Interfaces de Usuario Adaptativas Basadas en Modelos y Agentes Software

Tesis Doctoral

Víctor Manuel López Jaquero

Universidad de
Castilla-La Mancha

Julio 2005

Departamento de Sistemas Informáticos

UNIVERSIDAD DE
CASTILLA-LA MANCHA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS

INTERFACES DE USUARIO ADAPTATIVAS BASADAS EN MODELOS Y AGENTES SOFTWARE

VÍCTOR MANUEL LÓPEZ JAQUERO

DIRIGIDA POR

DR. D. PASCUAL GONZÁLEZ LÓPEZ
DR. D. ANTONIO FERNÁNDEZ CABALLERO

Albacete, Julio de 2005

RESUMEN DE LA TESIS

En la última década se ha realizado un gran esfuerzo en la investigación de métodos que permitan la inclusión del diseño de la interfaz de usuario dentro de un proceso de desarrollo basado en modelos, intentando obtener beneficios tales como la automatización de la generación de la interfaz de usuario, la generación de dicha interfaz para distintos dispositivos o lenguajes a partir de unos modelos comunes o la mejora de las propiedades de usabilidad del sistema.

Sin embargo, el avance tecnológico ha propiciado la aparición de dispositivos que suponen un cambio importante en la forma en que el usuario interactúa con los sistemas. El usuario ahora puede interactuar con el sistema en un coche, en la calle, desde un teléfono móvil, una PDA o el tradicional PC.

Este cambio en los hábitos en la interacción hombre máquina ha supuesto la necesidad de aportar soluciones que permitan diseñar interfaces de usuario capaces de funcionar bajo distintas plataformas y condiciones.

Aunque es posible llevar un desarrollo separado para cada familia de dispositivos, asumiendo el alto coste de desarrollo y mantenimiento que ello supone, es todavía más difícil si no imposible diseñar interfaces de usuario para cada una las situaciones en las que la interfaz de usuario puede ser potencialmente usada. Una solución más eficiente sería la generación de interfaces de usuario capaces de acomodarse a los distintos tipos de dispositivos, entornos de uso, e incluso tipos de usuarios de forma automática, aunque ello supone sin duda la modificación de los actuales métodos de desarrollo de interfaces de usuario.

Para afrontar este reto, dentro de este trabajo, se propone la ampliación de los métodos actuales basados en modelos para el diseño de interfaces de usuario con los mecanismos necesarios para el diseño de las capacidades de adaptación necesarias dentro de las distintas fases del desarrollo de una interfaz de usuario. Estas extensiones son recopiladas dentro de AB-UIDE *(Agent-Based User Interface Development Environment)*. El método permite la generación de interfaces de usuario capaces de

adaptarse a las distintas situaciones que potencialmente pueden surgir durante el proceso de interacción.

El método es apoyado por una arquitectura basada en un sistema multi-agente que permite proporcionar al usuario las capacidades de adaptación diseñadas usando el método AB-UIDE propuesto. Los agentes del sistema multi-agente colaboran para proporcionar inteligentemente al usuario las adaptaciones más adecuadas en cada situación que se presenta durante la interacción con la interfaz de usuario diseñada.

esta tesis está dedicada a mi gente:
aquellos que hacen camino conmigo.

AGRADECIMIENTOS

En el trecho de camino de mi vida recorrido durante el desarrollo de esta tesis, numerosas personas han quitado, y puesto, piedras en el camino. Aunque aquellos que las pusieron se van diluyendo en el olvido. Como según el dicho: "*no es bien nacido el que no es agradecido*", me gustaría agradecer, en estas breves líneas, su apoyo a algunas de las personas que han quitado obstáculos en el camino, o que me han ayudado a salvarlos.

Primeramente, me gustaría agradecer a mi familia su paciencia y apoyo desinteresado y constante durante mi vida, que me ha brindado la posibilidad de realizar una tesis doctoral: sin ellos esta tesis no sería una realidad.

En segundo lugar me vienen a la cabeza mis amigos, que con su compañía y ánimos han hecho todo este tiempo más llevadero. Mi más sincero agradecimiento para todos los "Parrockianos", "Roleros" y "Futbolineros" y demás que han amenizado los fines de semana que no estaba trabajando.

A continuación no puedo olvidarme de mis compañeros en el grupo de investigación LoUISE: especialmente de mis directores Pascual González y Antonio Fernández, y de mis compañeros José Antonio Gallud, José Eduardo Córcoles, María Lozano y Víctor Ruiz, pero muy especialmente de Francisco Montero y José Pascual Molina, grandes amigos además de compañeros, que han contribuido de manera activa cada día para que se llegara al final de esta tesis, y que han arrimado siempre el hombro cuando ha hecho falta.

No podrían faltar mis compañeros en el Departamento de Sistemas Informáticos de Albacete, que han convivido conmigo durante estos años, con especial mención a Gregorio Díaz, compañero de despacho y amigo, por contagiar su visión positiva de la vida, a Julia Flores y Elena Navarro por sus ánimos, a José Antonio Gámez, José Miguel Puerta y Antonio Garrido por iniciarme en el mundo de la investigación, para bien o para mal, y finalmente a José Luis Sánchez, Fernando Cuartero y Vicente López.

No quisiera olvidarme tampoco de los miembros del BCHI *(Belgian Laboratory of Computer-Human Interaction)* en Louvain-la-neuve (Bélgica), y especialmente de Jean Vanderdonckt y Quentin Limbourg, que me acogieron durante siete meses en su grupo, compartiendo sus ideas y conocimientos conmigo para mejorar esta tesis.

Agradecer también a mi amigo Alberto Martínez el tiempo y el esfuerzo que se ha tomado para el diseño de la portada que acompaña esta tesis.

Si lees estas líneas y me ayudaste a salvar obstáculos en el camino, y me he olvidado de ti en el momento de escribir estos agradecimientos, acepta mi más sincero agradecimiento y mis disculpas.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CAPÍTULO 1
OBJETIVOS Y PROBLEMÁTICA

"Todos los hombres estamos hechos del mismo barro, pero no del mismo molde."
(Proverbio mejicano)

1.1 Introducción

Durante los últimos 20 años el desarrollo del software, y especialmente de las aplicaciones que se le dan a dicho software, ha sufrido un cambio espectacular. La aparición de Internet, y especialmente de la Web como medio de transmisión de la información, ha supuesto un cambio drástico en el enfoque de las aplicaciones, y la aparición de nuevas maneras y contextos en los que utilizar la información. En los últimos años se ha producido otra revolución, de aspecto tecnológico, que de nuevo está cambiando muchos de los puntos de vista sobre la interacción que hasta ahora se habían dado por sentados. La aparición de nuevos dispositivos con distintas capacidades ha producido la aparición de igual manera de nuevas formas de interactuar, e incluso lo que es más importante si cabe, ha introducido nuevos colectivos de usuarios en el mundo de la interacción con características dispares. Así, de igual forma que nos encontramos usuarios expertos incluso con habilidades como programadores de aplicaciones, y con un extenso conocimiento del hardware subyacente, podemos encontrar neófitos sin ninguna experiencia en la utilización de aplicaciones software.

Estos cambios han producido la aparición de nuevos requisitos, tanto funcionales como no funcionales, pero especialmente no funcionales que deben satisfacer las nuevas necesidades y cualidades de los usuarios. La introducción dentro del mundo de la interacción de nuevos colectivos con un bajo grado de formación en la interacción con las interfaces de usuario tradicionales supone un gran reto cuando se persigue la creación de sistemas de calidad, cuyas capacidades de usabilidad [Nie93] se ajusten a lo que el usuario demanda.

La creación de sistemas de calidad exige la adecuación a estos nuevos requisitos de las aplicaciones, y especialmente la adaptación de la interfaz de usuario a los nuevos dispositivos y sus capacidades, distintos colectivos de usuarios y entornos de interación manteniendo la calidad de sistema, y más concretamente, la usabilidad del sistema.

1.2 Motivación

A continuación se describen los factores que han motivado el desarrollo de esta tesis, así como el marco en el cual se encuadra el trabajo realizado.

1.2.1 Calidad del software

El objetivo último de la Ingeniería del Software es construir software de calidad. Uno de los grandes escollos que nos encontramos en la construcción de dicho software de calidad es cómo definir qué es software de calidad, y aún más importante, cómo medir dicha calidad.

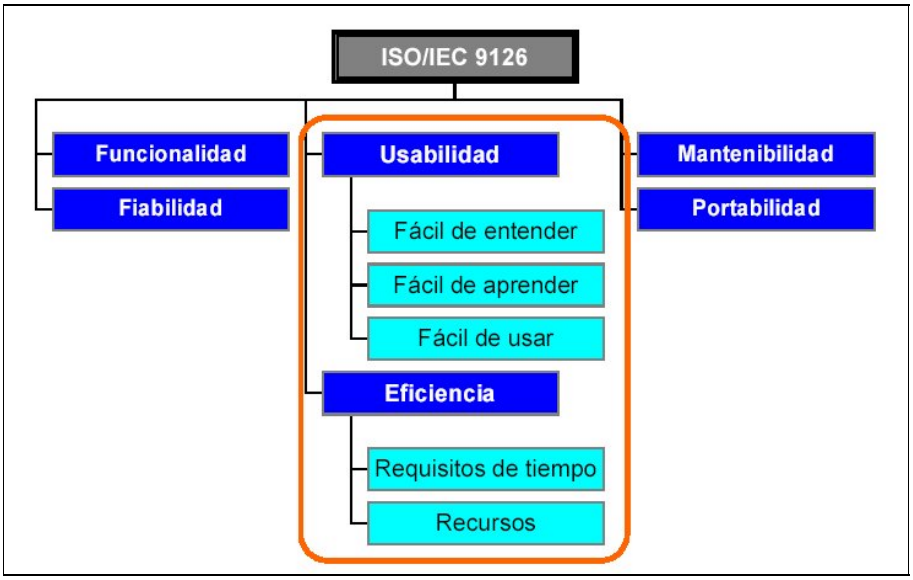


Figura 1.1 Parámetros de calidad del estándar ISO/IEC 9126.

