículo) para crear cualquier objeto y posteriormente desalojar el espacio asignado cuando desee borrarlo. Un olvido de liberar la memoria previamente solicitada puede llevar a una fuga de memoria, ya que un programa mal diseñado podría consumir una cantidad desproporcionada.

En Java, el problema de administrar la memoria de forma manual es evitado en gran medida por el recolector automático de basura (*automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java *runtime*) es responsable de gestionar el ciclo de vida de dichos objetos. Un objeto puede estar referenciado múltiples veces en un programa. Cuando no quedan referencias a dicho objeto, el recolector de basura de Java borra el objeto para liberar la memoria que ocupaba. En definitiva, el recolector de basura de Java ofrece mayor seguridad que otros lenguajes en la administración de la memoria y permite una fácil creación y eliminación de objetos.

La recolección de basura en Java es un proceso prácticamente invisible al desarrollador, i.e., él no tiene conciencia del momento en que tendrá lugar dicho proceso, ya que este no necesariamente guarda relación con las acciones que realiza el código fuente. Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados.

Las características que ofrece Java se ajustan a los requerimientos de implementación del *software* que sustenta la presente tesis de maestría. Cabe señalar que existe una variedad de tecnologías Java. Por lo general, se utiliza la tecnología **Java 2 Platform Standard Edition (J2SE)**, la cual es básica para muchos estilos de desarrollo de *software*, incluyendo *applets*, clientes y servidores. Java es un lenguaje capaz de superar las diferencias que existen entre computadoras heterogéneas, al ser independiente del sistema operativo.

Entre la variedad de tecnologías Java, se encuentra la plataforma **Java 2 Micro Edition (J2ME)** que provee una colección certificada de APIs de desarrollo de *software* para dispositivos con recursos restringidos (e.g., PDAs, teléfonos móviles o electrodomésticos). La plataforma J2ME es por lo tanto una versión reducida de J2SE.

De toda la tecnología Java, hasta hace poco tiempo, la plataforma J2ME era la única opción para el desarrollo de *software* destinado a dispositivos con recursos restringidos. Con base en estos antecedentes surge la "aparente" necesidad de implementar dos versiones del *software* propuesto en esta tesis de maestría. La primer versión tendría que ser desarrollada para dispositivos de cómputo que tengan suficientes capacidades de procesamiento (e.g., PC y laptop) utilizando la tecnología J2SE. La segunda versión tendría que ser desarrollada para dispositivos con recursos restringidos (e.g., PDA y PocketPC), empleando la tecnología J2ME. Sin embargo, el desarrollo de dos versiones de un *software* genera una problemática en la fase de implementación, ya que para el presente desarrollo se requiere un solo *software* que sea capaz de detectar cambios en la plataforma y de remodelar la interfaz de usuario de una aplicación colaborativa con base en dichos cambios.

Esta problemática fue resuelta gracias a la máquina virtual de Java **Mysaifu**, la cual se eje-

cuta en dispositivos móviles con recursos restringidos (e.g., PDA y PocketPC). Mysaifu es un *software* libre bajo los términos de la licencia GPLv2 (*GNU Public License versión 2*). El objetivo de los desarrolladores de Mysaifu es crear una máquina virtual de Java que se ajuste a los estándares de la tecnología J2SE (Java2 Standard Edition). Este objetivo hizo posible la implementación del *software* propuesto en la presente tesis de maestría, utilizando únicamente la tecnología J2SE. Los siguientes sistemas operativos son compatibles con la última versión de Mysaifu: a) Windows Mobile 6.0, b) Windows Mobile 5.0 y c) Windows Mobile 2003 Second Edition.

4.2 Principales algoritmos del mecanismo ReSAIC

Con base en el diseño orientado a objetos presentado en el capítulo 3, se han construido los algoritmos que conforman al mecanismo ReSAIC. Dichos algoritmos constituyen el inicio de una solución a la problemática planteada en la presente tesis de maestría. Esta solución basada en objetos está estructurada de tal forma que es posible modificar y/o agregar funcionalidades a dicho mecanismo.

Considere las siguientes definiciones: sea `adaptador` un objeto de la clase `Adaptador`, `núcleo funcional` un objeto de la clase `Núcleo funcional`, `intermediario` un objeto de la clase `Intermediario`, `vigía` un objeto de la clase `Vigía` y `coordinador` un objeto de la clase `Coordinador`.

Algoritmo 1 Adaptación plástica

Entrada: sea x y y el ancho y el alto de la pantalla del dispositivo en pixeles, i el número total de interfaces de usuario prediseñadas y $x_{1...i}$ y $y_{1...i}$ el ancho y el alto de dichas interfaces de usuario

1: **para** $j = 1$ hasta i **hacer**
2: **si** $x_j \leq x$ **y** $y_j \leq y$ **entonces**
3: $d = \sqrt{(x - x_j)^2 + (y - y_j)^2}$
4: $aux = j$
5: salir del ciclo
6: **fin si**
7: **fin para**
8:
9: **para** $j = aux$ hasta i **hacer**
10: **si** $x_j \leq x$ **y** $y_j \leq y$ **entonces**
11: $auxD = \sqrt{(x - x_j)^2 + (y - y_j)^2}$
12: **si** $auxD \leq d$ **entonces**
13: $d = auxD$
14: $aux = j$
15: **fin si**
16: **fin si**
17: **fin para**
18:
19: **devolver** aux

Algoritmo 2 objeto intermediario

Entrada: sea N el total de usuarios y $datos$ una variable que puede tomar dos valores, el primer valor sirve para notificar al usuario local su turno de participación y el segundo valor permite mostrar los efectos de las acciones remotas en la interfaz de usuario local

1: $i = 0$
2: recibir del objeto núcleo funcional $id\_usuarioLocal$
3: conectarse con el objeto vigía
4: enviar $id\_usuarioLocal$ al objeto vigía
5:
6: **mientras** $i < N$ **hacer**
7: recibir $id\_usuario_i$ de un objeto coordinador
8: pasar $id\_usuario_i$ al objeto núcleo funcional
9: $i = i + 1$
10: **fin mientras**
11:
12: **mientras** $cierto$ **hacer**
13: recibir $datos$ de un objeto coordinador
14: **si** $datos$ == $id\_usuarioLocal$ **entonces**
15: pasar $datos$ al objeto núcleo funcional
16: esperar una $cadena$ del objeto núcleo funcional
17: enviar $cadena$ al objeto coordinador
18: **si no**
19: /*$datos$ contiene una acción realizada por un usuario remoto*/
20: pasar $datos$ al objeto núcleo funcional
21: **fin si**
22: $datos = null$
23: **fin mientras**

El algoritmo 1 realiza la adaptación plástica de la interfaz de usuario de una aplicación colaborativa (cf. sub-sección 3.3.1 del Capítulo 3). En las líneas 1-7 se verifica que exista al menos una interfaz de usuario que se adapte al dispositivo de cómputo en el que se ejecuta el mecanismo ReSAIC. Las líneas 9-17 se encargan de seleccionar el identificador que corresponde a la interfaz de usuario que mejor se adapta al dispositivo de cómputo. En la línea 19 se retorna este identificador al `núcleo funcional` de una aplicación colaborativa. En caso de que no exista una interfaz de usuario que se adapte al dispositivo de cómputo, entonces se retorna un valor de cero para indicar la inexistencia.

El algoritmo 2 define la secuencia de pasos de las tareas que realiza un objeto `intermediario` (cf. sub-sección 3.3.2 del Capitulo 3). Específicamente las líneas 2-4 tienen la finalidad de permitir a un usuario el ingreso a la aplicación colaborativa. Posteriormente, el objeto `intermediario` establece comunicación con el objeto `vigía`. El ciclo de las líneas 6-10 se encarga tanto de recibir de un objeto `coordinador` como de pasar al `núcleo funcional` el identificador de los usuarios que ingresan a la aplicación colaborativa. En las líneas 12-23 inicia la sesión colaborativa. Particularmente, en la línea 13 se recibe un dato de un objeto `coordinador`. Este dato puede servir para indicar a un usuario su turno de participación (cf. líneas 14-17) o bien, para indicar una acción de un usuario remoto (líneas 18-21).

Algoritmo 3 objeto vigía

Entrada: sea N el total de usuarios

1: i = 0
2: **mientras** $i < N$ **hacer**
3: crear un $socket$ de comunicación
4: esperar una conexión
5: recibir $id\_usuario_i$ del objeto intermediario$_i$
6: añadir $id\_usuario_i$ a la lista de usuarios
7: crear hilo para atender las peticiones del objeto intermediario$_i$ (cf. Algoritmo 4)
8: informar a todos los hilos creados que se ha conectado un nuevo usuario
9: $i = i + 1$
10: **fin mientras**

Algoritmo 4 objeto coordinador

Entrada: sea j = 0 una variable compartida, $id\_usuariosPendientes$ los identificadores de los usuarios conectados que no han sido enviados al objeto intermediario$_i$

1: $datos$ = $null$
2: **mientras** no se conecten todos los objetos de la clase Intermediario **hacer**
3: **si** se conecta un nuevo objeto intermediario **entonces**
4: actualizar $id\_usuariosPendientes$
5: enviar $id\_usuariosPendientes$ al objeto intermediario$_i$
6: **fin si**
7: **fin mientras**
8:
9: **mientras** $cierto$ **hacer**
10: **si** es el turno del hilo$_j$ **entonces**
11: informar al objeto intermediario$_i$ que es su turno de participación
12: recibir $cadena$ del objeto intermediario$_i$
13: $datos$ = $cadena$
14: distribuir $datos$ al resto de los hilos
15: j = $(j + 1)\%N$
16: **si no**, **si** no es el turno del hilo$_j$ **y** $datos$!= $null$ **entonces**
17: enviar $datos$ al objeto intermediario$_i$
18: $datos = null$
19: **fin si**
20: **fin mientras**

El algoritmo 3 corresponde al objeto `vigía`, el cual se encarga de esperar a que se conecte un grupo de objetos de la clase `Intermediario` (líneas 2-10). Particularmente, el objeto `vigía` crea un *socket* de comunicación (línea 3). Posteriormente, espera a que se conecte un objeto `intermediario` (línea 4). Enseguida, recibe del objeto `intermediario` el identificador de un usuario y lo añade en una tabla *hash* (líneas 5 y 6). A continuación, crea un hilo (línea 7), el cual establece comunicación directa con el objeto `intermediario`. Finalmente, notifica el ingreso de un nuevo usuario a los hilos creados con anterioridad (línea 8).

El algoritmo 4 se refiere a la secuencia de pasos que realizan los hilos creados por el objeto `vigía`. Particularmente, en las líneas 2-7 un hilo envía a un objeto `intermediario` la lista de los usuarios que ingresan a la aplicación colaborativa. Finalizada la fase de ingreso, las líneas 9-20 se encargan de enviar: 1) un dato que puede informar a un usuario sobre su turno de participación (líneas 10-15) o 2) un dato que corresponde a una acción realizada por un usuario remoto (líneas 16-19).

Una vez finalizada la descripción de los principales algoritmos del mecanismo ReSAIC, es necesario probar su funcionamiento en la aplicación de validación "Serpientes y Escaleras". A continuación, se definen las principales tareas de dicha aplicación.

4.3 Principales tareas de la aplicación de validación

En la presente sección se describen las principales tareas que realiza el núcleo funcional de la aplicación de validación "Serpientes y Escaleras". Dichas tareas son: a) ubicar a los jugadores en el tablero del juego y b) verificar que un jugador local se posicionó en la casilla que le corresponde.

4.3.1 Ubicación de los jugadores en el tablero del juego

Es evidente que el tablero del juego "Serpientes y Escaleras" no puede tener el mismo tamaño en la pantalla de una *laptop* que en la de una PDA. Por lo tanto, es necesario diseñar un conjunto de tableros del juego de acuerdo al tamaño de la pantalla de cada dispositivo de cómputo. Sin embargo, un problema presentado durante la fase de implementación consistió en localizar a los jugadores en las casillas de los diversos tableros del juego.

Para abordar este problema, se utilizó una fórmula matemática mejor conocida como regla de tres simple directa [Baldor, 1998], la cual es una forma de resolución de problemas de proporcionalidad entre tres o más valores conocidos y una incógnita. Esta regla establece una relación de linealidad (proporcionalidad) entre los valores involucrados. De manera formal, la regla de tres simple directa enuncia el problema de la siguiente manera:

A es a **B** como **X** es a **Y**

lo que suele representarse así:

$$\begin{aligned} A &\rightarrow B \\ X &\rightarrow Y \end{aligned} \tag{4.1}$$

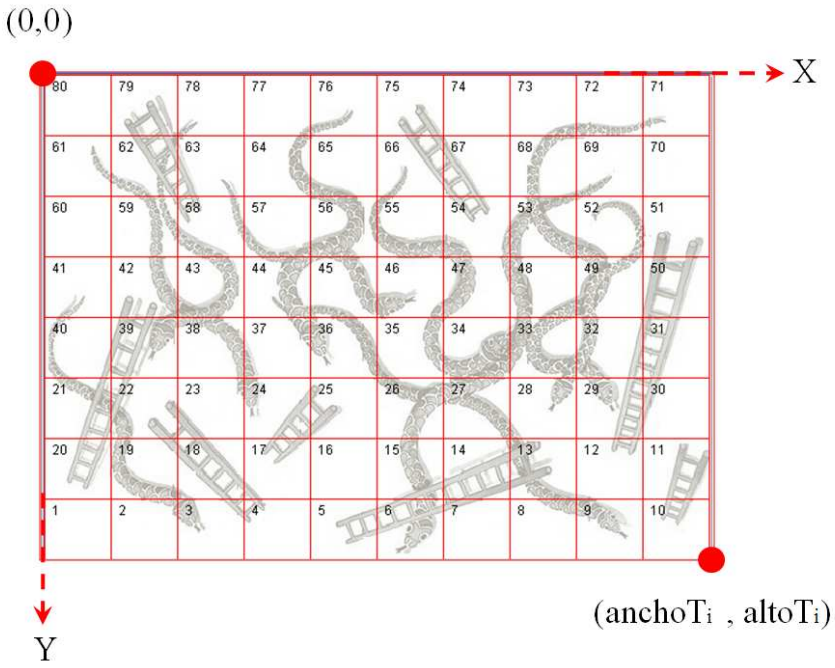


Figura 4.1: Tablero del juego "Serpientes y Escaleras" visualizado en el $dispositivo_i$

Para resolver una regla de tres simple directa basta con aplicar la siguiente fórmula:

$$Y = \frac{B \cdot X}{A} \tag{4.2}$$

Retomando el problema introducido en la presente sección, considere que el usuario u_i se posiciona en la coordenada (x, y) en el tablero del juego t_i (cf., Figura 4.1), entonces ¿Cuál es la coordenada correspondiente en el tablero del juego t_j (cf., Figura 4.2) para definir la posición del jugador? Partiendo de la definición de la regla de tres se tiene que:

$$\begin{aligned} t_i &\rightarrow (x, y) \\ t_j &\rightarrow (?, ?) \end{aligned} \tag{4.3}$$