El estándar ISO/IEC 9126 [ISO98] intenta aportar una definición de calidad en función de su funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad. Este trabajo surge de las reflexiones sobre cómo mejorar la calidad del software, y especialmente de las interfaces de usuario. Desde este enfoque se plantea este trabajo de investigación,

donde se persigue ahondar en los criterios y medios para mejorar la calidad de las interfaces de usuario, y en especial en los agentes de interfaz como medio para conseguir dicho propósito.

Dentro del objetivo de software de calidad podemos establecer dos vertientes: por un lado el software debe realizar las tareas para las que fue diseñado, es decir, debe cumplir los requisitos funcionales que le fueron encomendados, y por otra parte el sistema deber ser manejable, fácil de aprender y fácil de comprender. De la primera parte se encarga la Ingeniería del Software, mientras que la segunda parte es abordada principalmente por lo que se ha denominado Ingeniería de la Usabilidad [Nie93], dentro de la disciplina de Interacción Hombre-Máquina (HCI – *Human-Computer Interaction*). Dentro de los parámetros que caracterizan la calidad del software, son los que se refieren a esta segunda vertiente sobre los que pretendemos actuar de forma más directa. La figura 1.1 muestra los parámetros sobre los que influye directamente la interfaz de usuario, y sobre los que centraremos nuestros esfuerzos. El diseño de la interfaz de usuario debe permitir al usuario manejar la interfaz de usuario de forma cómoda y eficiente. El usuario debe ser capaz de entender cómo manejar la interfaz, debe ser capaz de recordar de una sesión a otra cómo se maneja la interfaz de usuario, y la interfaz de usuario debe ser fácil de usar. Aunque el diseño de la interfaz de usuario tiene un impacto directo en la usabilidad y eficiencia del sistema, es necesario destacar que la aplicación de un método de diseño de la interfaz de usuario apropiado también influye de forma importante tanto en la mantenibilidad como en la portabilidad de la aplicación.

1.2.2 Mejora de la usabilidad por adaptación

El incremento de la capacidad computacional y el contexto siempre creciente en el que se inserta dicha capacidad, sugieren que hay que buscar nuevas vías de interactuar con las computadoras que aumenten la facilidad de uso. Estas vías debieran estar más ajustadas a nuestras necesidades y habilidades. En los años más recientes una nueva conceptualización del fenómeno computacional ha dirigido el énfasis hacia la interacción en lugar del procedimiento [Weg97].

La búsqueda de maneras de interaccionar con las aplicaciones que sean más apropiadas para cada usuario o colectivo ha motivado el diseño de sistemas que sean capaces de adaptarse a cada usuario o colectivo, permitiendo proporcionar al usuario interfaces de usuario que sean fáciles

de entender, aprender y usar para el usuario a través de la utilización de representaciones y técnicas de interacción más compatibles con las características del usuario, y que por lo tanto permitan mejorar la productividad del usuario durante la utilización de la aplicación. Es decir, que mejore la calidad de la aplicación.

Sin embargo, para conseguir ofrecer al usuario interfaces de usuario más adecuadas a sus características, habilidades o preferencias, se necesitan también interfaces de usuario que mantengan un alto grado de usabilidad en cada una de las posibles plataformas donde el usuario puede hacer uso de la aplicación, y en cada uno de los entornos donde dicha interacción se pueda llevar a cabo. Aquellas aplicaciones capaces de adaptarse a los distintos usuarios, plataformas, y entornos donde se produce la interacción se denominan adaptativas.

El diseño de aplicaciones cuyas interfaces de usuario sean adaptativas requiere la modificación de las actuales técnicas para la creación de interfaces de usuario, introduciendo los mecanismos necesarios para el diseño de las capacidades de adaptación de la interfaz de usuario dentro del ciclo de desarrollo tradicional de las interfaces de usuario. Para ello será necesario enriquecer las actuales metodologías para el diseño de interfaces de usuario, y enlazar dichos diseños con los mecanismos que permitan la adaptación de la interfaz, y a su vez proporcionar una arquitectura de ejecución de las interfaces de usuario que permita el aprovechamiento de las capacidades de adaptación diseñadas.

1.3 Objetivos

Este trabajo tiene como objetivo principal la mejora de la calidad de las aplicaciones software, incidiendo sobre el diseño de la interfaz de usuario y la mejora de sus características de usabilidad. Para ello se busca el desarrollo de un método que proporcione los resortes necesarios para el diseño de interfaces de usuario adaptativas, que sean capaces de reaccionar ante los cambios en el contexto donde se está produciendo la interacción.

El diseño de interfaces de usuario adaptativas implica la especificación de una definición de la interfaz de usuario que sea dinámica, es decir, que permita su modificación en tiempo de ejecución, aunque siempre manteniendo las características de usabilidad. La interfaz seguirá sirviendo para la realización de las mismas tareas, pero la realización de dichas tareas se hará de distintas formas. A menudo a este tipo de interfaces de usuario

capaces de adaptarse, pero manteniendo su usabilidad, se las denomina *interfaces de usuario plásticas* [Cal01a].

La interacción del usuario con la aplicación se realiza dentro de un determinado contexto de uso, compuesto por el usuario que interacciona con la aplicación, la plataforma donde se ejecuta la aplicación, el entorno físico donde la interacción tiene lugar, y la tarea actual que está realizando el usuario. La adaptación de la interfaz de usuario a distintos contextos de uso implica tres pasos. Primero es necesario detectar los cambios en el contexto o en el usuario que provoquen la adaptación de la interfaz a las nuevas circunstancias. A continuación es necesaria una evaluación de la situación que lleve a la decisión de qué acciones se deben desencadenar para producir la adaptación. Finalmente, se ejecutarán las acciones que lleven a la adaptación de la interfaz. La detección de los cambios en el contexto, implica la existencia de mecanismos que permitan la toma de datos (sensores), mientras que la adaptación de la interfaz implica la existencia de mecanismos para su modificación (efectores). La toma de la decisión sobre qué acciones se desencadenan es sin duda un proceso de razonamiento basado en objetivos. Ello, nos ha llevado a la utilización del concepto de agente software como método para la creación de interfaces de usuario adaptativas, y más concretamente agentes de interfaz. Un agente de interfaz puede ser considerado como un robot cuyos sensores son las capacidades de entrada de la interfaz, y sus efectores las capacidades de salida de dicha interfaz. Los sensores permiten al agente percibir su entorno, mientras que los efectores son los medios que el agente posee para actuar sobre su entorno [Lie97].

Por otra parte, la modificación de la estructura de la interfaz de usuario en tiempo de ejecución para reflejar las adaptaciones practicadas sobre la interfaz de usuario necesita un marco donde la definición de dicha interfaz de usuario sea dinámica, y un mecanismo que permita realizar un proceso de introspección sobre el estado actual de la interfaz de usuario de forma que el sistema sea capaz de aplicar la adaptación más apropiada a cada caso.

Por lo tanto, como objetivos de este trabajo se plantea proporcionar un método de diseño capaz de permitir la especificación de interfaces de usuario adaptativas, y que por lo tanto permitan describir cómo se adapta la interfaz, qué dispara la adaptación de la interfaz de usuario, es decir, qué cambios del contexto producen qué adaptaciones, y qué condiciones

deben cumplir dichas adaptaciones para mantener la usabilidad de sistema tras su aplicación.

Por otra parte, la aplicación de los pasos descritos anteriormente para la aplicación de las adaptaciones diseñadas hace que sea necesaria una arquitectura de ejecución capaz de ejecutar los pasos descritos, y además hacerlo de forma adecuada. Es decir, el sistema debe ser capaz de percibir los cambios en el contexto de uso de la aplicación, a partir de dichos cambios elegir aquellas adaptaciones aplicables, seleccionar de forma "inteligente" aquellas adaptaciones que son actualmente más adecuadas, y finalmente aplicarlas manteniendo las usabilidad del sistema.

Estos son por lo tanto los objetivos que se persiguen dentro de este trabajo.

A continuación se describirá cómo se ha estructurado la memoria del trabajo de investigación realizado durante los últimos cinco años.

1.4 Estructura de la memoria del trabajo

Esta memoria esta estructurada de la siguiente manera:

- En el capítulo 1 el lector encontrará una introducción a la materia de la disertación, así como la motivación que ha provocado el trabajo y los objetivos del trabajo dadas las motivaciones expuestas.

- A continuación el capítulo 2 describe el estado actual de los trabajos relacionados con el tema tratado en esta tesis, y más concretamente el diseño de interfaces de usuario, especialmente el diseño de interfaces de usuario basado en modelos, y los sistemas multi-agente como modelo de especificación e implementación de interfaces de usuario inteligentes.

- El capítulo 3 describe la aproximación presentada en esta tesis de una forma general, esbozando cada uno de los componentes introducidos de forma que el lector se familiarice con los conceptos propuestos.

- A continuación el capítulo 4 describe en detalle el método de diseño de interfaces de usuario adaptativas propuesto,

desgranando cada una de las fases propuestas, y describiendo los modelos y notaciones utilizadas.

- El capítulo 5 muestra una arquitectura multi-agente para la ejecución de las interfaces de usuario adaptativas diseñadas siguiendo el método descrito en el capítulo 4, describiendo su diseño e implementación, así como una visión de cómo se integran los modelos diseñados a partir del método descrito en el capítulo 4 dentro de la arquitectura para ofrecer las capacidades de adaptación.

- En el capítulo 6 se describen dos casos de estudio donde se ilustran el método de diseño de interfaces de usuario adaptativas presentado en el capítulo 4, así como la integración en la arquitectura de ejecución de interfaces de usuario presentada en el capítulo 5.

- Finalmente, el capítulo 7 describe las conclusiones de esta tesis, las publicaciones realizadas, así como los trabajos futuros planteados.

CAPÍTULO 2
ESTADO DEL ARTE

"Lo peor es cuando has terminado
un capítulo y la máquina de escribir
no aplaude."
(Orson Welles)

2.1 Interfaces de usuario

La investigación en la interacción hombre-máquina ha tenido gran éxito y ha cambiado drásticamente el mundo de la computación.

La interacción entre el ser humano y la máquina se realiza a través de una interfaz de usuario que facilita intercomunicación traduciendo del lenguaje binario de la máquina al lenguaje humano y viceversa, pudiendo ésta adoptar múltiples modalidades (gráfica, textual, vocal, ...) e incluso combinarlas en las llamadas interfaces de usuario multimodales.

Las interfaces de usuario han evolucionado desde interfaces textuales donde cada orden debía ser escrita usando el teclado, hasta interfaces de usuario gráficas de gran complejidad.

Uno de los ejemplos más utilizados de interfaces gráficas son las llamadas WIMP *(Windows Icons Menus Pointer)*, como la interfaz gráfica de Microsoft Windows o los escritorios para el sistema operativo Linux KDE o GNOME, que están basados en la interfaz gráfica presentada por Apple para sus MacIntosh. Dicha interfaz fue basada en los trabajos en el Xerox Parc, que a su vez están basados en los trabajos iniciales realizados en el *Stanford Research Laboratory* y el MIT *(Massachusetts Institute of Technology)*. Un hecho comprobable es que prácticamente todo el software que utiliza interfaces gráficas usa *toolkits* o conceptos desarrollados originalmente en las universidades. Las investigaciones que llevan a las interfaces de usuario que tendremos en el futuro están siendo realizadas en la universidades y unos pocos laboratorios comerciales [Mye98].

2.1.1 Adaptación en las Interfaces de usuario

Existen distintas taxonomías que intentan clasificar la amplia variedad de posibles sistemas con algún grado de adaptación. Tradicionalmente se han considerado dos tipos de adaptación de la interfaz de usuario[Ben93]:

- Adaptabilidad: en este tipo de adaptaciones el usuario realiza la adaptación. Por lo tanto es el usuario el que explícitamente adapta la interfaz de usuario para que se ajuste a sus gustos y características. Un ejemplo típico de este tipo de adaptación es la configuración del aspecto del escritorio en gestores de ventanas como el de Microsoft Windows, o KDE y GNOME en el sistema operativo Linux. Estos gestores de ventanas permiten al usuario cambiar los colores, fuentes, el fondo del escritorio o el comportamiento de algunos de sus componentes (véase la figura 2.1).

Figura 2.1 Personalización del escritorio: un ejemplo de IU adaptable.

- Adaptividad: cuando se da este tipo de adaptación, el sistema es el actor responsable de realizar las acciones necesarias para realizar la adaptación. Un ejemplo de este tipo de adaptación es cuando durante la escritura de un documento en un procesador de texto,

como por ejemplo Microsoft Word, la aplicación detecta un error gramatical y automáticamente lo marca o incluso lo corrige.

Sin embargo, dentro del concepto de adaptividad existe un amplio rango de combinaciones en las que los actores inmersos en la interacción (normalmente el sistema y el usuario) pueden tomar la iniciativa en las distintas etapas necesarias para la realización de una adaptación. De esta forma podríamos encontrarnos con que una adaptación no se realiza de forma automática sino semiautomática. Las etapas en las que los distintos actores pueden tomar la iniciativa son [Die93]:

- **Iniciativa**: uno de los actores involucrados en la interacción sugiere su intención de realizar una adaptación. Los actores principales en este caso suelen ser el usuario o el sistema.

- **Propuestas**: si se detecta la necesidad de adaptación, será necesario proponer posibles adaptaciones que puedan ser aplicables dado el contexto de uso actual, para las necesidades detectadas. Una posible clasificación de los tipos de propuestas que se pueden dar sería [Cal01a]:

    - Sugerir un cambio a otra plataforma y otra configuración del entorno (por ejemplo, en un cliente de correo electrónico, cuando el usuario indique su intención de alejarse de su PC el sistema puede sugerir migrar el estado actual del cliente de correo a una plataforma móvil, como puede ser una PDA).

    - Sugerir el cambio a otro código ejecutable (por ejemplo, cuando el código actual de la aplicación no se pueda adaptar a los cambios que se han producido en el contexto).

    - Sugerir la ejecución de determinadas tareas (por ejemplo en un sistema de ayuda sensible al contexto).

    - Adaptar la interfaz de usuario manteniendo el mismo código ejecutable (por ejemplo, sería posible ocultar información no relevante para la tarea actual del usuario manteniendo el mismo código ejecutable).

- **Decisión**: durante la fase anterior se sugieren una serie de adaptaciones plausibles. Sin embargo, normalmente no será posible la aplicación de todas las adaptaciones propuestas, sino que habrá que decidir cuáles son las mejores adaptaciones dada la situación actual. Para decidir qué adaptaciones deben ser aplicados dentro de esta tesis se propone la evaluación de cada una de las posibles adaptaciones aplicables de acuerdo a una serie de criterios de usabilidad, midiendo la repercusión que la aplicación de cada una de las posibles adaptaciones tendría en ellos (véase la sección 5.2.6).

- **Ejecución**: finalmente, la adaptación o adaptaciones elegidas serán ejecutadas. Un factor importante cuando se realiza cualquier tipo de modificación a la interfaz de usuario sobre la que el usuario está actualmente interactuando es cómo se debe realizar la transición desde la interfaz de usuario original a la adaptada. Antes de la ejecución de una adaptación se suele ejecutar un prólogo para preparar la interfaz de usuario para la aplicación de la adaptación. Por ejemplo, si la adaptación incluye cambiar de un código a otro, la función de prólogo debería almacenar el estado actual de la aplicación, de forma que pueda ser reanudado tras la adaptación. De igual manera, una función de epílogo puede ejecutarse tras la adaptación para restaurar el estado del sistema. Siguiendo con el ejemplo anterior, el epílogo se encargaría de restaurar el estado de la aplicación y de reanudar la ejecución de la aplicación.

2.1.1.1 Adaptividad vs. Adaptabilidad

Tal y como se ha discutido anteriormente, en un sistema adaptable, el usuario es el responsable de la realización de cualquier adaptación a la interfaz de usuario, mientras que los sistemas adaptivos es el propio sistema el actor a cargo de la realización de las adaptaciones.

Sin embargo, es también posible combinar ambas aproximaciones cooperando para reducir las desventajas que cada una de dichas aproximaciones presentan. El principal problema que las interfaces de usuario adaptativas presentan es la sensación de pérdida de control que algunos usuarios experimentan a lo largo del proceso de adaptación. Esta sensación de pérdida de control puede acentuarse por las imprecisiones

del sistema [Fis93] a la hora de la realización de las adaptaciones (por ejemplo, detecciones de adaptación incorrectas o la elección de adaptación incorrecta entre las posibles adaptaciones aplicables). Por otra parte, existen indicadores que dejan entrever que los usuarios rara vez personalizan la interfaz de usuario [Mac91], y que incluso cuando lo hacen, no parece que lo hagan de forma muy eficiente.

Un ejemplo claro es la falta de personalización que se da en los usuarios de Microsoft Word. Aunque la utilización de las macros y estilos adecuados podría realmente aumentar su productividad, los usuarios no se detienen para aprender cómo se puede personalizar la aplicación. En estos casos, el proceso de adaptación podría aprender de la interacción y personalizar e indicar al usuario cómo usar las características personalizadas. En aquellos casos en que el usuario sea capaz de personalizar la aplicación de forma efectiva, no sería necesaria la aplicación de técnicas adaptativas.

Son por lo tanto, adaptabilidad y adaptividad técnicas complementarias, y no opuestas, capaces de mejorar la experiencia general del usuario actuando de forma colaborativa.

2.1.1.2 Descripción de la adaptividad

La adaptividad puede ser caracterizada usando los siguientes cuatro factores [Kar95]:

- **Constituyentes**: son los elementos de la interfaz de usuario que pueden ser adaptados (contenidos, modalidad, navegación, técnica de interacción, ...).

- **Determinantes**: son los factores que guían la adaptividad (características del usuario, tareas, entorno). En nuestro caso los determinantes son aquellas características del contexto de uso que pueden producir la ejecución de un proceso de adaptación.

- **Objetivos**: son los objetivos del proceso de adaptación (minimizar el número de errores, optimizar la eficiencia, facilidad de uso, mostrar al usuario que desea ver, acelera el uso, considerar la experiencia del usuario, etc). Los objetivos de la adaptación son diseñados dentro de la propuesta de esta tesis a través del diagrama de compromiso de usabilidad (véase el apartado 4.2.2.3).

- **Reglas**: guían la adaptación de los constituyentes de acuerdo a los cambios en los determinantes.

A través de la caracterización de la adaptividad se puede conseguir que la adaptividad sea computable, y por tanto incorporable en una interfaz de usuario. El modelado apropiado de esta caracterización conduce a una integración del proceso de diseño de la adaptación de una interfaz de usuario dentro de un ciclo de vida habitual de diseño de interfaces de usuario. Una descripción detallada de la caracterización de la adaptividad en la propuesta de esta tesis puede encontrarse en el apartado 3.4.1.