El tamaño de un tablero t está definido por la coordenada $(anchot, altot)$ siempre y cuando su origen se localice en la coordenada $(0, 0)$. Por lo tanto, se tiene que:

$$\begin{aligned} (anchot_i, altot_i) &\rightarrow (x, y) \\ (anchot_j, altot_j) &\rightarrow (u, w) \end{aligned} \tag{4.4}$$

En donde:

$$u = \frac{anchot_j \cdot x}{anchot_i} \tag{4.5}$$

y

$$w = \frac{altot_j \cdot y}{altot_i} \tag{4.6}$$

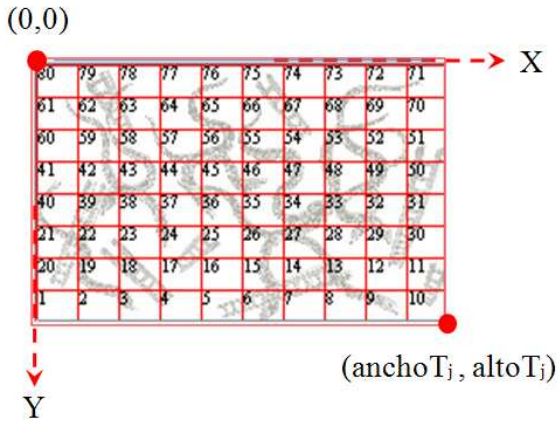


Figura 4.2: Tablero del juego "Serpientes y Escaleras" visualizado en el $dispositivo_j$

A guisa de ilustración, suponga que el usuario Rafael interactúa desde una PC con la aplicación "Serpientes y Escaleras". En dicho dispositivo el tamaño del tablero es de 540 pixeles de ancho por 440 pixeles de alto. En un tiempo t el usuario Rafael avanza a la tercera casilla del tablero, posicionándose en la coordenada (144, 415). ¿Cuál es la coordenada correspondiente para ubicar la posición de Rafael en una PDA que tiene un tamaño de tablero de 207 pixeles de ancho por 144 pixeles de alto? Con base en la regla de tres (cf. formulas 4.1, 4.2, 4.3, 4.4, 4.5 y 4.6) se tiene que:

$$(540, 440) \rightarrow (144, 415)$$
$$(207, 144) \rightarrow (u, w)$$

En donde:

$$u = \frac{207 \cdot 144}{540} = 55$$

y

$$w = \frac{144 \cdot 415}{440} = 135$$

4.3.2 Verificación del posicionamiento de un jugador en la casilla correcta

Uno de los objetivos del juego "Serpientes y Escaleras" es complementar las actividades escolares, principalmente en educación básica. Para lograr dicho objetivo, es necesario verificar que el usuario (e.g., un niño) se posicione en la casilla correcta del tablero del juego. En caso de que el usuario no se coloque en la casilla que le corresponde, no podrá cambiar su ubicación en el tablero. Esta verificación motiva al usuario a llevar un conteo correcto de los números que obtiene para cambiar de posición en el tablero.

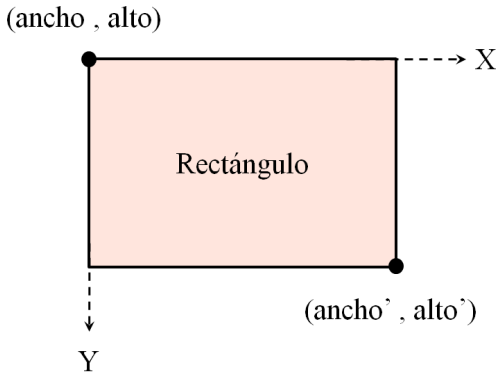


Figura 4.3: Casilla del tablero del juego "Serpientes y Escaleras"

Un rectángulo dibujado en un contenedor (e.g., un panel de la *api java.awt*) tiene una coordenada $(ancho, alto)$ que corresponde a la esquina superior izquierda y una coordenada $(ancho', alto')$ que corresponde a la esquina inferior derecha (cf. Figura 4.3). Particularmente, el tablero del juego "Serpientes y Escaleras" está formado por ochenta rectángulos, i.e., cada casilla del tablero es un rectángulo.

Para ejemplificar el proceso de verificación suponga que (x, y) es la coordenada en donde el usuario hace *click* sobre el tablero del juego y que la Figura 4.3 representa la casilla en la cual dicho usuario debe posicionarse realmente (cf. línea 1 del algoritmo 5). A continuación, es necesario comprobar si esta coordenada es un punto que se localiza en el interior de la casilla (cf. línea 2 del algoritmo 5). Si dicha comprobación resulta afirmativa entonces se permite al usuario cambiar de casilla (cf. línea 3 del algoritmo 5), de lo contrario significa que es un movimiento incorrecto y por lo tanto no permitido (cf. líneas 4 y 5 del algoritmo 5).

Algoritmo 5 Ubicar la posicion del usuario

Entrada: sea (x, y) la coordenada en donde el usuario se coloca en el tablero del juego

1: localizar la casilla en la que el usuario realmente debe posicionarse
2: **si** $(x < ancho'$ **y** $y < alto')$ **y** $(x > ancho$ **y** $y > alto)$ **entonces**
3: permitir al usuario cambiar de casilla
4: **si no**
5: no permitir al usuario cambiar de casilla
6: **fin si**

Después de haber presentado las principales tareas que realiza la aplicación "Serpientes y Escaleras", es necesario definir sus interfaces de usuario, las cuales tienen la finalidad de permitir la interacción entre el usuario y la aplicación colaborativa. Estas interfaces de usuario también tienen la finalidad de servir como guía al desarrollador que desee implementar las interfaces de usuario de una aplicación colaborativa empleando el mecanismo ReSAIC.

4.4 Interfaz de usuario

La presente sección está divida en tres partes. En la primera parte, se explica en qué consiste el sistema de coordenadas de una interfaz de usuario en Java. En la segunda parte, se describe el proceso de colocación de componentes gráficos en una interfaz de usuario. Finalmente, se muestran las interfaces de usuario de la aplicación "Serpientes y Escaleras" para una PC y una PDA, la cual puede mostrar tanto vista vertical como horizontal. La selección de la interfaz de usuario para la PDA no solo se basa en la adecuada organización y apariencia de sus componentes gráficos, sino también en su funcionalidad, ya que el espacio de la pantalla de estos dispositivos es muy reducido en comparación con el de la PC.

4.4.1 Sistema de coordenadas

En Java, un componente gráfico es cualquiera de los botones, etiquetas, listas, menús, etc, que se pueden colocar en una ventana. También existen componentes gráficos llamados contenedores, los cuales no tienen una forma definida, ya que únicamente están destinados a contener otros componentes, e.g., si una ventana contiene dos botones, esta actúa como contenedor.

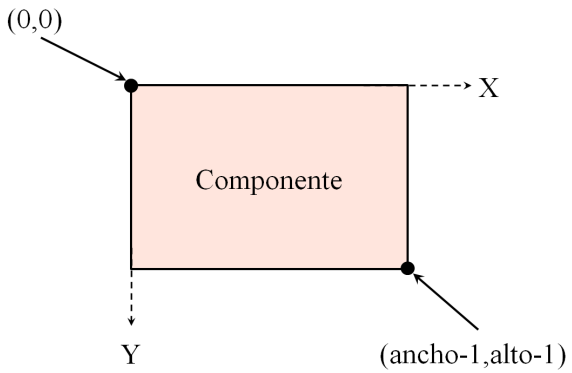


Figura 4.4: Componente de Java

Cada uno de los componentes y contenedores gráficos de Java tiene su propio sistema de coordenadas, el cual inicia en la posición (0, 0) y finaliza en la posición determinada tanto por la anchura, como por la altura total del componente gráfico (en pixeles) menos una unidad. Como se puede apreciar en la Figura 4.4, la esquina superior izquierda del componente gráfico tiene asignadas las coordenadas (0, 0). La coordenada en el eje de abscisas se incrementa hacia la derecha, en tanto que la coordenada en el eje de las ordenadas aumenta hacia abajo.

Para pintar un componente gráfico, se debe tener en cuenta además de su tamaño, el tamaño de su borde, si lo tuviera, e.g., un borde alrededor de un componente que ocupa un pixel, haría que las coordenadas de la esquina superior izquierda pasen de (0, 0) a (1, 1). Las dimensiones de un componente gráfico pueden conocerse a través de sus métodos *getWidth()* y *getHeight()*, en tanto que el método *getInsets()* permite obtener el tamaño del borde.

4.4.2 Concepto de *layout*

Todos los contenedores gráficos de Java tienen una clase, cuyo nombre genérico es *Layout*, que es responsable de distribuir los botones dentro del contenedor, e.g., la clase *FlowLayout* se encarga de posicionar los botones secuencialmente de izquierda a derecha, como sería el caso de la barra de herramientas de un navegador Web, mientras que la clase *GridLayout* los coloca en forma de matriz[2].

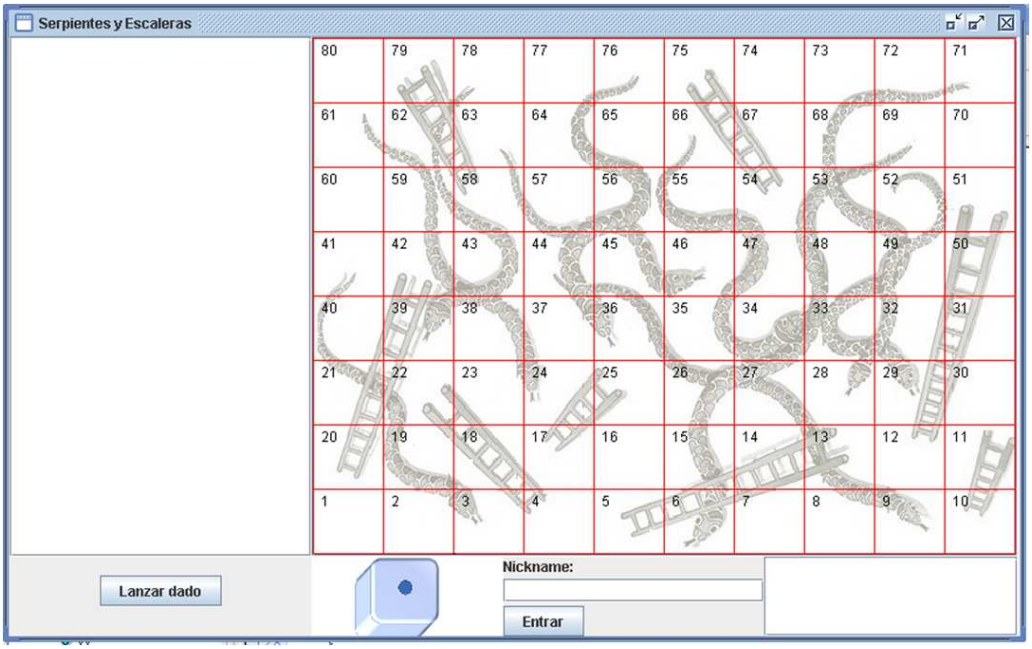


Figura 4.5: Ventana de la aplicación "Serpientes y Escaleras"

Cualquier contenedor gráfico de Java tiene una clase *Layout* por defecto, pero se puede cambiar con el método *setLayout*. En Java existen varias clases *Layouts*, la mayoría de ellas son sencillas de usar, ya que ofrecen pocas opciones de configuración. Estas clases se utilizan para crear ventanas con una distribución de botones no muy compleja (e.g., la barra de herramientas de un navegador Web o el teclado de una calculadora) en la que todos los botones son del mismo tamaño. Sin embargo, las clases *Layout* no son suficientes para construir ventanas más complejas.

El lenguaje Java proporciona una clase *Layout* llamada *GridBagLayout*, la cual es muy potente, pero muy compleja de utilizar. La clase *Layout* será suficiente para hacer casi cualquier distribución de botones en una ventana, por compleja que parezca, pero se debe entender su funcionamiento para que el resultado final se parezca a lo que se quiere.

A continuación, se describe el proceso para crear una ventana con sus respectivos componentes utilizando la clase *GridBagLayout*. La ventana que se pretende obtener se muestra en la Figura 4.5.

Diseño de la rejilla

[2]http://www.chuidiang.com/java/layout/GridBagLayout/GridBagLayout.php

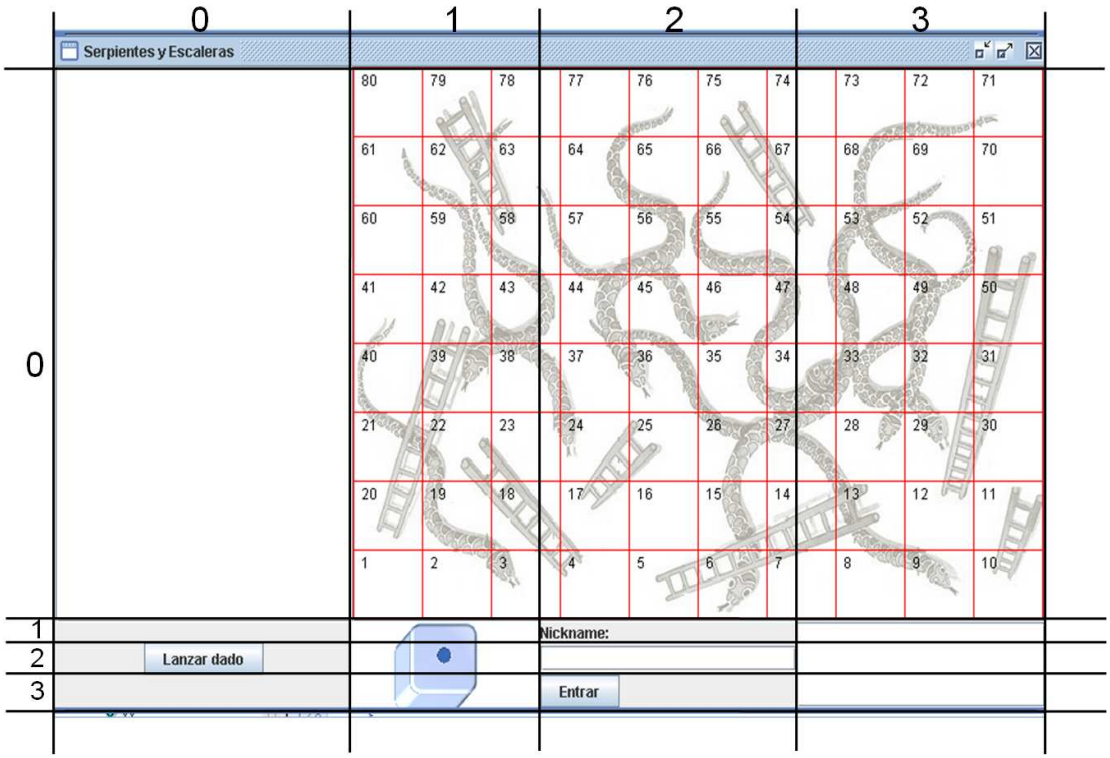


Figura 4.6: Trazado de rejillas sobre la ventana de la aplicación “Serpientes y Escaleras”

Primeramente, se debe dibujar en papel la ventana que se quiere obtener. Es importante dibujarla suficientemente estirada para tener claro si los botones deben hacerse grandes o no. Luego, hay que trazar líneas horizontales y verticales para tratar de delimitar la rejilla en la que estarán colocados los botones. Para trazar esta rejilla conviene tener en cuenta lo siguiente:

- cada componente puede ocupar una o más celdas, pero dos componentes gráficos no pueden ocupar la misma celda;
- no es necesario que todas las celdas de la rejilla sean del mismo tamaño ni se requiere que un componente gráfico ocupe una celda completa; y
- es conveniente que el componente gráfico ocupe toda una celda y que esté centrado en la celda o que esté pegado a uno de sus bordes.

Particularmente, un ejemplo de rejilla para la ventana de la aplicación “Serpientes y Escaleras” se muestra en la Figura 4.6. El panel superior izquierdo ocupa una celda, en tanto que el panel inferior derecho, el panel que contiene el tablero del juego y el panel que contiene el dado ocupan tres celdas cada uno. La etiqueta, el campo de texto y los botones ocupan una celda cada uno.

Primera aproximación para la colocación de componentes gráficos en una ventana

La clase *GridBagConstraints* guarda en sus atributos información de cómo y dónde añadir un componente gráfico. A continuación, se describen cuatro de estos atributos:

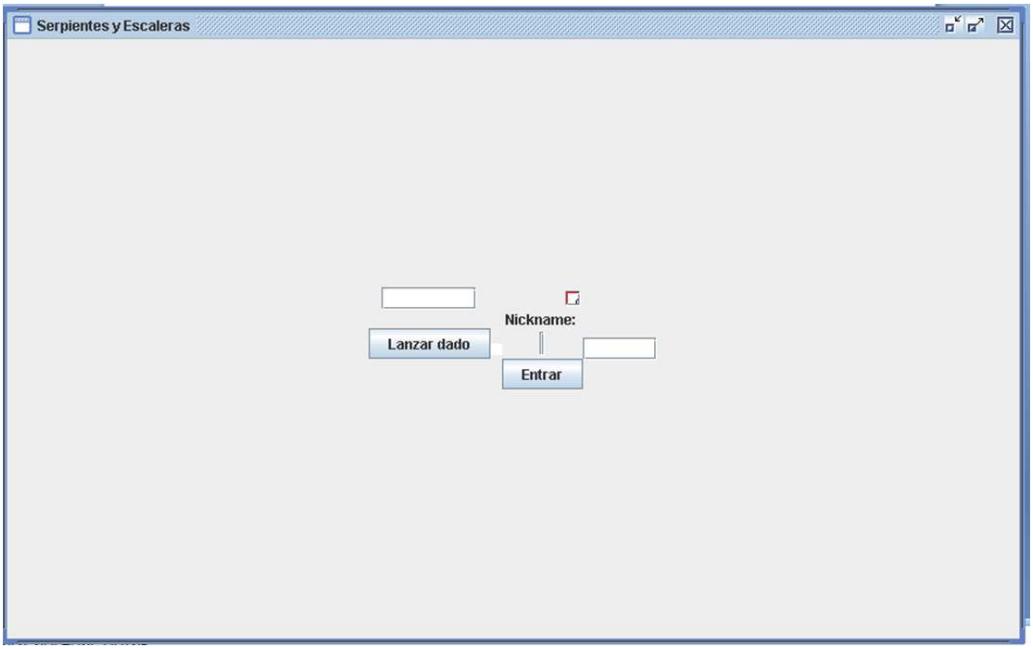


Figura 4.7: Componentes agrupados en el centro de una ventana

- ***GridBagConstraints.gridx*** hace referencia a la posición de un componente gráfico en el eje x, i.e., el número de columna en la que está posicionado, donde la columna 0 es la primera columna de la parte izquierda de la ventana. Si el componente gráfico ocupa varias columnas (e.g., el panel en donde se dibuja el tablero del juego), se debe indicar la columna en la que está ubicada su esquina superior izquierda.
- ***GridBagConstraints.gridy*** hace referencia a la posición de un componente gráfico en el eje y, i.e., el número de fila en la que está posicionado, donde la fila 0 es la primera fila de la parte superior de la ventana. Si el componente gráfico ocupa varias filas (e.g., el panel de la esquina inferior derecha de la ventana), se debe indicar la fila en la que está colocada su esquina superior izquierda.
- ***GridBagConstraints.gridwidth*** hace referencia al número de celdas horizontales que debe ocupar un componente gráfico, i.e., ancho.
- ***GridBagConstraints.gridheight*** indica el número de celdas verticales que debe ocupar un componente gráfico, i.e., alto.

La Figura 4.7 muestra los resultados de ejecutar la aplicación "Serpientes y Escaleras" después de configurar los cuatro atributos de la clase *GridBagConstraints*. Evidentemente, la ventana que se muestra en dicha figura no se parece a lo que se pretende obtener. Se observa que todos los componentes se encuentran agrupados en el centro de la ventana y no tienen el tamaño adecuado. Esta configuración se debe a que la clase *GridBagConstraints* únicamente se encarga de colocar cada componente en su celda correspondiente. Sin embargo, falta especificar en dicha clase la forma en que se deben estirar las filas y columnas, las cuales tienen el tamaño mínimo necesario para albergar a sus componentes. En la Figura 4.8 se observa que las filas y columnas están ubicadas en el centro de la ventana.

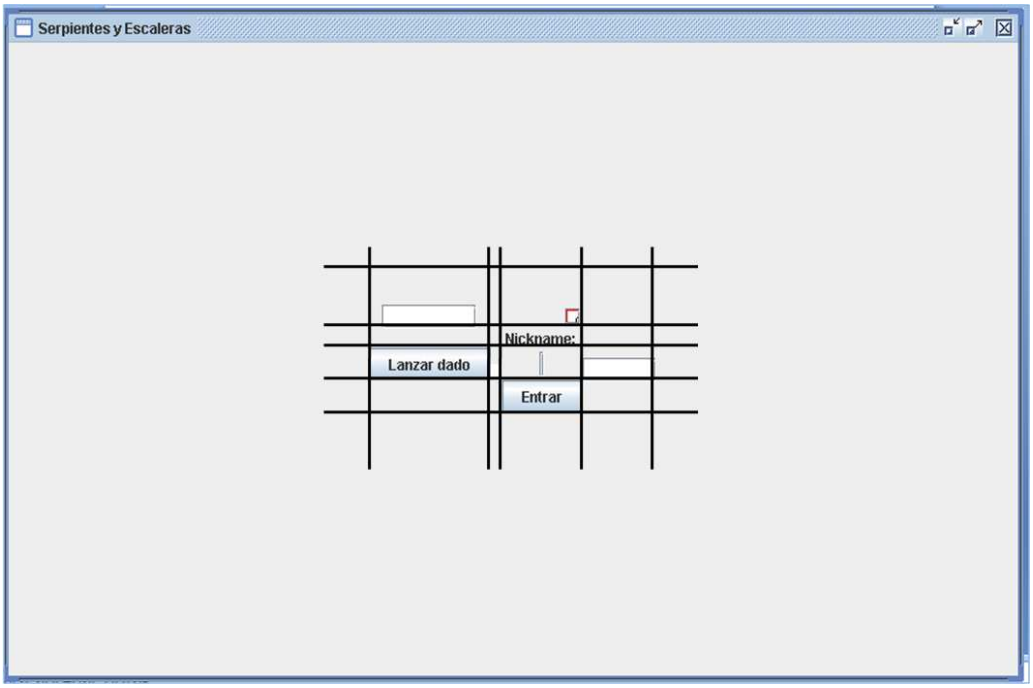


Figura 4.8: Ubicación de las filas y columnas en la ventana

Estiramiento de filas y columnas

El siguiente paso consiste en definir la forma en que se deben estirar las filas y columnas, para lo cual se utilizan los campos *weigthx* y *weigthy* de la clase *GridBagConstraints*. Estos campos indican cómo estirar respectivamente las columnas y filas. A estos campos se les da el valor 0.0 (que es el valor por defecto) si no se quiere que la fila o columna se estire. En caso contrario, se asigna el valor 1.0. El problema de estos campos es que afectan a una fila o columna completa. Por esta razón, cada vez que se añade un componente gráfico a una fila o columna, se debe asignar el mismo valor (0.0 o 1.0).

La Figura 4.9 muestra los resultados de ejecutar la aplicación "Serpientes y Escaleras" una vez configurados los campos *weigthx* y *weigthy*. Evidentemente, dicha figura tampoco se parece a la ventana que se desea obtener. La clase *GridBagLayout* se encarga de estirar las filas y columnas de forma correcta, sin embargo falta especificar en dicha clase que también se deben estirar los componentes gráficos.

En la Figura 4.10 se muestran los trazos que delimitan las filas y columnas. Estos trazos confirman que efectivamente se han estirado las filas y columnas. Sin embargo, los componentes gráficos mantienen su tamaño original y están ubicados en el centro de las celdas que tienen asignadas.

Estiramiento de componentes gráficos

El siguiente paso consiste en estirar los componentes gráficos (e.g., el panel superior izquierdo y el panel que contiene el tablero del juego). El atributo *fill* de la clase *GridBagConstraints*

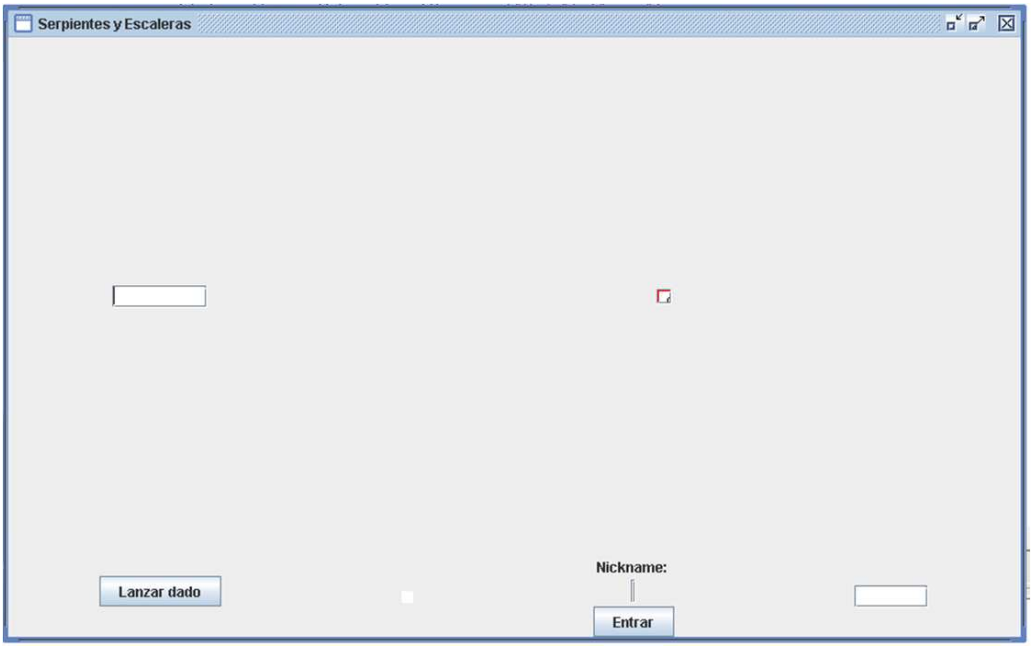


Figura 4.9: Ventana con las filas y columnas estiradas

permite que un componente gráfico se estire en una ventana. Dicho atributo puede tomar los siguientes valores:

- ***GridBagConstraints.NONE***: permite que un componente gráfico no se estire en ningún sentido (opción por *default*);
- ***GridBagConstraints.VERTICAL***: propicia que un componente gráfico se estire verticalmente;
- ***GridBagConstraints.HORIZONTAL***: permite que un componente gráfico se estire horizontalmente; y
- ***GridBagConstraints.BOTH***: posibilita que un componente gráfico se estire tanto horizontal como verticalmente.

Si un componente gráfico no requiere ser estirado en ninguna dirección, se podrá indicar, por medio del atributo *anchor* de la clase *GridBagConstraints*, la posición de la celda en la que debe ser colocado. Este atributo puede tomar los siguientes valores:

- ***GridBagConstraints.CENTER***: permite que un componente gráfico ocupe el centro de la celda (opción por *default*),
- ***GridBagConstraints.NORTH***: propicia que un componente gráfico se centre en la parte superior de la celda,
- ***GridBagConstraints.NORTHEAST***: permite que un componente gráfico se posicione en la esquina superior derecha de la celda,

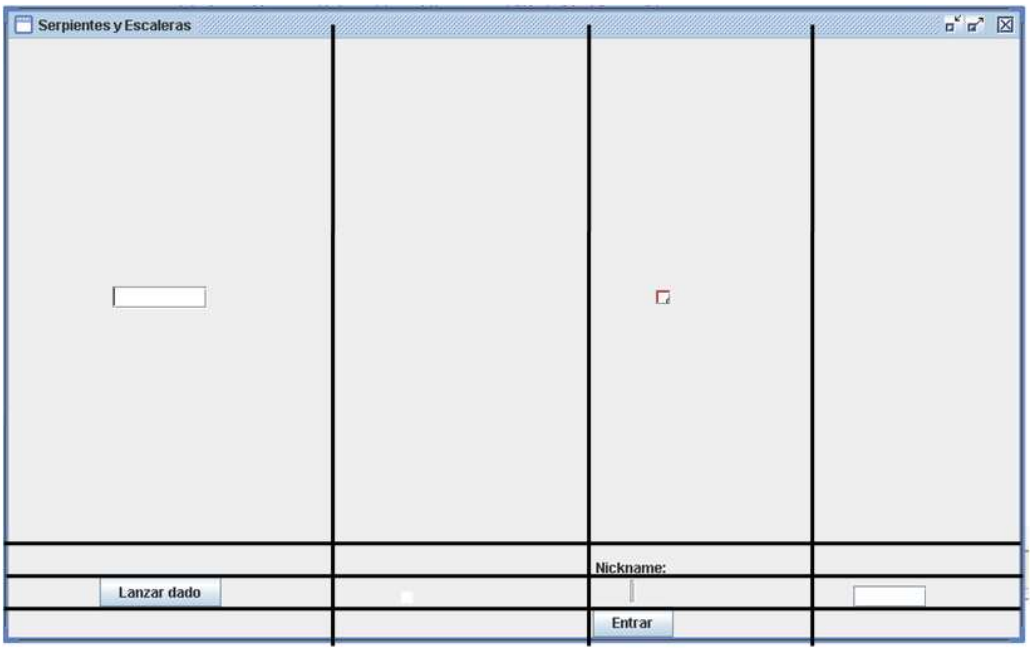


Figura 4.10: Trazo de líneas que delimitan filas y columnas en una ventana

- ***GridBagConstraints.WEST***: posibilita que un componente gráfico se centre en el lado izquierdo de la celda y
- ***GridBagConstraints.NORTHWEST***: permite que un componente gráfico se posicione en la esquina superior izquierda de la celda.