2.1.2 Desarrollo de interfaces de usuario

Hoy en día la facilidad de uso [Nie93] se ha convertido en uno de los factores más críticos a la hora de decantarse por un sistema u otro. El tiempo es precioso, y el usuario no quiere leer manuales, sino que lo que quiere es dedicar el tiempo simplemente a realizar las tareas, y no a cómo aprender a manejar un sistema. Las demandas del usuario han cambiado drásticamente. El usuario lo que espera actualmente de un sistema es que pueda sentarse ante su ordenador y comenzar a trabajar sin sentir ningún tipo de frustración por no saber cómo realizar una tarea.

Los requisitos que el usuario impone a la interfaz de usuario de los sistemas software actualmente hacen especialmente importante la participación de especialistas en Interacción Hombre-Máquina en el proceso de desarrollo. Un experimento desarrollado por Bailey [Bai93] mostró que los expertos en HCI creaban interfaces con un número menor de errores y que permitían que el usuario ejecutara sus tareas de forma más rápida que en las interfaces creadas por programadores. Esto demuestra que el diseño de la interacción hombre-máquina no es una cuestión de suerte o sentido común, y que la experiencia en la utilización de sistemas informáticos no es suficiente para diseñar una buena interfaz, sino que es necesario un entrenamiento específico en HCI.

2.1.2.1 Complejidad del diseño de interfaces de usuario

Se ha hablado mucho sobre la dificultad del diseño de las interfaces de usuario, pero, ¿qué hace que sea difícil diseñar una buena interfaz? Brad Myers [Mye93] presenta una serie de cuestiones que intentan justificar dicha dificultad.

El primer punto que aborda Myers es la necesidad de conocer al usuario. Esta tarea se intenta abordar desde el punto de vista del análisis de tareas En la práctica este análisis de tareas es muy complejo. La dificultad de este análisis de tareas es ilustrada con gran acierto por Don Normal de Apple cuando afirma: "*Mi experiencia es que las especificaciones iniciales son normalmente erróneas, ambiguas o incompletas. En parte, porque son desarrolladas por gente que no entiende los problemas que afrontan los usuarios… Peor, los usuarios pueden no saber lo que quieren, así que tenerlos dentro del equipo de desarrollo no es la solución, porque un verdadero entendimiento de una herramienta sólo se consigue con el uso, en parte porque una nueva herramienta cambia el sistema, y de esta manera cambia tanto las necesidades como los requisitos*" [correo electrónico].

El segundo punto es que la creciente complejidad de las aplicaciones plantea serias dificultades en el diseño de interfaces de usuario. Los teléfonos fijos que tenemos en casa son sencillos de manejar, tienen una cantidad de funcionalidades disponibles muy reducidas. Sin embargo, hoy en día las funcionalidades que ofrecen los teléfonos móviles crecen día a día. Esto ocurre de forma análoga en las aplicaciones. Microsoft Word contiene más de 300 órdenes distintas, y AutoCAD más de 1000. Es imposible crear una interfaz para este tipo de aplicaciones que sea tan fácil de usar y aprender como una interfaz para una aplicación que sólo tenga un número pequeño de funciones disponibles.

Teniendo en cuenta los puntos anteriores se plantea la necesidad de un diseño iterativo, donde se creen prototipos de la interfaz que son probados por los usuarios finales de la aplicación y rediseñados repetidamente. Sin embargo, esta aproximación presenta algunos problemas, donde el más grave seguramente sea que el refinamiento sobre un diseño originalmente deficiente conduce a diseños de interfaces de usuario finales de peor calidad que una interfaz diseñada originalmente de forma correcta [Bai93].

A las dificultades presentadas sobre el diseño de la interfaz hay que añadir la dificultad de su implementación. El porcentaje de código dedicado al manejo de la interfaz hoy en día supera el 50% del total. Las razones para la dificultad a la hora de implementar una interfaz son las mismas que presentan las aplicaciones típicas de mayor complejidad: multiproceso, robustez y requisitos de tiempo real. El multiproceso es necesario, pues casi todos los sistemas de interfaces gráficas incluyen un sistema de

eventos que atienden las peticiones de los distintos dispositivos de entrada. El diseñador de interfaces de usuario se encuentra, por lo tanto, con los problemas habituales del multiproceso. Por otro lado, las interfaces de usuario son especialmente exigentes en cuanto a sus requisitos de robustez. Esto se debe a que cuando se diseña el cuerpo de la aplicación se puede controlar para que sólo funcione con ciertos parámetros. Sin embargo, en el caso de las interfaces de usuario deben ser capaces de responder a cualquier entrada y seguir funcionando, además de mostrar un mensaje de error que sea útil en caso de que ocurra algún problema, de forma que el usuario pueda reparar el error y continuar. Otro aspecto que complica la implementación de las interfaces de usuario son los requisitos de tiempo real de las interfaces de usuario. Todas las actualizaciones deben ser mostradas en tiempo real, y cualquier tipo de retraso o desfase en la actualización produce una grave degradación de la calidad de la interfaz.

Ante los problemas planteados en el diseño e implementación de las interfaces de usuario se están proponiendo nuevas ideas que modifican la visión que actualmente tenemos de las interfaces de usuario. El modelo de interfaz gráfica actual WIMP *(Windows, Icons, Menus and Pointer)* no es suficiente para algunas aplicaciones, la utilización de la metáfora del escritorio y el ratón es demasiado limitada en su capacidad de representación teniendo en cuenta la cantidad de barras de herramientas y ventanas que las aplicaciones complejas necesitan hoy en día. Una de las posibles alternativas para paliar este problema sería utilizar pantallas con amplio campo de visión o añadir la tercera dimensión a los interfaces gráficos [Mol02]. Otras tendencias apuntan hacia interfaces adaptativas que sean capaces de amoldarse a los distintos sistemas donde sean mostrados.

2.1.3 Desarrollo de interfaces de usuario adaptativas

El desarrollo de interfaces de usuario adaptativas desempeña dos papeles principales: por una parte puede permitir la creación de una interfaz de usuario única capaz de adaptarse a las distintas plataformas donde puede ejecutarse. Sin duda, el diseño de una interfaz de usuario para cada una de las plataformas donde potencialmente puede ser ejecutada la aplicación no es el camino apropiado hacia el éxito, ya que conduce a altos costes de desarrollo y mantenibilidad, y a la aparición de inconsistencias entre las distintas versiones. El otro papel que desempeñan las interfaces de usuario adaptativas es suministrar al usuario una interfaz que acomode, por

ejemplo, su aspecto, o la técnica de interacción necesaria, a la tarea que se está realizando en cada momento. Un ejemplo de este tipo de adaptación es la reducción de la cantidad de información que el usuario debe procesar para realizar cada tarea, mostrándole al usuario en cada momento tan solo la información estrictamente necesaria para la realización de la tarea.

2.1.4 Desarrollo de interfaces de usuario inteligentes

El desarrollo de interfaces de usuario adaptativas plantea grandes retos dentro del proceso de creación de una interfaz de usuario. Entre ellos se pueden destacar la detección de la necesidad de adaptación de la interfaz de usuario dado el contexto de uso actual (especialmente cuando se trata de adaptación al usuario) y la decisión sobre cuáles son las mejores adaptaciones a aplicar.

Estos retos plantean la necesidad de introducir mecanismos dentro de la interfaz de usuario que permitan una toma de decisiones con un cierto grado de inteligencia, para evitar que decisiones erróneas degraden la usabilidad del sistema.

La introducción de modelos de razonamiento para la toma de decisiones relacionadas con la adaptación de la interfaz de usuario dentro del motor de ejecución de la interfaz, fuerza al diseño de dicho modelo de razonamiento y sus constructores, y por lo tanto al enriquecimiento de los métodos de diseño de interfaces de usuario habituales con los resortes necesarios para la especificación de los constructores que describan cómo se debe realizar la toma de decisiones durante le proceso de adaptación.

En el apartado 2.3 puede encontrarse una descripción detallada del modelo de razonamiento planteado dentro de esta tesis, así como la motivación que ha llevado a su elección.

2.1.5 Evolución del desarrollo de interfaces de usuario

A lo largo del tiempo, la creciente complejidad de las interfaces de usuario para el manejo de unas aplicaciones a su vez más y más complejas, ha ido obligando a la utilización de herramientas para el diseño de interfaces de usuario cada vez con un mayor nivel de abstracción

Por otra parte, la creciente heterogeneidad de las plataformas destino de los desarrollos ha propiciado de igual forma un aumento en el nivel de abstracción de los desarrollos de las interfaces, en busca de mecanismos

de diseño capaces de permitir que un único diseño pueda ser generado para distintas plataformas destino de forma automática o semiautomática.

El desarrollo de interfaces de usuario ha evolucionado desde la programación de interfaces de usuario usando lenguajes de propósito general hasta las actuales aproximaciones basadas en modelos [Pat99], donde se persigue un aumento del nivel de abstracción de la especificación de la interfaz de usuario, siguiendo las líneas definidas dentro del campo de las arquitecturas guiadas por modelos (MDA – *Model Driven Architectures*)[1].

Los trabajos de esta tesis se hallan encuadrados dentro de este tipo de aproximaciones basadas en modelos, cuya descripción detallada es presentada a continuación.