Finalmente, una vez que los atributos *fill* y *anchor* han sido configurados correctamente se obtiene la ventana que se muestra en la Figura 4.5.

4.4.3 Interfaces de usuario de la aplicación "Serpientes y Escaleras"

La Figura 4.11 muestra la interfaz de usuario de la aplicación "Serpientes y Escaleras" para una PC y una *laptop*. Esta interfaz de usuario se compone de cinco partes:

- la parte superior izquierda muestra los nombres de los usuarios de la aplicación;
- la parte superior derecha muestra el tablero del juego;
- la parte inferior izquierda muestra el botón mediante el cual se solicita a la aplicación que genere un número aleatorio entre 1 y 6;
- la parte inferior central permite al usuario ingresar a la aplicación; y
- la parte inferior derecha muestra el semáforo que indica los turnos de participación de los usuarios en la aplicación.

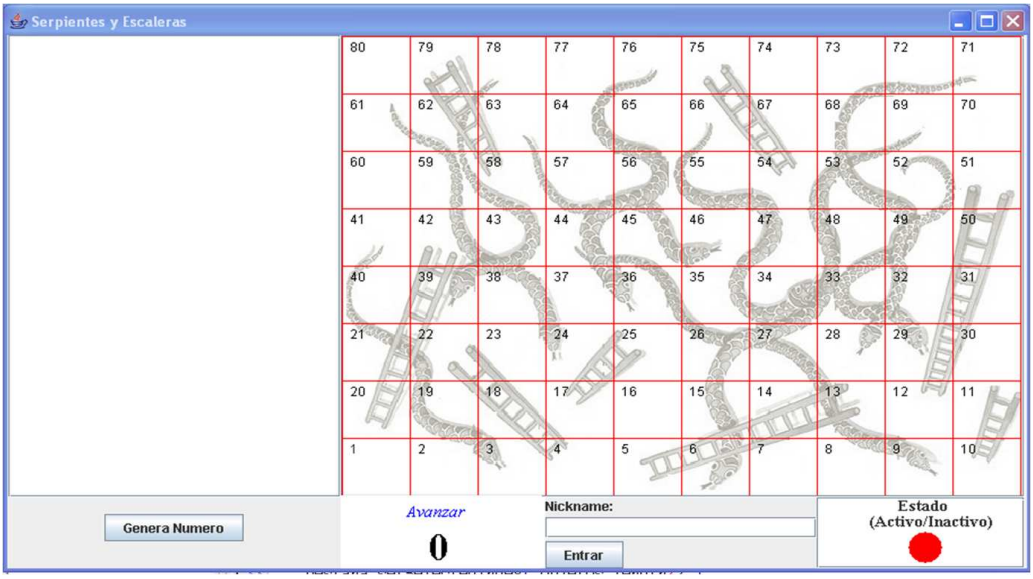


Figura 4.11: Interfaz de usuario de la aplicación "Serpientes y Escaleras" en una PC

Por otro lado, en la interfaz de usuario de la aplicación "Serpientes y Escaleras" para una PDA es imposible mostrar todos los componentes gráficos en una sola vista, debido al reducido tamaño de la pantalla del dispositivo. Con base en estos antecedentes, es necesario dividir la interfaz de usuario en dos partes, las cuales están representadas por las pestañas **Juego** y **Colaboradores** (cf. Figura 4.12). El usuario puede cambiar de pestaña con tan solo hacer *click* sobre ella.

Para la inserción de pestañas en la interfaz de usuario de una PDA se utiliza la clase *JTabbedPane*, la cual organiza un grupo de contenedores. El método *addTab* de dicha clase se encarga de mostrar una pestaña. En el primer argumento de este método se especifica el texto que se requiere colocar sobre la pestaña, mientras que en el segundo argumento se especifica la vista que se necesita mostrar cuando se seleccione la pestaña.

La vista que se muestra cuando se selecciona la pestaña **Juego** se compone de dos partes:

- la parte superior muestra el tablero del juego; y
- la parte inferior izquierda muestra un botón, mediante el cual se solicita a la aplicación que genere un número aleatorio entre 1 y 6.

La vista que se muestra cuando se selecciona la pestaña **Colaboradores** se compone de tres partes:

- la parte superior izquierda muestra los nombres de los usuarios de la aplicación;
- la parte inferior izquierda muestra el semáforo que indica los turnos de participación de los usuarios en la aplicación; y
- la parte inferior derecha permite al usuario ingresar a la aplicación.

Figura 4.12: Interfaz de usuario de la aplicación “Serpientes y Escaleras” en una PDA colocada en posición vertical

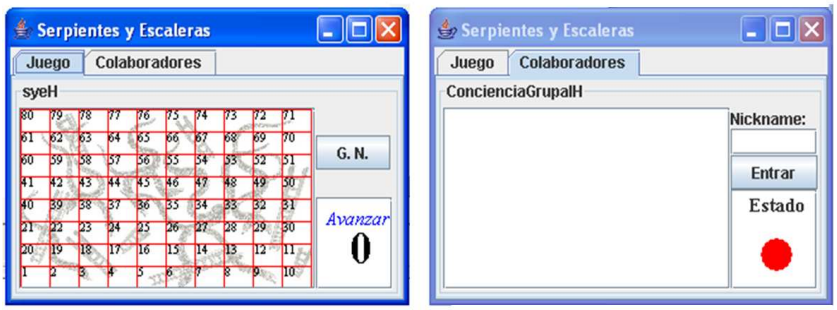


Figura 4.13: Interfaz de usuario de la aplicación “Serpientes y Escaleras” en una PDA colocada en posición horizontal

Finalmente, la Figura 4.13 muestra la interfaz de usuario de la aplicación “Serpientes y Escaleras” para una PDA, cuando se rota horizontalmente el despliegue del dispositivo de cómputo. Dicha interfaz de usuario es similar a la que se muestra en la Figura 4.12. Sin embargo, algunos componentes gráficos cambiaron de posición para aprovechar el tamaño de la pantalla del dispositivo de cómputo.

Con base en los requerimientos de la aplicación “Serpientes y Escaleras”, sus interfaces de usuario están diseñadas e implementadas para soportar un máximo de cuatro usuarios. En versiones anteriores de dichas interfaces de usuario se utilizaban imágenes de tipo .png para simular el lanzamiento del dado (cf. Figura 4.5). Sin embargo, el proceso de cargar imágenes en dispositivos con recursos restringidos (e.g., PDA) ocasiona que la aplicación sea ineficiente para el usuario. Con base en esta problemática solo se muestran números del 1 al 6 para sustituir el dado.

4.5 Pruebas y resultados

Para participar en el juego se requiere que los usuarios cuenten con una copia de la aplicación "Serpientes y Escaleras" en sus respectivos dispositivos (i.e., un archivo ejecutable Java). Adicionalmente, los usuarios que utilizan dispositivos móviles (e.g., PDA o PocketPC) deben instalar una versión actual de la maquina virtual Mysaifu. Para comenzar el juego, es necesario que todos los usuarios sean invitados (e.g., a través de mensajes de texto) y acepten dicha invitación. Cabe señalar que debe existir una persona a la que se le asigne la tarea de inicializar el servidor, en el que se especifica el número de usuarios que deben ser coordinados.

Las pruebas referentes al mecanismo ReSAIC se llevaron a cabo en una red privada con una dirección IP fija en cada dispositivo. Estas pruebas se realizaron con tres usuarios: *Roberto*, *Carlos* y *Nubia*.

En el tiempo t, *Roberto* ejecuta en una PC el código del mecanismo ReSAIC que corresponde al servidor (cf. subsección 3.3.2 del capítulo 3). En un tiempo $t+1$, los tres usuarios ejecutan el código del mecanismo ReSAIC que se localiza en sus respectivos dispositivos de cómputo (cf. subsección 3.2.1 y 3.2.2 del capítulo 3). Como consecuencia de dicha ejecución, *Roberto*, quien interactúa desde una PC, visualiza la interfaz de usuario que se muestra en la Figura 4.14.a. Por su parte *Carlos*, quien interactúa desde una PDA (vista vertical), percibe la interfaz de usuario que se muestra en la Figura 4.14.b. Finalmente *Nubia*, quien también interactúa desde una PDA (vista horizontal), visualiza la interfaz de usuario que se muestra en la Figura 4.14.c.

En el tiempo $t+2$, *Roberto* ingresa a la aplicación "Serpientes y Escaleras", la cual se encarga de generar aleatoriamente un color para representar a dicho usuario. Posteriormente, se muestra en el panel superior izquierdo de su interfaz de usuario tanto el color que identifica a *Roberto* como su *nickname* (cf. Figura 4.15.a). En las Figuras 4.15.b y 4.15.c se puede observar que no hay cambios en las interfaces de usuario de *Carlos* y *Nubia* en el tiempo $t+2$ porque aún no han ingresado a la aplicación.

En el tiempo $t+3$, *Carlos* ingresa a la aplicación "Serpientes y Escaleras". El panel central localizado en la ventana asociada a la pestaña **Colaboradores** muestra los nombres de los usuarios que han ingresado a la aplicación (cf. Figura 4.16.b). Estos nombres también se muestran en la interfaz de usuario de *Roberto* (cf. Figura 4.16.c). Por su parte, la interfaz de usuario de *Nubia* permanece sin cambios (cf. Figura 4.16.c) porque todavía no ha ingresado a la aplicación.

En el tiempo $t+4$, *Nubia* ingresa a la aplicación "Serpientes y Escaleras". En todas las interfaces de usuario se muestran los nombres de los tres usuarios (cf. Figuras 4.17.a, 4.17.b y 4.17.c). En este momento, inicia el juego. Los turnos de participación se asignan de acuerdo al orden en que los usuarios ingresaron a la aplicación. Con base en esta asignación, inicia *Roberto*. Para notificarle a dicho usuario que es su turno de participar, el semáforo localizado en el panel inferior derecho de su interfaz de usuario cambia de color **rojo** a **verde** (cf. Figura 4.17.a).

En el tiempo $t+5$, *Roberto* oprime el botón **Genera Número**. En respuesta, la aplicación genera aleatoriamente un número entero entre 1 y 6, el cual se muestra en el panel que se localiza a la derecha de dicho botón. Este número indica cuentas casillas debe avanzar *Roberto* en el tablero (en este caso, son tres casillas). Mediante un *click* del *mouse* dicho usuario se posiciona en la casilla correcta. En todos los tableros del juego se muestra el movimiento realizado por *Roberto* (cf. Figuras 4.18.a, 4.18.b y 4.18.c). Ahora, toca el turno de *Carlos*. Por lo tanto, el

semáforo que visualiza dicho usuario cambia a color **verde** (cf. Figura 4.18.b).

En el tiempo $t+6$, *Carlos* oprime el botón **Genera Número**, obtiene el número **dos** y avanza en el tablero. El movimiento realizado por *Carlos* es visualizado en el tablero de cada usuario (cf. Figuras 4.19.a, 4.19.b y 4.19.c). *Nubia* concluye la primera ronda del juego. Para notificarle que es su turno de participar, el semáforo localizado en el panel inferior derecho de la ventana asociada a la pestaña **Colaboradores** en la PDA cambia a color **verde** (cf. Figura 4.19.c).

En el tiempo $t+7$, *Nubia* oprime el botón **Genera Número**, obtiene el número **cuatro** y avanza en el tablero (cf. Figura 4.20.c). Sin embargo, en la casilla cuatro se localiza la base una escalera, por lo que *Nubia* deberá avanzar a la casilla en donde termina dicha escalera. En las interfaces de usuario de *Roberto* y *Carlos* no se muestra el movimiento realizado por *Nubia* (cf. Figuras 4.20.a y 4.20.b), porque ella aún no se ha posicionado en la casilla que le corresponde.

En el tiempo $t+8$, *Nubia* avanza a la casilla en donde termina la escalera. El movimiento realizado por ella ahora es visualizado en el tablero de cada usuario (cf. Figuras 4.21.a, 4.21.b y 4.21.c). Al concluir la primera ronda del juego, es el turno nuevamente de *Roberto*. El juego prosigue siguiendo las reglas descritas, de manera que el usuario que llegue primero a la última casilla del tablero gana el juego.

Con base en los resultados presentados anteriormente, se concluye que el mecanismo ReSAIC supera la prueba de portabilidad, ya que la aplicación "Serpientes y Escaleras" se ejecuta de manera correcta en los diferentes dispositivos de cómputo. Evidentemente, este mecanismo dota a dicha aplicación de la propiedad de plasticidad, ya que se muestra la interfaz de usuario adecuada en cada dispositivo de cómputo. En cuanto a las pruebas de comunicación se observa que existe un flujo continuo de datos entre los clientes y el servidor, por lo tanto también se comprueba que el servidor realiza de forma eficiente la coordinación entre los usuarios.