2.2 Diseño de interfaces de usuario basado en modelos

El desarrollo de interfaces de usuario basado en modelos (MB-UIDE – *Model-Based User Interface Development Environment)* consiste en la especificación de la interfaz de usuario utilizando modelos declarativos que describen las distintas facetas y artefactos involucrados en el desarrollo de una interfaz de usuario. La creación de los modelos necesarios suele realizarse mediante herramientas visuales donde el usuario hace uso de una notación gráfica que permite la especificación de los distintos modelos de una manera sencilla. Las aproximaciones basadas en modelos persiguen aumentar el nivel de abstracción usado en el diseño de la interfaz de usuario, dejando los detalles de implementación a generadores de código, y permitiendo una generación total o parcial de la interfaz de usuario de forma sencilla cuando los requisitos cambian. De igual manera, dentro de dichas aproximaciones también se persigue la portabilidad de las interfaces de usuario, de forma que un mismo diseño declarativo pueda ser convertido en código ejecutable para distintas plataformas o lenguajes sin necesidad de un rediseño de la interfaz.

La arquitectura general dentro del diseño de interfaces de usuario basado en modelos puede observarse en la figura 2.2 [Sch96]. El desarrollador de la interfaz de usuario aplica un proceso de diseño utilizando una herramienta de modelado en un entorno interactivo. El entorno

[1] http://www.omg.org/mda/

interactivo de modelado crea y modifica los modelos que representan el conocimiento que se tiene de la interfaz de usuario, además dicho entorno hace uso durante el proceso de diseño de una base de conocimiento donde se recopila la experiencia adquirida por los desarrolladores. Dentro de las bases de conocimiento podemos encontrar guías de estilo [Smi86][Shn92][IBM92], heurísticas y patrones [Mon02][Mon03] que servirán para la transformación de los modelos en código.

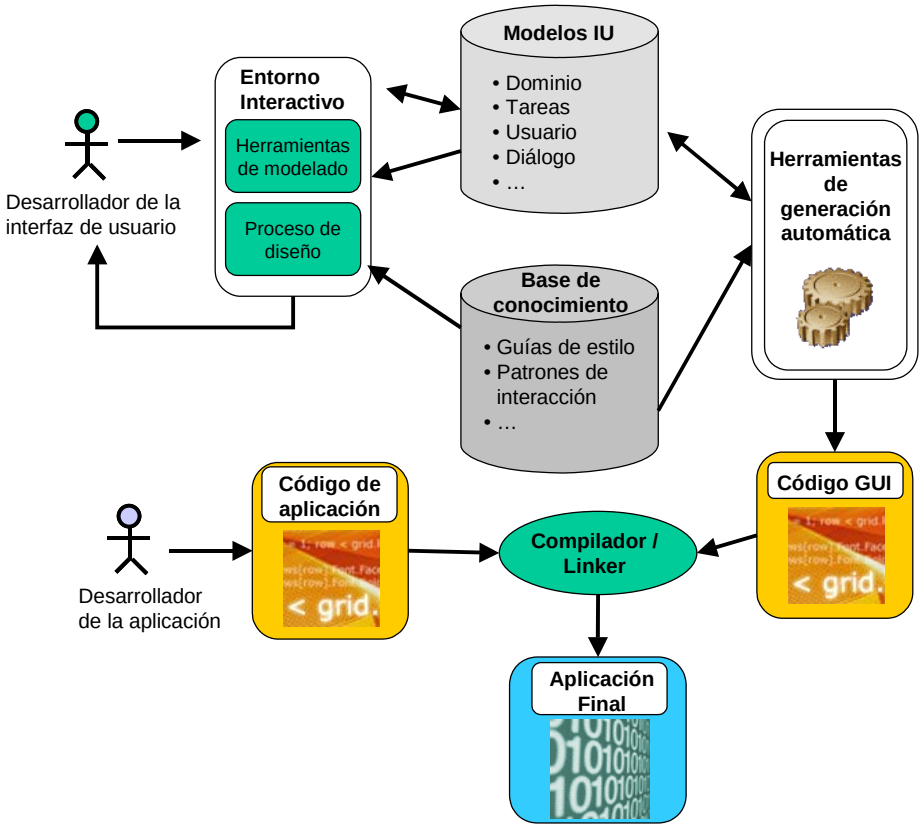


Figura 2.2 Arquitectura general de un entorno de desarrollo basado en modelos.

Un motor de generación automática de código generará el código de la interfaz de usuario usando para ello los modelos creados por el desarrollador de la interfaz de usuario y la experiencia recopilada sobre el diseño de interfaces de usuario para generar una interfaz de usuario usable. Como se puede ver en la figura en esta aproximación existe una separación entre el desarrollo de la parte funcional de la aplicación y su interfaz de usuario. Cada parte es desarrollada por separado para más

tarde unirlas para generar la aplicación final. Mientras que la interfaz de usuario es creada por el desarrollador de la interfaz de usuario, el código de la aplicación será creado por el desarrollador de la aplicación.

La utilización de este tipo de aproximaciones presenta una serie de ventajas [Pin00]:

- Permiten una descripción más abstracta de la interfaz de usuario que los métodos de descripción de interfaces de usuario más tradicionales.

- Permiten diseñar e implementar las interfaces de usuario de forma sistemática, ya que facilitan:

    a) El modelado de la interfaz de usuario en distintos niveles de abstracción.
    b) Refinar incrementalmente los modelos.
    c) Reutilizar las especificaciones de los modelos de usuario.

- Proporcionan la infraestructura necesaria para la automatización de parte de las tareas de diseño y generación de la interfaz de usuario.

Por otra parte, estas aproximaciones también presentan actualmente ciertos problemas por resolver:

- La complejidad de los modelos provoca que no sea habitualmente fácil aprender su funcionamiento, aunque se espera que el desarrollo de herramientas de diseño visuales reduzca la complejidad en gran medida.

- La integración de la parte funcional de la aplicación y de su interfaz de usuario todavía no está totalmente resuelta.

- No existe consenso sobre cuáles son los modelos más adecuados para modelar una interfaz de usuario. Ni siquiera lo existe sobre cuáles son los aspectos de la interfaz de usuario que deben ser modelados.

2.2.1 Los modelos en MB-UIDE

Actualmente no existe un estándar que defina cuáles son los modelos que debería tener un entorno de desarrollo basado en modelos, aunque sí que existen una serie de modelos comunes que aparecen en prácticamente todas las aproximaciones, como son los modelos de tareas, dominio, usuario, diálogo y presentación. La falta de consenso en la adopción de un conjunto estándar de modelos es debida en gran medida a las distintas disciplinas de los grupos de investigación involucrados en la definición de los MB-UIDE. De esta forma encontramos grupos más afines a la HCI [Pue97][Van93], otros más afines a la Ingeniería del Software [Loz00][Pas97], y finalmente otros grupos que proceden del mundo hipermedial [Gom01][Koc01].

2.2.1.1 El modelo de tareas

El modelo de tareas expresa cuáles son las tareas que va a realizar el usuario de la aplicación a través de la interfaz de usuario. Las tareas se descomponen en acciones atómicas que representan los pasos necesarios para alcanzar los objetivos de la tarea. Dentro de los datos capturados en este modelo también se recogen los requisitos no funcionales de las tareas, como son por ejemplo los requisitos de tiempo de respuesta. En la figura 2.3 se ejemplifican los conceptos de tarea y subtarea para un cuadro de diálogo para abrir un fichero.

Para la especificación del modelo de tareas se han utilizado distintas aproximaciones, entre las que destacan los métodos textuales basados en el análisis cognitivo como GOMS *(Goals*, *Operators*, *Methods*, *Selection rules)*[Car83][New85] o los métodos basados en formalismos como ConcurTaskTrees [Pat99] presentado por Paternò.

Habitualmente la captura de los requisitos de las distintas tareas que se realizarán con la interfaz de usuario suele realizarse utilizando diagramas de casos de uso [Boo99].

2.2.1.2 El modelo de dominio

El modelo de dominio incluye una visión de los objetos sobre los que actúan las tareas capturadas en el modelo de tareas. La especificación del modelo de dominio va muy ligada a las especificaciones realizadas dentro del modelador del dominio de la parte funcional. Esto puede producir una duplicidad de información con el consecuente peligro de aparición de incoherencias. En la figura 2.3 se muestran algunos de los objetos del

dominio que sería necesario especificar para modelar un típico cuadro de diálogo para abrir un fichero.

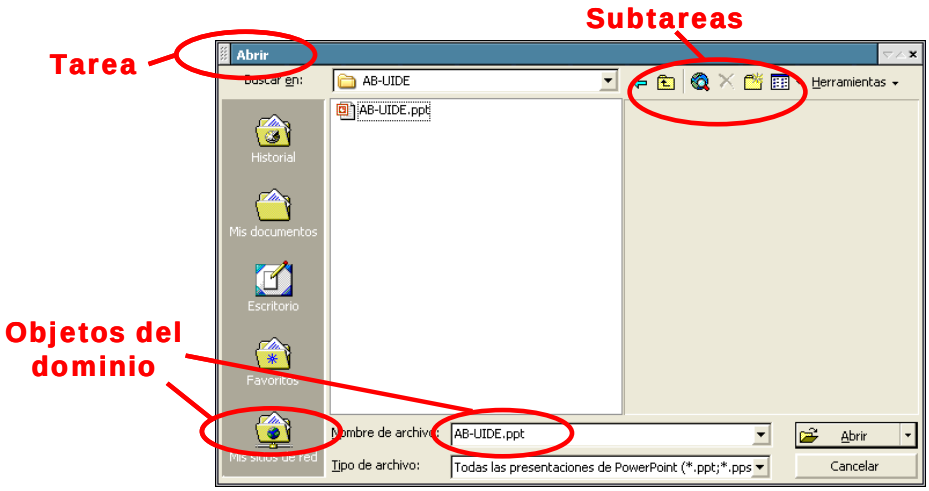


Figura 2.3 Ejemplos de tareas, subtareas y objetos del dominio.

Algunas aproximaciones utilizan para la representación del modelo de dominio diagramas entidad/relación [Che76], como por ejemplo *GENIUS* [Jan93], mientras que otras utilizan técnicas de orientación a objetos (diagramas de clases principalmente), por ejemplo *MECANO* [Pue96] o *IDEAS* [Loz00]. Sin embargo no existe un método estándar para la representación del modelo de dominio.