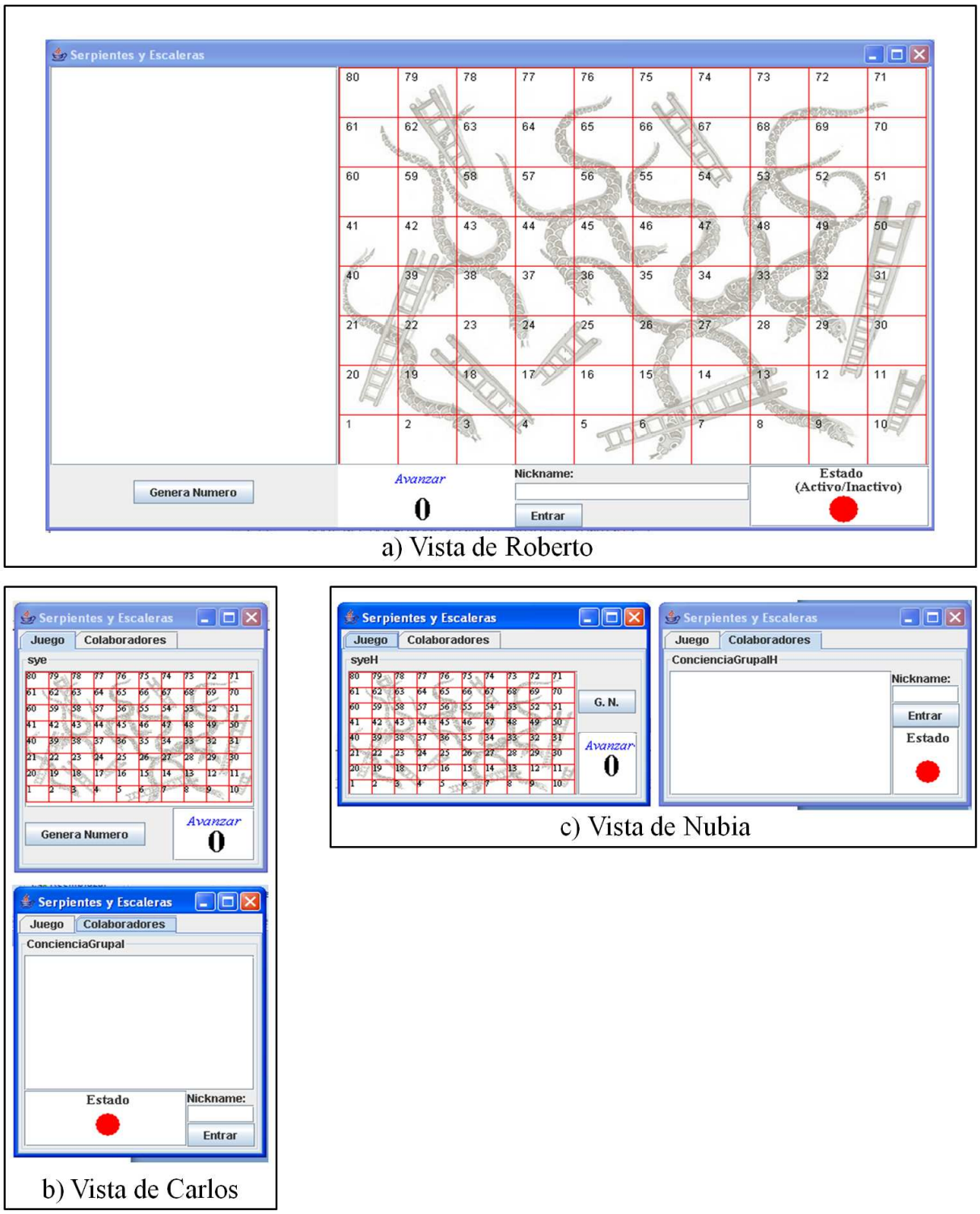


Figura 4.14: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 1$

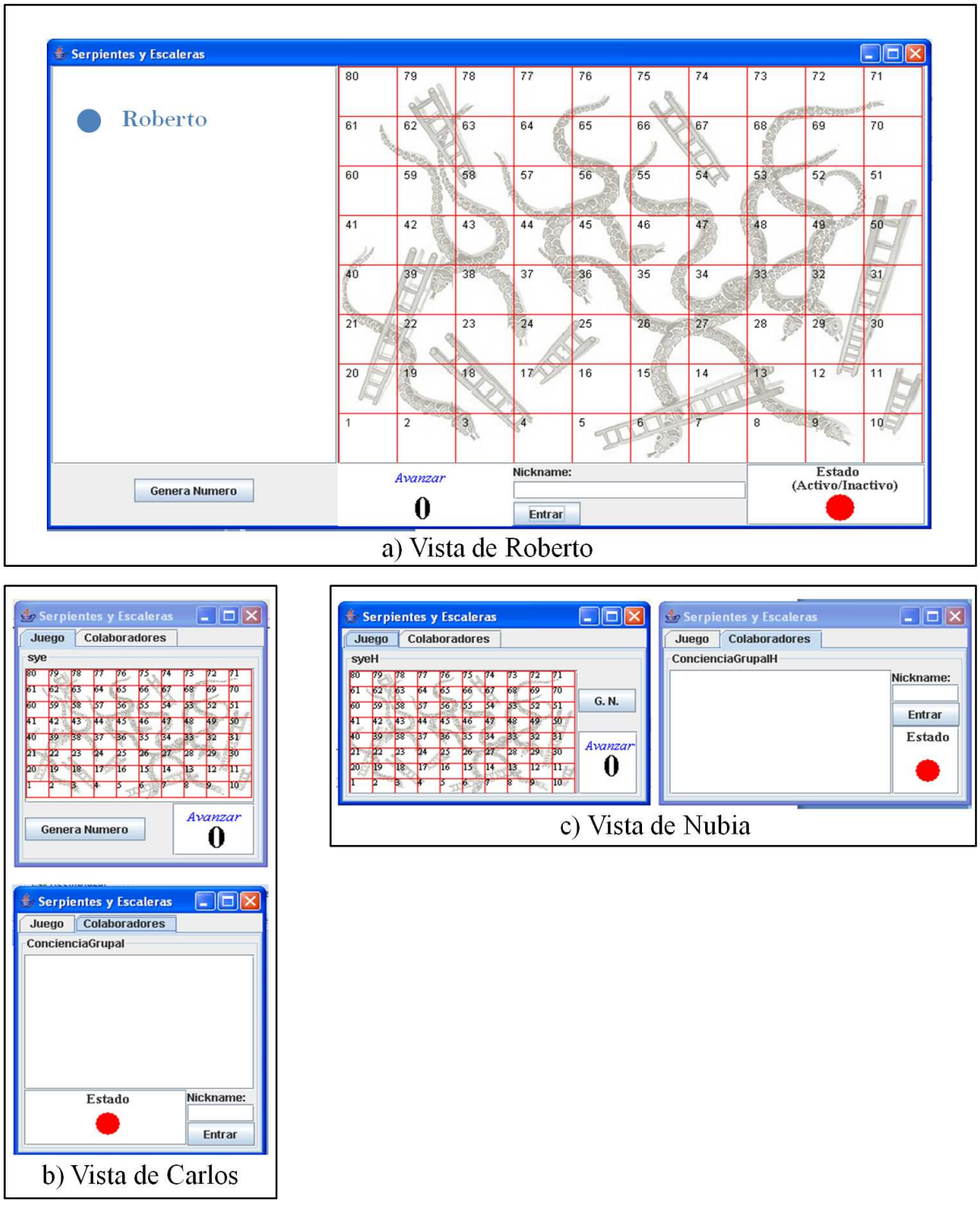


Figura 4.15: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 2$

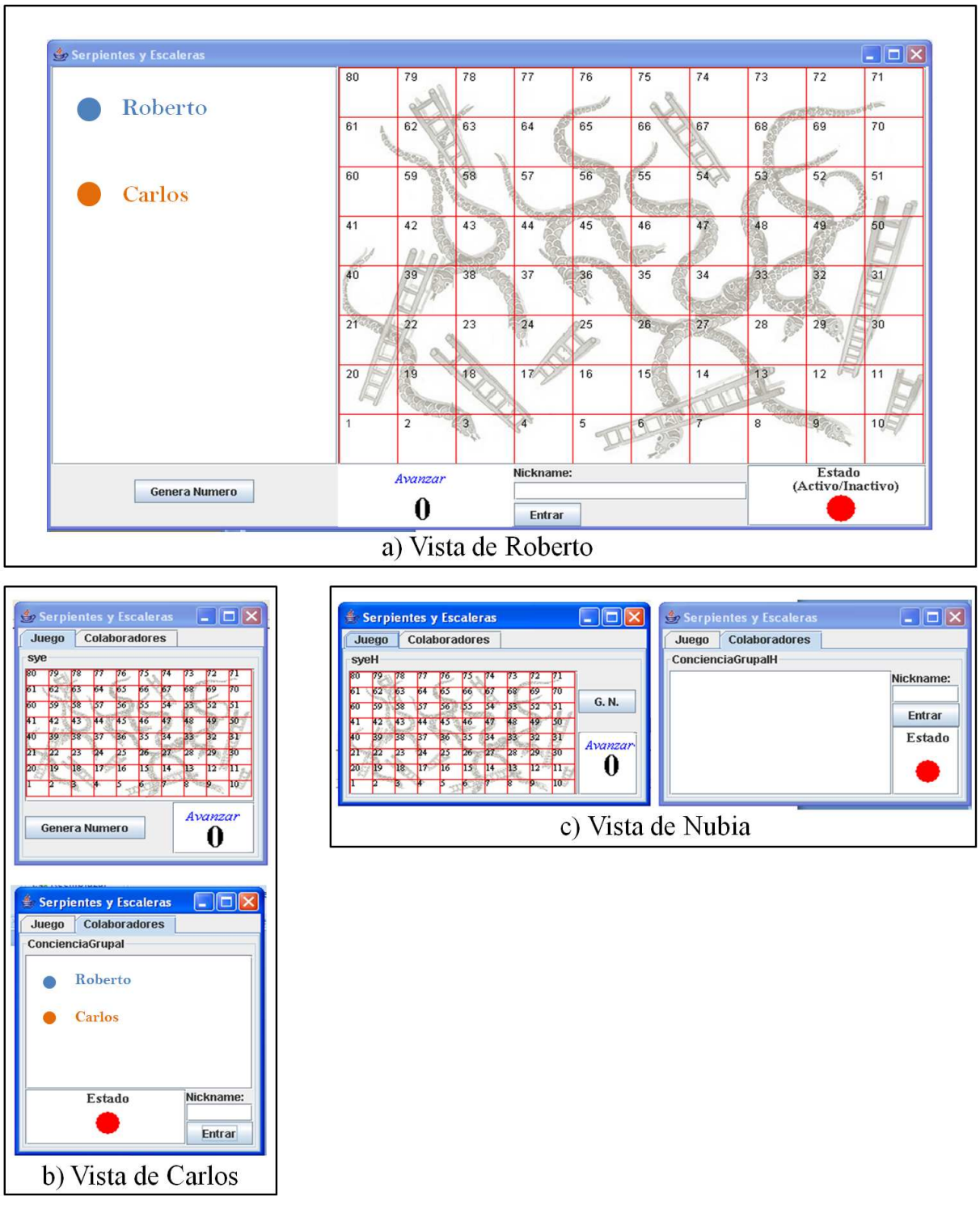


Figura 4.16: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 3$

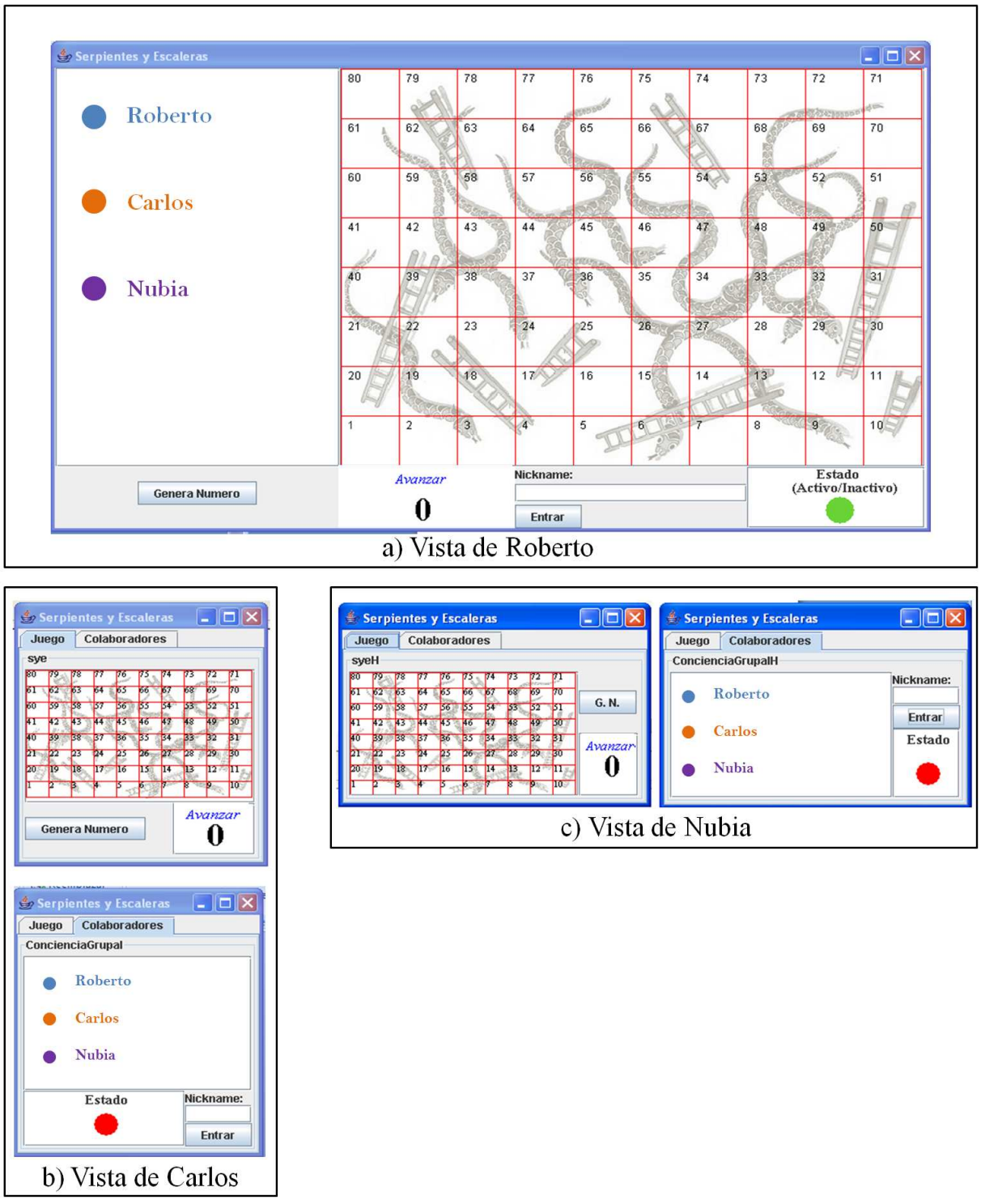


Figura 4.17: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 4$

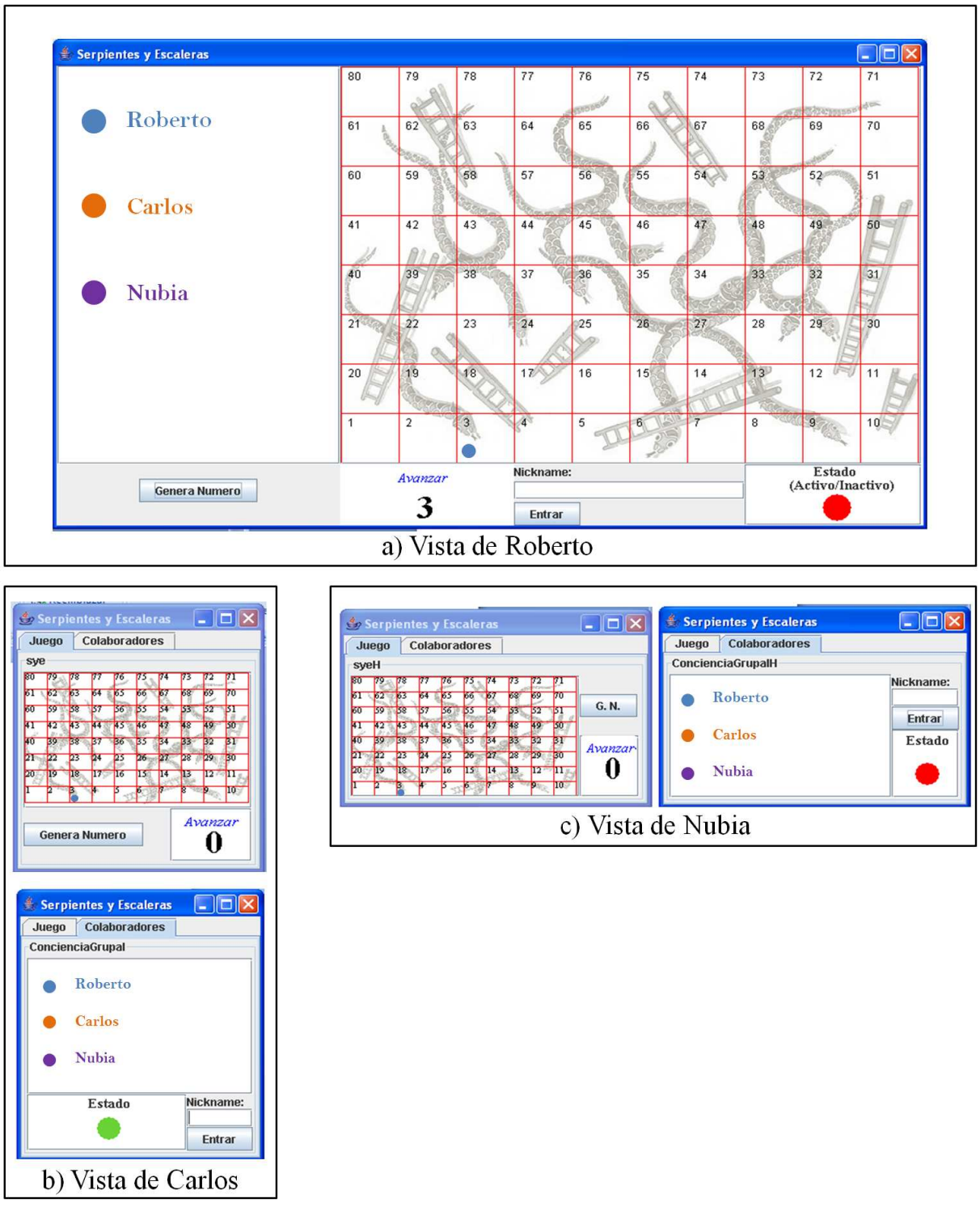


Figura 4.18: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 5$

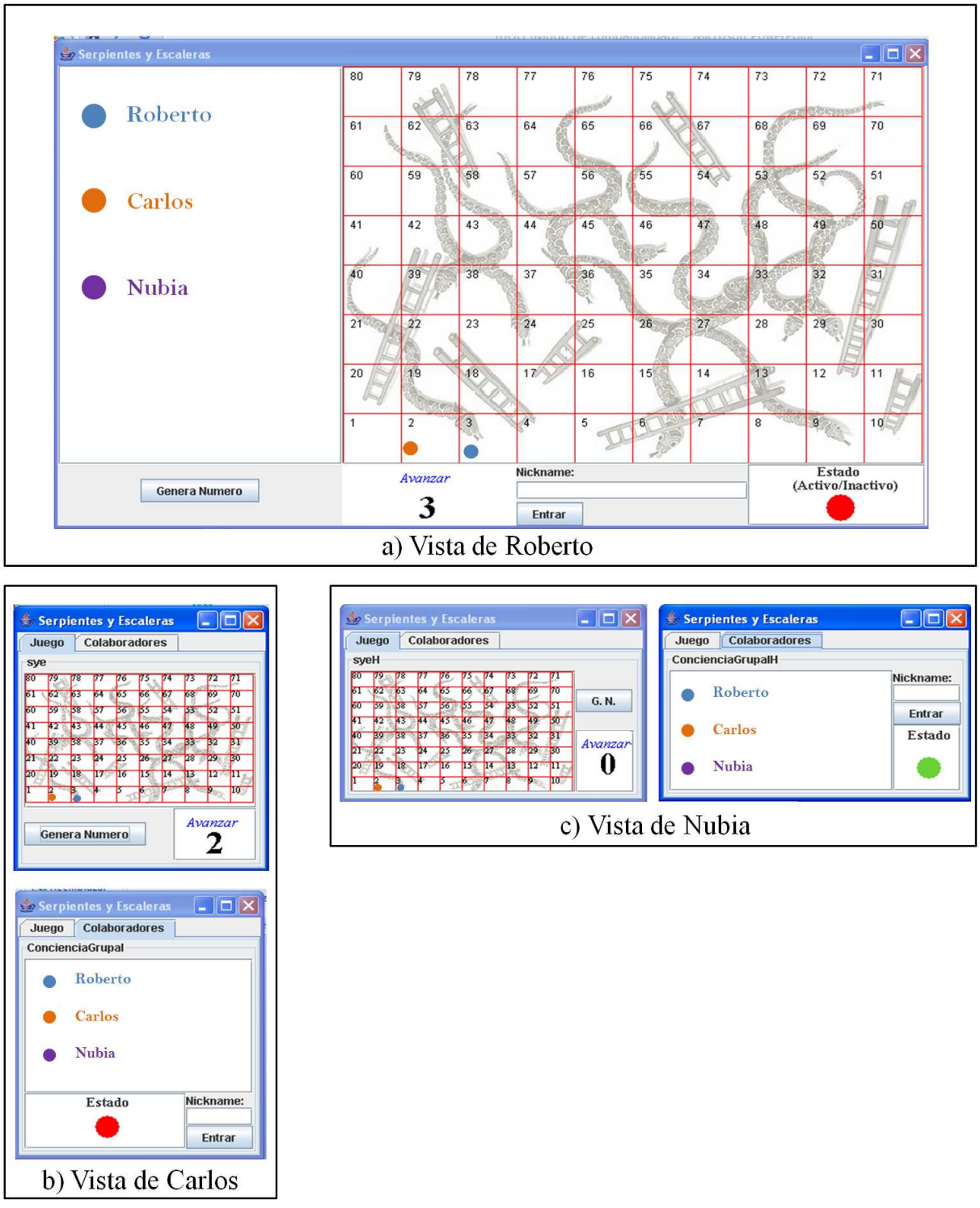


Figura 4.19: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 6$

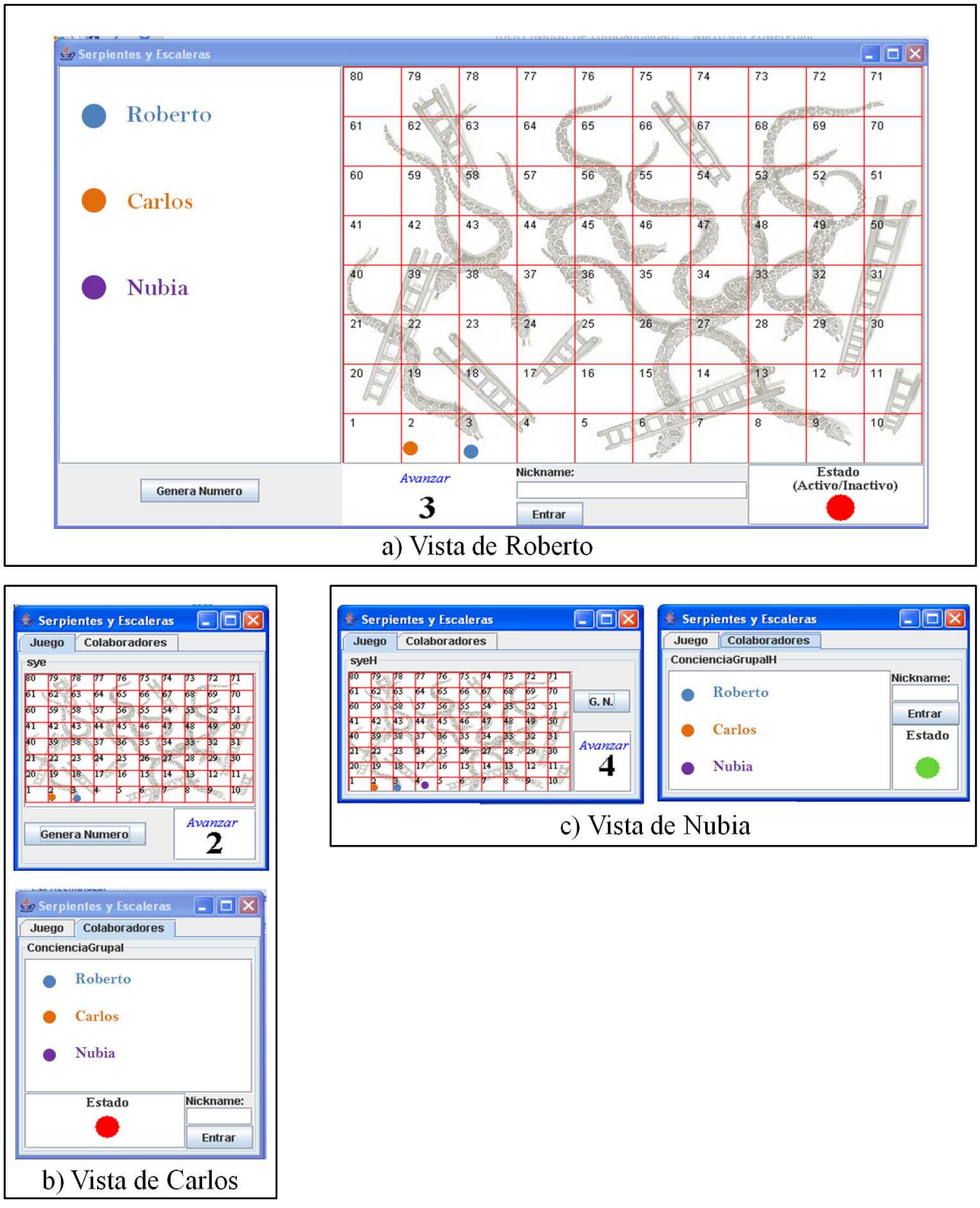