2.2.1.3 El modelo de usuario

El modelo de usuario captura las características y requisitos individuales de cada usuario o grupo de usuarios [Fis00]. Habitualmente cada uno de los tipos de usuarios que interactuarán con la aplicación es denominado *rol* (papel). El objetivo de este modelo es ofrecer una interfaz de usuario que se ajuste a las características y requisitos de cada usuario. La adaptación de la interfaz de usuario al usuario puede realizarse en tiempo de diseño o en tiempo de ejecución. En tiempo de diseño se puede definir, entre otras muchas cosas, cuales serán las tareas disponibles para cada tipo de usuario. Por otra parte la adaptación en tiempo de ejecución requiere un modelo de usuario más completo [Hor98].

Hasta la fecha los modelos de usuario introducidos dentro del desarrollo de interfaces de usuario basado en modelos han sido bastante reducidos, e insuficientes para la adaptación de la interfaz de usuario en tiempo de ejecución.

Las características del usuario contenidas en el modelo de usuario se suelen clasificar en dependientes de la aplicación e independientes de la aplicación [Hol90]. Las características independientes de la aplicación podrán ser reutilizadas de una aplicación a otra para el mismo usuario, mientras que las dependientes de la aplicación deberán ser elaboradas para cada nueva aplicación.

2.2.1.4 El modelo de diálogo

El modelo de diálogo describe las posibles conversaciones entre la interfaz de usuario y el usuario. Representa cuando el usuario puede introducir datos, seleccionar o cuando se muestran los datos. En general representa una secuencia de entradas y salidas.

Para la representación del modelo de diálogo se utilizan tanto técnicas textuales como visuales. Entre estas técnicas las más utilizadas son las distintas variaciones de los diagramas de transición, las redes de Petri [Bas90] o los diagramas de secuencia o estados [Har87][Loz00] de UML.

2.2.1.5 El modelo de presentación

El modelo de presentación contiene una descripción de la interfaz de usuario final con la que el usuario interactuará. Este modelo contiene los componentes que contiene la interfaz de usuario, su disposición y su apariencia.

En algunos entornos existen dos modelos de presentación, por una parte se construye un modelo abstracto, el cual describe la interfaz de usuario en función de objetos abstractos de interacción (AIO – *Abstract Interaction Object)* [Van93], y por otro lado se construye un modelo de presentación concreto, que se conformará con objetos concretos de interacción (CIO – *Concrete Interaction Object)*. El conjunto de objetos concretos de interacción para una plataforma no tiene por qué coincidir con los objetos de interacción concretos de otra plataforma. Esta separación en nivel abstracto y concreto del modelo de presentación permite una generación de la interfaz de usuario para distintas plataformas a partir de una misma descripción abstracta de la interfaz, donde será necesario seleccionar los objetos concretos de interacción correspondientes a cada objeto abstracto de interacción a partir de heurísticas, guías de estilo o patrones de interacción. Por otra parte, a partir de la información recogida en el modelo de usuario se pueden también crear distintas presentaciones

concretas de la interfaz de usuario dependiendo de las características y habilidades del usuario que está utilizando la aplicación en cada momento.

2.2.2 Aproximaciones basadas en modelos para el diseño de interfaces de usuario

A lo largo de la última década, distintas aproximaciones han sido presentadas para el diseño de interfaces de usuario basado en modelos, donde se soporta en mayor o menor medida la generación automática de la interfaz de usuario. La mayoría de estas herramientas proceden del ámbito académico y en general sólo presentan prototipos de las herramientas CASE para soportar los métodos de diseño y especificación propuestos, aunque ya podemos encontrar algunos ejemplo de herramientas comerciales basadas en modelos como WebRatio[2], VisualWADE[3] u Olivanova[4].

A continuación se describen algunos de los métodos basados en modelos más destacados para la generación de interfaces de usuario, entre los que se encuentran: TRIDENT, OVID, UIDE, JANUS, CTT, MOBI-D, Wisdom, IDEAS, Just-UI, OO-H, UMLi y UWE.

2.2.2.1 TRIDENT

TRIDENT *(Tools foR an Interactive Development ENvironmenT)* [Bod93][Bod94][Bod95] propone un entorno metodológico para el desarrollo de aplicaciones altamente interactivas. Esta aproximación permite la generación de interfaces de usuario de forma automática o semiautomática.

El modelo de tareas en TRIDENT: ACG

Dentro de este método se tratan tanto aspectos de presentación, como es la distribución de los componentes de la interfaz de usuario dentro de la pantalla, como los aspectos de la derivación de la interfaz de usuario a partir de modelos de tareas. Dentro de TRIDENT se utiliza el modelo entidad/relación enriquecido, así como grafos de encadenamiento de actividades (ACG - *Activity Chaining Graph*).

[2] http://www.webratio.com
[3] http://www.visualwade.com
[4] http://www.care-t.com

La figura 2.4 muestra un ejemplo de ACG donde se especifica la identificación de un cliente. Para ello se utilizan tres tipos de enlaces: los enlaces AND que representan conjunción de información, los enlaces OR que representan disyunción de información y los enlaces XOR que representan disyunción de información, pero con exclusión mutua. En la figura, cada rectángulo representa una función terminal identificada en el análisis de tareas. Cada trozo de papel (Calle, Ciudad, Nombre, etc) representa información proveniente del modelo Entidad/Relación o de la información de la aplicación. Los que están situados a la izquierda de las funciones serán datos de entrada, mientras que los situados a la derecha serán datos de salida. La información dibujada utilizando líneas continuas representa información externa, mientras que la información dibujada utilizando líneas punteadas representa información interna que no es visible por el usuario y que es producida por la aplicación (mensajes del sistema, puntos de decisión, etc).

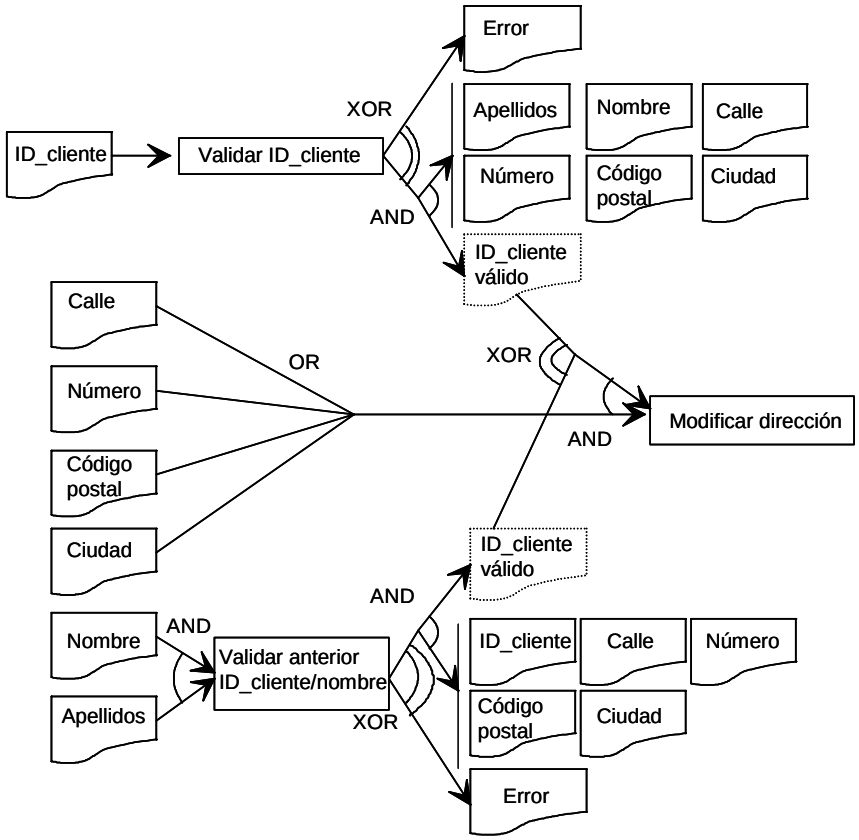


Figura 2.4 Ejemplo de especificación de un modelo de tareas con ACG.

En el ejemplo de la figura se especifica que para validar un cliente con un número que lo identifica, el sistema necesita el "ID_Cliente". La función "Validar ID_Cliente" buscará el cliente, y si no lo encuentra generará un mensaje de error. En caso de encontrar el cliente, la función generará la siguiente información: "ID_Cliente válido", que es información interna y que permitirá disparar la ejecución de "Modificar dirección", y la información externa (nombre, apellidos, ciudad, calle, número y código postal).

El modelo de presentación en TRIDENT

El diseño de la interfaz de usuario en TRIDENT se fundamenta en los conceptos de AIO (objeto abstracto de interacción), CIO (objeto concreto de interacción) o widgets, PU *(Presentation Unit* - unidades de presentación) y LW *(Logical Window* - ventanas lógicas).

- **AIO**: los objetos abstractos de interacción permiten la descripción de la interfaz de usuario de una forma abstracta independiente de la plataforma.

- **CIO**: los objetos concretos de interacción serán la cristalización de los objetos abstractos de interacción en una plataforma concreta. Por lo tanto, estos objetos concretos de interacción sí que serán totalmente dependientes de la plataforma donde se esté mostrando la interfaz de usuario.

- **PU**: las unidades de presentación representan la interacción dentro de una subtarea. La unidad de presentación puede estar compuesta por varias ventanas, que no tienen por qué ser mostradas simultáneamente. Sin embargo, siempre una ventana básica marca el inicio de la unidad de presentación, y el resto de las ventanas pertenecientes a esa unidad de presentación estarán encadenas a partir de la ventana básica.

- **LW**: las unidades de presentación están compuestas por ventanas lógicas, y sirven como contenedores para los objetos abstractos de interacción en el nivel abstracto, y para los objetos concretos de interacción en el nivel concreto.

Las relaciones entre estos conceptos se encuentran ilustradas en la figura 2.5, donde un objeto de interacción puede ser tanto abstracto como concreto. Cada unidad de presentación será presentada usando una serie de ventanas, las cuales incluirán un conjunto de objetos de interacción sencillos o compuestos. Los objetos de interacción compuestos son aquellos compuestos por más de un objeto de interacción sencillo.

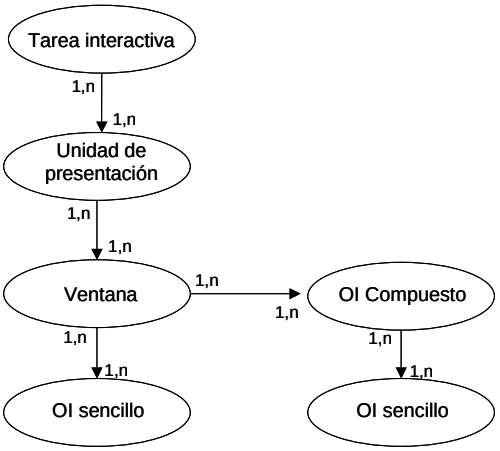


Figura 2.5 Relación entre los distintos objetos que componen la presentación.

Siguiendo con el ejemplo anterior, tras definir el modelo de tareas usando AGC, se definen las unidades de presentación (PU). En la figura 2.6 se pueden ver la especificación de las unidades de presentación para el ejemplo de la figura 2.4.

La arquitectura de las aplicaciones en TRIDENT

La arquitectura propuesta se basa en tres tipos de elementos:

- La clase objetos de control *(Control Objects* – CO) es una clase genérica que se puede descomponer en objetos de control de distintos tipos para manejar los diálogos, asegurando la correspondencia entre las estructuras de datos de la aplicación y su representación en la presentación. Cada CO mantiene su parte del comportamiento del diálogo y sus correspondencias. Este comportamiento se describe en forma de un guión (*script*) escrito en forma de reglas.

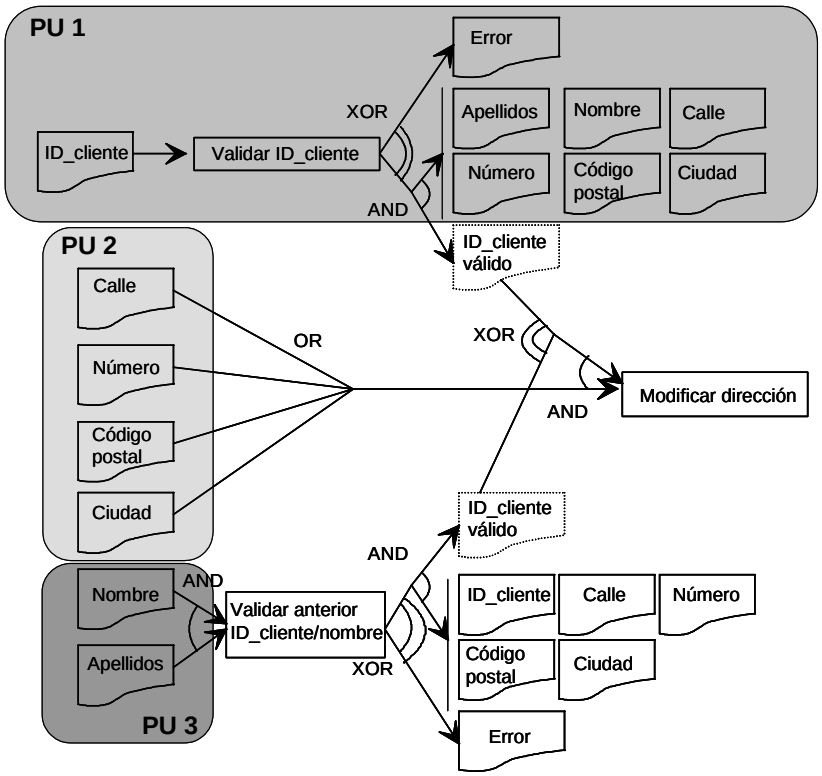


Figura 2.6 Unidades de presentación para el ACG especificado en la figura 2.4.

- La clase de objetos de aplicación *(Application Objects* – AO) es una clase cuyos elementos representan las funciones de la aplicación. Es importante asegurar que las funciones necesarias están incluidas en la arquitectura, ya que los requisitos funcionales juegan un papel determinante en la asignación de la semántica a las tareas interactivas y sus subtareas.

- La clase de objetos de interacción *(Interaction Objects* – IO) es una clase genérica que contiene dos clases de objetos concretos de interacción (CIO):

  (i). CIOs para entrada/salida de información.
  (ii). CIOs implicados estrictamente en el diálogo (botones, etc).

La estructura genérica de la arquitectura puede verse en la figura 2.7. En dicha figura se observa como los objetos de control (CO) realizan la comunicación entre los objetos de aplicación (AO9, que representan la

funcionalidad, y los objetos de interacción (IO) que sirven para realizar la presentación y la manipulación de la interfaz.

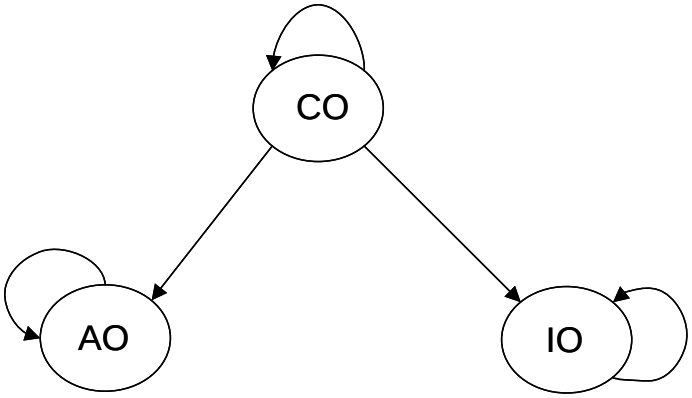


Figura 2.7 Arquitectura genérica de las aplicaciones en TRIDENT.

Dentro de TRIDENT encontramos una estructura similar a la propuesta en PAC *(Presentation*, *Abstraction*, *Control)* [Cou87] para la separación de los distintos aspectos de las aplicaciones. En TRIDENT los objetos de aplicación son equivalentes al concepto de Abstracción en PAC, los objetos de interacción son equivalentes al concepto de Presentación en PAC, y finalmente los objetos de control son equivalentes al concepto de Control en PAC.

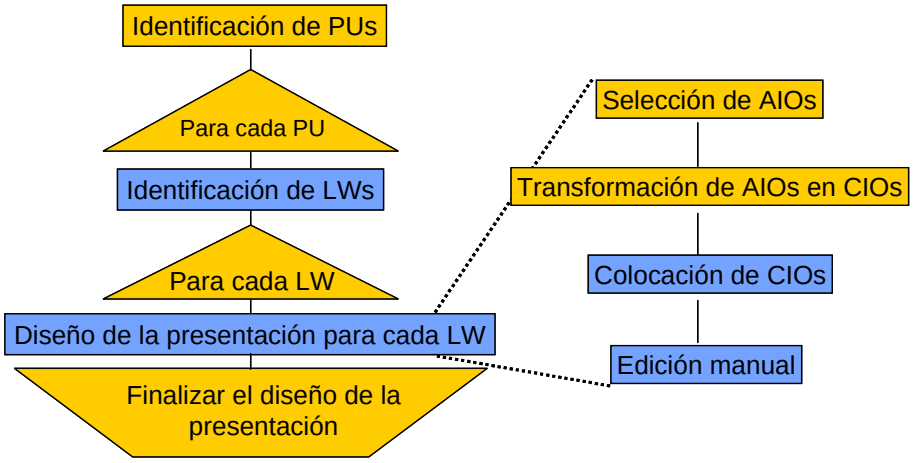


Figura 2.8 Proceso de desarrollo usando la herramienta SEGUIA.

Todo el proceso de diseño en TRIDENT es soportado por la herramienta SEGUIA *(System Export for Generating a User Interface Automatically)* [Van03] que permite la generación de la interfaz de usuario a partir de los modelos basados en TRIDENT. SEGUIA contiene además un sistema experto que selecciona mediante reglas y heurísticas la mejor traducción de AIO a CIO en el proceso de generación. La figura 2.8 muestra las distintas fases dentro del diseño de interfaz de usuario basado en TRIDENT usando SEGUIA.

2.2.2.2 OVID

La metodología OVID *(Object, View and Interaction Design)* [Rob98] es un conjunto de técnicas para el diseño de interfaces de usuario orientadas a objetos desarrolladas por IBM. La metodología se centra en tres elementos del diseño de la interfaz de usuario: los *objetos* que el usuario percibe, las *vistas* que se proporcionan de esos objetos, y las *interacciones* que el usuario tiene con los objetos.

El proceso básico de esta metodología es el siguiente:

1. Se genera un conjunto inicial de objetos examinando el análisis de tareas.

2. Se definen las vistas para permitir al usuario ver los aspectos correspondientes de cada objeto.

3. Se describen las tareas en términos de los nuevos objetos y vistas.

4. Se describen en detalle las interacciones del usuario con los objetos.

En la figura 2.9 se puede observar este ciclo de desarrollo.

Para diseñar los objetos, las vistas y las interacciones, OVID toma como entrada los requisitos y el análisis de las tareas del usuario. OVID genera a partir de estas entradas una salida estructurada que puede ser integrada fácilmente en las metodologías para el diseño de programas. Para ello, OVID se sirve de UML como lenguaje de especificación.