Figura 4.20: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 7$

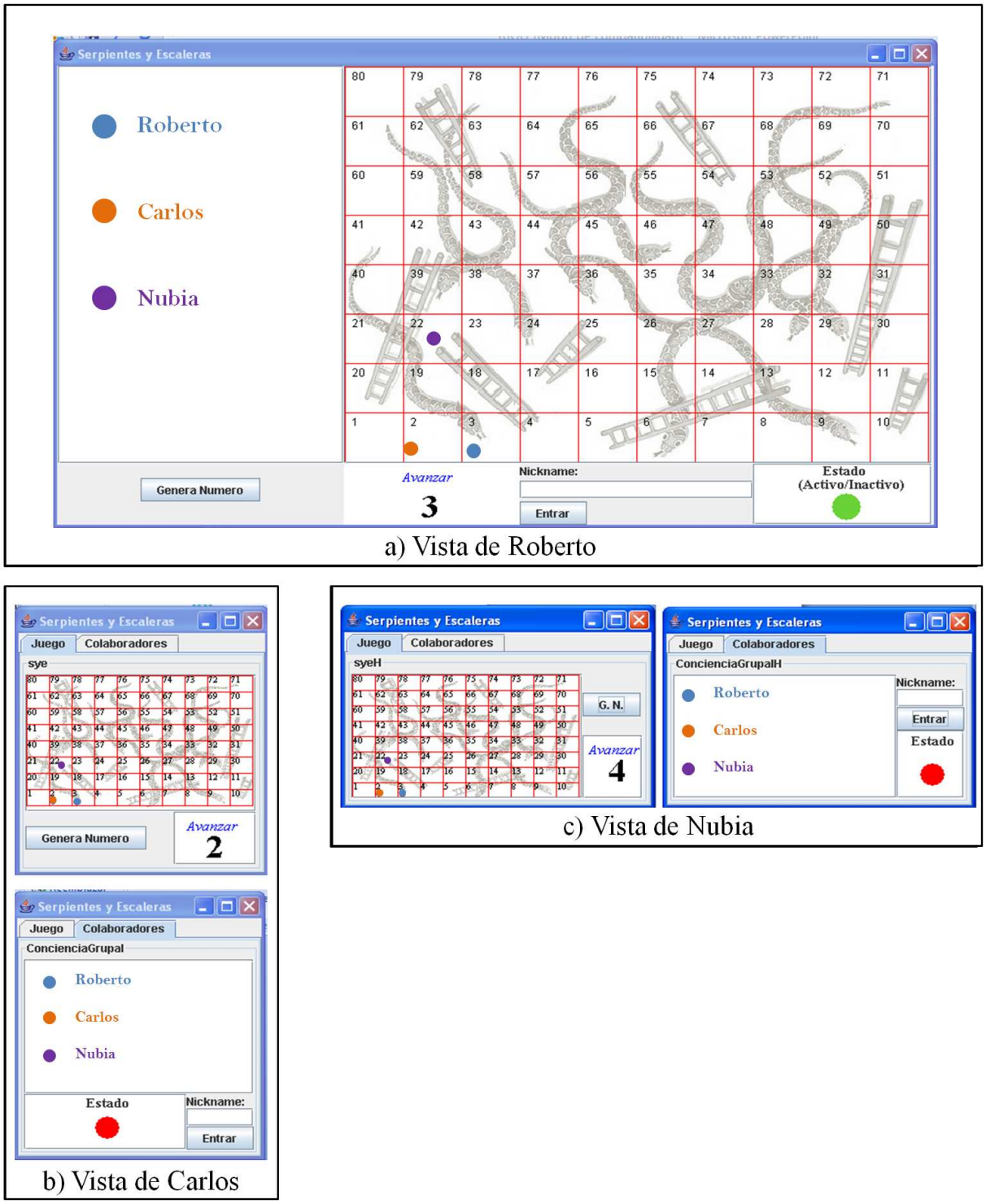


Figura 4.21: Interfaz de usuario plástica de la aplicación “Serpientes y Escaleras” en el tiempo $t + 8$

Capítulo 5

Conclusiones y trabajo futuro

El presente capítulo está estructurado en tres partes. En primer lugar, se recapitula la problemática abordada en esta tesis de maestría (cf. sección 5.1). Enseguida, se presentan las principales aportaciones y limitaciones de la solución propuesta para resolver el problema planteado (cf. sección 5.2). Finalmente, se describen algunas funcionalidades del mecanismo ReSAIC que pueden ser mejoradas y extendidas (cf. sección 5.3).

5.1 Resumen de la problemática

Este trabajo de tesis se inscribe en el dominio de investigación de dos áreas de las Ciencias de la Computación: 1) el Trabajo Cooperativo Asistido por Computadora (TCAC) y 2) la Interacción Hombre Máquina (IHM). El TCAC es un dominio multidisciplinario que estudia tanto los aspectos sociales de las actividades individuales y colectivas, como los aspectos tecnológicos de la información y de las comunicaciones, con el fin de asistir la colaboración entre personas físicamente distribuidas. Por otra parte, la Interacción Hombre Máquina (IHM), se relaciona con: 1) el diseño, la implementación y la evaluación de sistemas interactivos que están enfocados al uso humano y 2) el estudio de los principales fenómenos que rodean a dichos sistemas. En el ámbito de la IHM, la interfaz de usuario constituye el artefacto tecnológico de un sistema interactivo que posibilita una interacción amigable entre el usuario y el sistema.

Una interfaz mono-usuario provee componentes gráficos que permiten al usuario hacer uso de las funciones del sistema. Por el contrario, una interfaz de usuario colaborativa proporciona, además de los componentes ya mencionados, componentes gráficos adicionales para remplazar la información que se pierde cuando los miembros del grupo no interactúan cara a cara. En consecuencia, esta adición de componentes gráficos a las interfaces de usuario colaborativas implica una mayor dificultad en la administración del espacio de despliegue visual.

En la actualidad existe una gran variedad de dispositivos de cómputo, los cuales cuentan con diferentes capacidades de procesamiento, memoria, almacenamiento y comunicación. Con base en estas capacidades, se requiere que las interfaces de usuario de los sistemas interactivos se adapten a las características (e.g., tamaño de pantalla) de cada dispositivo. La propiedad de plasticidad permite satisfacer dicho requerimiento. Esta propiedad se ha comenzado a explorar en los sistemas mono-usuario. En el caso de los sistemas colaborativos, la propiedad de plasticidad no ha sido abordada plenamente, dejando un campo abierto a la investigación.

5.2 Conclusiones

El trabajo realizado en la presente tesis consistió en la adaptación de la interfaz de usuario de un sistema colaborativo a diferentes contextos de uso (específicamente a la plataforma). Para desempeñar esta adaptación, se desarrolló el mecanismo ReSAIC (Remodelación Semi-plástica Adaptativa para Interfaces Colaborativas), el cual proporciona a una aplicación colaborativa las siguientes características:

1. soporta un grupo de usuarios;
2. notifica a cada usuario su turno de participación en el momento que le corresponda; y
3. ofrecer conciencia de grupo.

Además, el mecanismo ReSAIC dota a la interfaz de usuario de una aplicación colaborativa de las siguientes propiedades de plasticidad:

1. adaptabilidad a diversos dispositivos de cómputo (e.g., PC, laptop, PDA, PocketPC); y
2. portabilidad entre diferentes sistemas operativos (e.g., Windows, Linux, Mac y Windows Mobile).

Con la finalidad de probar el desempeño de dicho mecanismo, se implementó la aplicación "Serpientes y Escaleras", la cual tiene las siguientes características:

1. a petición de un usuario genera un número aleatorio, el cual le indica cuantas casillas debe avanzar en el tablero del juego;
2. mediante un *click* del *mouse* permite que un usuario se posicione en una casilla del tablero; y
3. verifica que un usuario se posicione en la casilla que le corresponde, i.e., si el jugador intenta un movimiento incorrecto, la aplicación impide que se lleve a cabo dicho movimiento.

Para la implementación tanto del mecanismo ReSAIC como de la aplicación "Serpientes y Escaleras" se utilizó el lenguaje de programación Java. Una de las razones de utilizar dicho lenguaje se debe a que permite la portabilidad de las aplicaciones en diversos sistemas operativos (e.g., Windows, Linux y Mac). Sin embargo, con base en los objetivos planteados en la presente tesis, surgió la necesidad de permitir la adaptabilidad de aplicaciones colaborativas en diversos dispositivos de cómputo.

La máquina virtual de Java **Mysaifu** está diseñada para ejecutarse únicamente en dispositivos móviles con recursos restringidos (e.g., PDA y PocketPC). Sin embargo, Mysaifu se ajusta a los estándares de la tecnología J2SE (Java 2 Stardard Edition). Esta tecnología se utiliza en el desarrollo de aplicaciones para dispositivos de cómputo con suficientes recursos (e.g., PC y *laptop*).

Gracias a Mysaifu, una aplicación (desarrollada en Java) que se ejecuta en una PC puede ser ejecutada en una PDA. Con base en esta característica de Mysaifu, se procedió a diseñar e

implementar las interfaces de usuario de la aplicación "Serpientes y Escaleras" para los diversos dispositivos de cómputo (e.g., PC, laptop, PDA). El siguiente paso consistió en diseñar e implementar un módulo que realizará el proceso de adaptación plástica de estas interfaces de usuario a dichos dispositivos de cómputo. Finalmente, se diseñó e implementó un módulo que permite la comunicación entre los diversos dispositivos.

El soporte de colaboración se realizó por medio de sockets utilizando la tecnología de redes inalámbricas Wi-Fi. Se utilizó la arquitectura cliente-servidor, de acuerdo con la cual una PC desempeña la función de servidor y los dispositivos que utilizan los usuarios para interactar con la aplicación "Serpientes y Escaleras" fungen como clientes.

El mecanismo ReSAIC está compuesto tanto del módulo de adaptación plástica como del módulo de comunicación. Con base en las pruebas realizadas a la aplicación "Serpientes y Escaleras", se demuestra el correcto funcionamiento del mecanismo de remodelación semiplástica adaptativa para interfaces colaborativas. A partir de los resultados obtenidos, se concluye que el mecanismo ReSAIC podría ser reutilizado en otros dominios de aplicación, e.g., herramientas de análisis de problemas o soluciones y paneles de discusión virtual que cuenten con un moderador.

En la presente tesis de maestría, los esfuerzos se concentraron en el componente de adaptación plástica, el cual constituye el tema central de esta tesis de maestría. Por lo tanto, el mecanismo ReSAIC ofrece actualmente un entorno básico de colaboración, el cual maneja turnos de participación, pero no permite el ingreso de más usuarios una vez iniciada la fase de colaboración.

5.3 Trabajo futuro

El proceso de adaptación plástica implementado en la presente tesis solo se lleva a cabo una vez. i.e., detecta el dispositivo en que se ejecuta la aplicación "Serpientes y Escaleras" y, en respuesta, despliega la interfaz de usuario correspondiente. Sin embargo, existen dispositivos de cómputo (e.g., PDA, iPod y PocketPC) que al ser rotados giran la vista de sus aplicaciones. En tales dispositivos, es necesario que el proceso de adaptación plástica no solo se aplique una vez ya que, en función de dichos giros, es necesario desplegar una interaz de usuario adecuada. Para satisfacer el cuarto paso del proceso de adaptación plástica, i.e., transición entre estados (cf. sub-sección 2.1.2 del capítulo 2), se propone añadir, al módulo de adaptación plástica, un proceso de censado que verifique la posición del dispositivo de cómputo (i.e., vertical/horizontal), con la finalidad de mostrar la mejor interfaz de usuario en todo momento. El proceso de censado permitirá preservar la continuidad de interacción de la aplicación, que es una característica ineludible de la plasticidad.

El soporte de colaboración del mecanismo ReSAIC está implementado con base en la arquitectura de comunicación cliente-servidor. Se eligió dicha arquitectura porque las aplicaciones que se ejecutan en dispositivos con capacidades restringidas (e.g., PDA) se vuelven ineficientes para los usuarios cuando se utilizan otras arquitecturas de comunicación (e.g., *peer to peer*). Sin embargo, la arquitectura cliente-servidor tiene la desventaja de que si el servidor deja de funcionar, entonces se romperá la comunicación entre los dispositivos. Por lo tanto, se propone implementar estrategias para el manejo de este tipo de fallos, con la finalidad de mantener la consistencia de los datos compartidos.

En el párrafo anterior se explicaron las desventajas de la arquitectura cliente-servidor al ocu-

rrir una falla en el servidor. Sin embargo, una falla en el cliente también puede ocasionar graves desperfectos. Por ejemplo, si un usuario abandona la aplicación "Serpientes y Escaleras" sin que el juego haya concluido, entonces el servidor se bloquea debido a que se queda esperando la jugada de dicho usuario. Esta espera impide al servidor pasar el turno a otro usuario. Para resolver este problema, se propone diseñar una estrategia que verifique que todos los usuarios se encuentren activos en una aplicación. En caso de que algún usuario no esté activo, entonces se debe eliminar su *id* de la tabla de registros.

Las interfaces de usuario de la aplicación "Serpientes y Escaleras" están implementadas para soportar como máximo a cuatro usuarios. Por lo tanto, si se utiliza dicha aplicación en un aula escolar en donde existen muchos usuarios, entonces se deberán formar cuatro equipos para que cada uno represente a un usuario. Evidentemente, solo una persona del grupo interactuará directamente con la aplicación, lo que puede ocasionar ciertos disgustos entre los miembros de dicho grupo. Para resolver este problema se propone implementar mecanismos que permitan a todos los integrantes de un grupo participar activamente en la aplicación colaborativa.

Apéndice A

Instalaciones y configuraciones

A.1 Instalación de Mysaifu

Los pasos para instalar Mysaifu son los siguientes:

1. descargar Mysaifu[1] en una PC;
2. extraer los archivos que están comprimidos en un archivo zip;
3. copiar (vía Bluetooth o cualquier otra tecnología de comunicación) el archivo `jvm.Release.CAB` a la PDA; y
4. localizar el archivo `jvm.Release.CAB` en la PDA y hacer *click* sobre él para iniciar automáticamente la instalación.

A.2 Archivos JAR

Al desarrollar una aplicación en Java, normalmente se tienen varios archivos `.java`, los cuales al compilarlos generan archivos `.class`. Si se quiere ejecutar dicha aplicación, entonces se deben colocar todos los archivos `.class` en un mismo directorio. Java permite empaquetar toda esta estructura de archivos `.class` en un único archivo con extensión `.jar`[2]. La forma de crear un archivo `.jar` desde Ms-dos o *shell* de Unix es la siguiente:

```
$ jar cf archivo.jar archivo1.class archivo2.class...
```

La instrucción anterior incluirá en un archivo `.jar` todos los archivos `.class` que se le indique. Si la clase que contiene el `main` se encuentra en el archivo `prueba.java`, entonces la forma de ejecutar este archivo `.jar` es así:

[1] http://www2s.biglobe.ne.jp/~dat/java/project/jvm/download_en.html

[2] Un archivo `JAR` (por sus siglas en inglés, *Java ARchive*) es un tipo de archivo que permite ejecutar aplicaciones escritas en lenguaje Java

```
$ java -cp archivo.jar prueba
```

En la opción `-cp` (*classpath*) se indica el nombre del archivo `.jar` y luego el nombre de la clase que tiene el *main*.

A.2.1 Archivo de manifiesto

Si se quiere ejecutar una aplicación sin necesidad de utilizar la línea de comandos, i.e., únicamente dando doble *click* sobre el archivo `.jar`, entonces se debe crear un archivo especial llamado "archivo de manifiesto", el cual se incluye dentro del archivo `.jar`. Para la creación de dicho archivo primero se debe abrir el bloc de notas, donde se debe introducir la siguiente línea:

```
Main-Class:  prueba
```

La palabra `prueba` representa el nombre de la clase que contiene el *main*. Posteriormente se guarda el archivo de texto con cualquier nombre, e.g., `manifiesto.txt`, el cual se debe incluir en el archivo `.jar` de la siguiente manera:

```
$ jar cmf manifiesto.txt archivo.jar archivo1.class...
```

También se puede ejecutar una aplicación desde línea de comandos con la siguiente instrucción:

```
$ java -jar archivo.jar
```

La opción `-jar` indica que se debe buscar dentro del archivo `.jar` el "archivo de manifiesto" y ejecutar la clase que se indica allí. A continuación, se debe configurar Windows para que abra los archivos `.jar` con el comando `javaw`. Si se tiene instalado Java 5 no se requiere esta configuración. De lo contrario, se debe hacer *click* sobre los siguientes iconos: "Mi PC", "Herramientas", "Opciones de carpeta" y "Tipos de archivo". A continuación, aparece una lista en la cual hay que seleccionar la opción `JAR`. Enseguida se debe hacer *click* sobre los iconos "Opciones avanzadas", "Open", "Editar" y escribir lo siguiente:

```
"C:\Archivos de programa\Java\jre1.5.0_06\bin\javaw.exe" -jar "%1"
%*
```

Finalmente, al hacer doble *click* sobre el archivo `.jar` se podrá ejecutar la aplicación.

A.2.2 Inclusión de otros archivos en el archivo jar

Además del "archivo de manifiesto" y los archivos `.class`, se puede incluir otro tipo de archivos, e.g., imágenes .gif, .jpg, .png, etc, en el archivo `.jar`. Para que las imágenes puedan ser visualizadas en una aplicación, se utiliza la clase *Loader*, de la cual se crea una instancia (con la finalidad de emplear sus métodos) en un objeto de la clase donde se requiere procesar

dichas imágenes. La sintaxis es la siguiente:

```
ClassLoader classLoader = NombreClase.class.getClassLoader();
URL url= classLoader.getResource ("archivo.gif");
ImageIcon imagen = new ImageIcon (url);
```

A.3 Ejecución de la aplicación "Serpientes y Escaleras"

Todas las clases de Java que componen el mecanismo ReSAIC se encuentran dentro del archivo `mecanismoReSAIC.jar`. Para ejecutar la aplicación "Serpientes y Escaleras" en una PC basta con hacer doble *click* en dicho archivo. Sin embargo, en una PDA el proceso es un poco más largo. En primer lugar, se debe ejecutar la máquina virtual Mysaifu, la cual se localiza en el directorio `Programas` (cf. Figura A.1).

Figura A.1: Localización del icono de Mysaifu en la PDA

Posteriormente, se despliega una ventana como la que se muestra en la Figura A.2. En la opción "Type" de esta ventana, se debe seleccionar la opción "archiveJAR". Seguidamente, se tiene que dar *click* sobre el botón "Browse..." para localizar el archivo `mecanismoReSAIC.jar`. Finalmente, se debe dar *clik* sobre el botón "Execute".

A.4 Requerimientos de ejecución del mecanismo ReSAIC

Los requisitos mínimos de la PDA o PocketPC son:

- sistema operativo Windows Mobile 5.0

- pantalla táctil TFT de 2,8 pulgadas, 65.000 colores, 240 x 320 pixeles con luz LED
- procesador Marvell PXA270 a 520 MHz
- 128 MB de memoria SDRAM principal para ejecutar aplicaciones, 256 MB de flash ROM
- WLAN 802.11b/g con seguridad WPA2, Bluetooth 2.0 integrado con EDR (*Enhanced Data Rate*)

La aplicación “Serpientes y Escaleras” es compatible con PCs y laptops que cuenten con los sistemas operativos Windows, Linux o Mac.

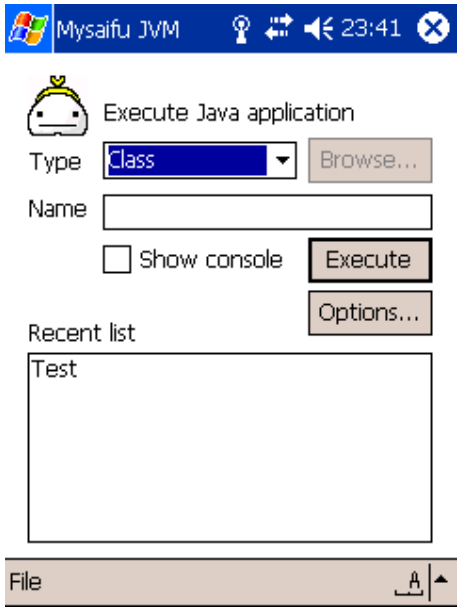


Figura A.2: Ventana principal de Mysaifu

Referencias

[Abowd et al., 1997] Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R., and Pinkerton, M. (1997). *Cyberguide: A Mobile Context-aware Tour Guide*. Baltzer/ACM Wireless Networks, 3(5), pp. 421-433, Kluwer Academic Publishers.

[Ardaiz et al., 2004] Ardaiz, O., de Cerio, L. D., Freitag, F., Gallardo, A., Marqués, J. M., Messeguer, R., Navarro, L., and Sanjeevan, K. (2004). *Sistemas Distribuidos y CSCL*. Revista Iberoamericana de Inteligencia Artificial, 8(24), pp. 13-20.

[Baldor, 1998] Baldor, A. (1998). *Aritmética*. Décima cuarta reimpresión, Ed. Publicaciones Cultural.

[Balme et al., 2005] Balme, L., Demeure, A., Calvary, G., and Coutaz, J. (2005). *Sedan-Bouillon: A Plastic Web Site*. Plastic Services for Mobile Devices (PSMD), Workshop held in conjunction with Interact'05, pp. 1-3, Rome, Italy.

[Bell et al., 2006] Bell, M., Chalmers, M., Barkhuus, L., Hall, M., Sherwood, S., Tennent, P., Brown, B., Rowland, D., Benford, S., Capra, M., and Hampshire, A. (2006). *Interweaving Mobile Games with Everyday Life*. Proceedings of the Conference on Human Factors in Computing Systems, ACM Press, pp. 417-426, Montreal, Canada.

[Bernstein et al., 1987] Bernstein, P., Goodman, N., and Hadzilacos, V. (1987). *Concurrency Control and Recovery in Database Systems*. Addison-Wesley.

[Berti and Paternò, 2005] Berti, S. and Paternò, F. (2005). *Migratory MultiModal Interfaces in MultiDevice Environments*. Proceedings of the 7th International Conference on Multimodal Interfaces, ACM Press, pp. 92-99, Torento, Italy.

[Bevan, 1999] Bevan, N. (1999). *Quality in Use: Meeting User Needs for Quality*. Journal of Systems and Software, 49(1), pp. 89-96, Elsevier Science Inc.

[Bevan and Macleod, 1994] Bevan, N. and Macleod, M. (1994). *Usability Measurement in Context*. Behaviour and Information Technology, 13(1-2), pp. 132-145.

[Brewster, 1998] Brewster, S. (1998). *The Design of Sonically-Enhanced Widgets*. Interacting with Computers, 11, pp. 211-235.

[Brewster and Cryer, 1999] Brewster, S. and Cryer, P. (1999). *Maximising Screen-Space on Mobile Computing Devices*. CHI '99 extended abstracts on Human factors in computing systems, ACM Press, pp. 224-225, Pittsburgh Pennsylvania.

[Brown et al., 2003] Brown, B., MacColl, I., Chalmers, M., Galani, A., Randell, C., and Steed, A. (April 2003). *Lessons from the Lighthouse: Collaboration in a Shared Mixed Reality System*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, pp. 577-584, Florida, USA.

[Calvary et al., 2004] Calvary, G., Coutaz, J., Daassi, O., Balme, L., Demeure, A., and Sottet, J. S. (2004). *Towards a New Generation of Widgets for Supporting Software Plasticity: The Comet*. Conference on Engineering for Human-Computer Interaction, Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems, In R. Bastide, P. A. Palanque and J. Roth (Eds.), icolcomlec, pp. 306-324, Hamburg Germany.

[Calvary et al., 2001b] Calvary, G., Coutaz, J., and Thevenin, D. (2001b). *A Unifying Reference Framework for the Development of Plastic User Interfaces*. Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction, Springer-Verlag, pp. 173-192, London, UK.

[Calvary et al., 2007] Calvary, G., Coutaz, J., and Thevenin, D. (2007). *Métamorphose des IHM et Plasticité*. Revue d'Interaction Homme-Machine (RIHM), 8(1), pp. 35-60.

[Calvary et al., 2001a] Calvary, G., Coutaz, J., and Thevenin, D. (September 2001a). *Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism*. Proceding of Joint Conf. on Human-Computer Interaction IHM-HCI 2001, Springer-Verlag, pp. 349-363, Lille, France.

[Calvary et al., 2002] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., and Vanderdonckt, J. (2002). *Plasticity of User Interfaces: A Revised Reference Framework*. Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design, INFOREC Publishing House Bucharest, pp. 127-134.

[Cheverst et al., 2000] Cheverst, K., Davies, N., Mitchell, K., Friday, A., and Efstratiou, C. (2000). *Developing a Context - Aware Electronic Tourist Guide: Some Issues and Experiences*. Proceeding of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, pp. 17-24, The Hague, The Netherlands.

[Ciavarella and Paternò, 2004] Ciavarella, C. and Paternò, F. (May 2004). *The Design of a Handheld, Location-Aware Guide for Indoor Environments*. Personal and Ubiquitous Computing, 8(2), Sprinter Verlag, pp. 82-81.

[Cook, 1990] Cook, W. R. (1990). *Object-Oriented Programming Versus Abstract Data Types*. Proceeding of the REX Workshop/School on the Foundations of Object-Oriented Languages (FOOL), 489, Springer Lecture Notes in Computer Science, pp. 151-178.

[Coutaz and Calvary, 2008] Coutaz, J. and Calvary, G. (2008). *HCI and Software Engineering: Designing for User Interface Plasticity*. The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications, pp. 1107-1125.

[Crease, 2001] Crease, M. (2001). *A Toolkit of Resource-Sensitive, Multimodal Widgets*. PhD Thesis, Department of Computing Science, University of Glasgow.

[Crease et al., 2000] Crease, M., P. Gray, P., and Brewster, S. (2000). *A Toolkit of Mechanism and Context Independent Widgets*. In: Design, Specification and Verification of Interactive Systems (Workshop 8, ICSE 2000), P. Palanque and F. Paterno (eds.), pp. 127-141.

[Danesh et al., 2001] Danesh, A., Inkpen, K., Lau, F., Shu, K., and Booth, K. (2001). *Geney: Designing a Collaborative Activity for the Palm Handheld Computer*. Proceedings of the ACM CHI 2001 Human Factors in Computing Systems Conference, ACM Press, pp. 388-395, Seattle, USA.

[Darrell et al., 2002] Darrell, T., Tollmar, K., Bentley, F., Checka, N., Morency, L., Rahimi, A., and Oh, A. (2002). *Face-Responsive Interfaces: From Direct Manipulation to Perceptive Presence*. Proceedings of the 4th international conference on Ubiquitous Computing, Springer-Verlag, pp. 135-151, Góteborg Sweden.

[Demeure et al., 2007] Demeure, A., Calvary, G., Coutaz, J., and Vanderdonckt, J. (2007). *The Comets Inspector: Towards Run Time Plasticity Control based on a Semantic Network*. Lecture Notes in Computer Science, 4385, Springer Heidelberg, pp. 324-338.

[Dini et al., 2007] Dini, R., Paternò, F., and Santoro, C. (2007). *An environment to support multi-user interaction and cooperation for improving museum visits through games*. Proceedings of the 9th international conference on Human computer interaction with mobile devices and services, ACM, pp. 515-521, New York USA.

[Dommel and Garcia-Luna-Aceves, 1997] Dommel, H. P. and Garcia-Luna-Aceves, J. J. (1997). *Floor Control for Multimedia Conferencing and Collaboration*. Multimedia Syst., 5(1), Springer-Verlag, pp. 23-38.

[Ellis and Gibbs, 1989] Ellis, C. A. and Gibbs, S. J. (1989). *Concurrency control in groupware systems*. Proceedings ACM SIGMOD International Conference on Management of Data, 18(2), ACM Press, pp. 399-407.

[Forum, 1998] Forum, W. (1998). *Wireless Application Protocol: Wireless Markup Language Specification*.

[García et al., 2007] García, M., Paderewski, P., and Hornos, M. J. (2007). *Procesos de adaptación para aplicaciones colaborativas*. Actas de Talleres de Ingeniería del Software y Bases de Datos, 1, pp. 1-8.

[González et al., 2007] González, J. L., Cabrera, M., and Gutiérrez, F. L. (2007). *Diseño de Videojuegos aplicados a la Educación Especial*. VIII Congreso Internacional de Interacción Persona-Ordenador (Interacción2007), Zaragoza, Spain.

[Gram and Cockton, 1996] Gram, C. and Cockton, G. (1996). *Design Principles for Interactive Software*. Chapman & Hall Publishers, London, UK.

[Greenberg and Bohnet, 1991] Greenberg, S. and Bohnet, R. (1991). *GroupSketch: A multi-user Sketchpad for Geographical and Distributed Small Groups*. Proceedings of Graphics Interjizce '91, pp. 207-215, June 5-7, Calgary Alberta, Canada.

[Greenberg and Marwood, 1994] Greenberg, S. and Marwood, D. (1994). *Real-Time Groupware as a Distributed System: Concurrency Control and its Effects on the Interface*. Proceedings of CSCW '94, the ACM Conference on Computer-Supported Cooperative Work, pp. 207-218, ACM Press, 22-26 octobre, Chapel Hill North Carolina, USA.

[Greenberg et al., 1992] Greenberg, S., Roseman, M., Webster, D., and Bohnet, R. (1992). *Human and Technical Factors of Distributed Group Drawing Tools*. Interacting with Computers, 4(1), pp. 364-392, December.

[Grinter et al., 2002] Grinter, R. E., Aoki, P. M., Hurst, A., Szymanski, M. H., Thornton, J. D., and Woodruff, A. (2002). *Sotto Voce: Exploring the Interplay of Conversation and Mobile Audio Spaces*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, pp. 431-438, Minneapolis Minnesota, USA.

[Grolaux et al., 2001] Grolaux, D., Roy, P. V., and Vanderdonckt, J. (2001). *QTk: A Mixed Model-Based Approach to Designing Executable User Interfaces*. Lecture Notes in Computer Science, 2254, Springer-Heidelberg, pp. 109-110.

[Grolaux et al., 2002] Grolaux, D., Roy, P. V., and Vanderdonckt, J. (2002). *FlexClock, a Plastic Clock Written in Oz with the QTk toolkit*. Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design, Costin Pribeanu and Jean Vanderdonckt (Eds), INFOREC Publishing House Bucharest, pp. 135-142.

[Kindberg et al., 2008] Kindberg, T., O'Neill, E., Bevan, C., Kostakos, V., Stanton Fraser, D., and Jay, T. (2008). *Measuring trust in Wi-Fi hotspots*. Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems, ACM Press, pp. 173-182, Florence, Italy.

[Laurillau and F. Paternò, 2004] Laurillau, Y. and F. Paternò, F. (September 2004). *Supporting Museum Co-visits Using Mobile Devices*. Proceedings Mobile Human-Computer Interaction, 3160, Springer Verlag, pp. 451-455.

[Moran et al., 1992] Moran, T., McCall, K., van Melle, B., Pedersen, E., and Halasz, F. (1992). *Design Principles for Sharing in Tivoli, a Whiteboard Meeting-Support Tool*. In Designing Groupware for Real Time Drawing, S. Greenberg, S. Hayne and R. Rada ed. McGraw Hill.

[Morikawa and Maesako, 1998] Morikawa, O. and Maesako, T. (1998). *HyperMirror: Toward Pleasanttouse Video Mediated Communication System*. Proceedings of Computer Supported Cooperative Work, ACM Press, pp. 149-158, Seattle Washington, USA.

[Oppermann and Specht, 2000] Oppermann, R. and Specht, M. (2000). *A Context-Sensitive Nomadic Exhibition Guide*. Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing, Springer Verlag, pp. 127-142, Bristol, UK.

[Phillips, 1999] Phillips, W. G. (1999). *Architecture for Synchronous Groupware*. Technical Report, Department of Computing and Information Science, pp. 11-24, Queen's University Kingston, Ontario, Canada.

[Rekimoto, 1997] Rekimoto, J. (1997). *Pick-and-drop: a direct manipulation technique for multiple computer environments*. Proceedings of the 10th Anual ACM Symposium on User Interface Software and Technology, ACM Press, pp. 31-39, Banff Alberta, Canada.

[Robertson et al., 1991] Robertson, G. G., Mackinlay, J. D., and Card, S. K. (1991). *Cone Trees: animated 3D visualizations of hierarchical information*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press, pp. 189-194, New York, NY.

[Roseman and Greenberg, 1996] Roseman, M. and Greenberg, S. (1996). *Building Real-Time Groupware with GroupKit, a Groupware Toolkit*. ACM Transactions Comput.-Hum. Interact. 3(1), pp. 66-106.

[Shen and Dewan, 1992] Shen, H. and Dewan, P. (1992). *Access control for collaborative environments*. Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work, ACM Press, pp. 51-58, Toronto Ontario, Canada.

[Stefik et al., 1987] Stefik, M., Bobrow, D. G., Foster, G., Lanning, S., and Tatar, D. (1987). *WYSIWIS Revised Early Experiences with Multiuser Interfaces*. ACM Transactions on Once Information Systems, 5(2), pp. 147-167, April.

[Tang, 1991] Tang, J. C. (1991). *Findings from Observational Studies of Collaborative Work*. International Journal on Man Machine Studies, 34(2), pp. 143-160, February.

[Thevenin, 2001] Thevenin, D. (2001). *L'adaptation en Interction Homme-Machine: le cas de la plasticité*. PhD thesis. Thèse de doctorat Informatique préparé au Laboratoire de Communication Langagière et Interaction Personne-Système (IMAG), Université Joseph Fourier.

[Thevenin and Coutaz, 1999] Thevenin, D. and Coutaz, J. (1999). *Plasticity of User Interfaces: Framework and Research Agenda*. Proceedings Interact '99, Edinburgh, Sasse, A., Johnson, C. Eds, IFIP IOS Press Publ., pp. 110-117.

[Vernier et al., 1999] Vernier, F., Lachenal, C., Nigay, L., and Coutaz, J. (November 1999). *Interface Augmentée Par Effet Miroir*. IHM'99 AFIHM Conference on Human-Machine Interface, Cepadues Publ., pp. 158-165, Montpellier, France.