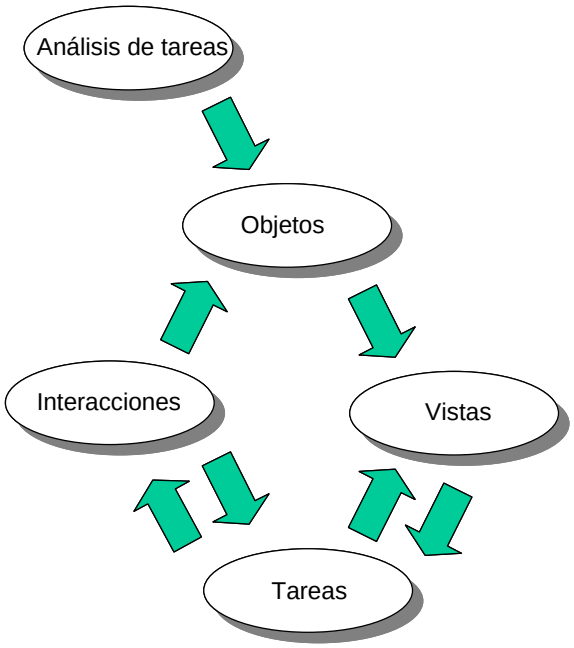


Figura 2.9 Ciclo de desarrollo en OVID.

2.2.2.3 UIDE

En UIDE [Fol91] el diseñador debe especificar un *modelo de aplicación* (modelo de dominio) que está formado por: *acciones de la aplicación*, *acciones de la interfaz,* y *técnicas de interacción*. A cada acción se le asignan los parámetros, las precondiciones y las post-condiciones. Las precondiciones y las post-condiciones son utilizadas en tiempo de ejecución para controlar la interfaz de usuario a través del sistema de ejecución de UIDE. La extensión de UIDE propuesta en [Suk95] añade un *modelo de interfaz*. Mientras que el modelo de aplicación contiene las tareas que serán realizadas por los usuarios, sus restricciones operacionales y los objetos sobre los que estas tareas actúan, el modelo de interfaz contiene los componentes de la interfaz, las tareas de la interfaz independientes de la aplicación y sus restricciones operacionales.

El modelo de aplicación dirige el sistema de ejecución (SUIMS – *Simple User Interface Management System)* para crear una interfaz de usuario funcional. En esta aproximación la información semántica que se almacena en el modelo de aplicación no se pierde en tiempo de ejecución. Esto permite que herramientas más sofisticadas permitan la generación de

sistemas de ayuda sensibles al contexto o interfaces de usuario adaptativas [Suk93].

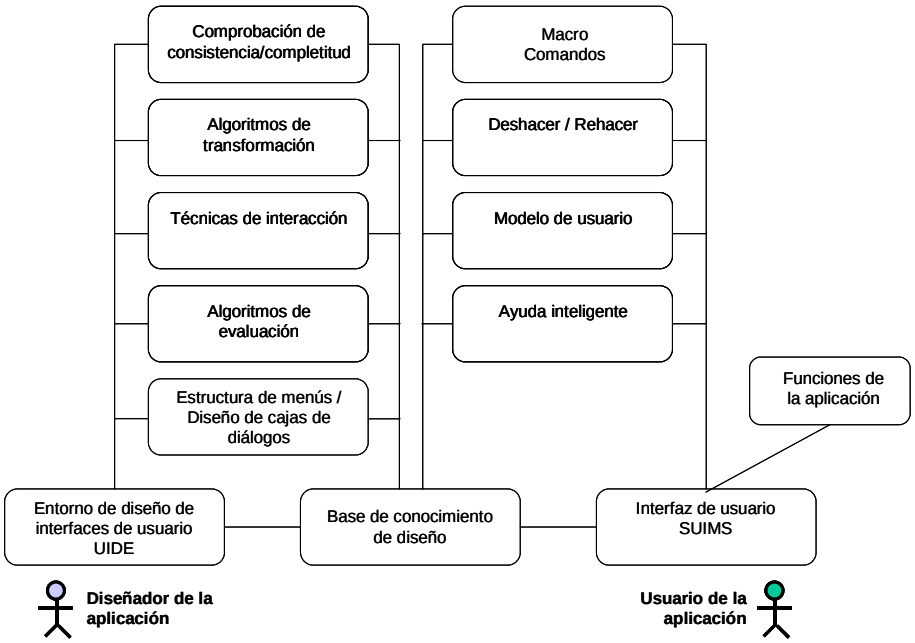


Figura 2.10 Arquitectura de UIDE.

2.2.2.4 Janus

Janus [Bal96] es un sistema centrado en la generación de una interfaz de usuario a partir de modelos de dominio orientados a objetos. Para la generación distintas bases de conocimiento son usadas. Este método no incluye ningún modelo declarativo adicional.

Como principal desventaja de Janus encontramos que al utilizar tan solo un modelo de dominio, donde no se recogen los requisitos de la interfaz de usuario, las interfaces de usuario generadas no son de una calidad demasiado alta.

2.2.2.5 TERESA

TERESA *(Transformation Environment for inteRactivE Systems representAtions)* es una herramienta diseñada para la generación de interfaces de usuario para distintas plataformas a partir de modelos de tareas creados utilizando la notación de ConcurTaskTrees. ConcurTaskTrees (CTT) [Pat99][Mor02] es una notación basada en el lenguaje formal LOTOS [ISO88] para el

análisis y generación de interfaces de usuario a partir de modelos de tareas. Esta notación está siendo utilizada ampliamente en distintas metodologías como método para la especificación del modelo de tareas.

CTT se centra en las actividades, presentando una estructura jerárquica de éstas. Para facilitar su especificación CTT posee una notación gráfica. Las relaciones temporales y de concurrencia entre las distintas tareas pueden ser especificadas utilizando los operadores correspondientes. La notación CTT puede ser editada utilizando el editor CTTE[5].

En CTT existen cuatro tipos de tareas:

- Tareas de usuario: son las tareas realizadas por el usuario; normalmente son actividades cognitivas importantes, como por ejemplo decidir cual es la mejor estrategia para resolver un problema.

- Tareas de aplicación: son las tareas ejecutadas completamente por la aplicación. Las tareas de aplicación suministran información al usuario, como por ejemplo presentar los resultados de una consulta a una base de datos.

- Tareas de interacción: son las tareas que realiza el usuario interactuando con el sistema, por ejemplo pulsar un botón.

- Tareas abstractas: son las tareas que necesitan actividades complejas para su ejecución (y que normalmente se descompondrán en tareas más simples), como por ejemplo una sesión del usuario con el sistema.

Operadores temporales en CTT

Una tarea podrá ser descompuesta en distintas subtareas de distintos tipos. Estas subtareas podrán ser relacionadas mediante los operadores temporales de CTT. Dichos operadores se hallan descritos en la tabla 1.

[5] http://giove.cnuce.cnr.it/ctte.html

Tabla 1. Operadores temporales de CTT.

Operador	*Descripción*
Concurrencia independiente (T1 ||| T2)	Las tareas pueden acontecer en cualquier orden sin restricción. Por ejemplo, monitorizar una pantalla y hablar por un micrófono.
Alternativa (T1 [] T2)	Elección entre un conjunto de tareas. Una vez se ha seleccionado una tarea, el resto de tareas no estarán disponibles hasta que la tarea seleccionada haya sido completada. Por ejemplo, al iniciar un procesador de textos, el usuario puede abrir un documento existente o crear uno nuevo.
Concurrencia con intercambio de información (T1 |[]|T2)	Dos tareas pueden ejecutarse concurrentemente pero han de sincronizarse para intercambiar información. Por ej., en un procesador de texto se puede editar y desplazar verticalmente el texto en cualquier orden. Sin embargo, sólo es posible editar la información que es visible en la región actual.
Desactivación (T1 [> T2)	La tarea T1 es definitivamente desactivada una vez que la primera acción de la segunda tarea T2 comienza. Este concepto es usado en las interfaces de usuario cuando el usuario puede deshabilitar un conjunto de tareas y habilitar otro conjunto. Por ejemplo, tras pulsar un botón.
Activación (T1 >> T2)	La tarea T1 al acabar habilita la realización de la segunda (T2). Por ej., un usuario ha de conectarse 1º al sistema para poder interactuar con los datos
Activación con paso de información (T1 []>>T2)	En este caso, la tarea T1 proporciona a T2 más información que la mera activación. Por ejemplo, T1 permite al usuario especificar una consulta y T2 proporciona los resultados de la consulta que, evidentemente, dependen de la información suministrada por T1.
Parar-Reanudar (T1 |> T2)	Este operador permite que T2 interrumpa la ejecución de T1. Cuando T2 finaliza, T1 puede ser reanudado desde el punto en el que quedó antes de la interrupción. Por ejemplo, un usuario puede interrumpir sus tareas de edición por la aparición de una ventana modal de impresión. Una vez finalizada la impresión, puede retomar la edición donde lo dejó.
Iteración T*	La tarea se realiza de modo repetitivo. Una vez finalizada, la tarea comienza desde el principio una y otra vez hasta que es interrumpida por otra tarea.
Iteración finita (T1(n))	Se emplea cuando se conoce a priori el número de repeticiones de la tarea.
Tareas opcionales ([T])	La tarea puede ejecutarse o no. Por ejemplo, al rellenar un formulario, algunos campos son opcionales y pueden no ser completados.
Recursión	Significa que en el subárbol generado por esta tarea aparece a su vez la misma tarea como subtarea. Dicho proceso ocurre recursivamente hasta que la tarea es interrumpida por otra.

CTTE: Una herramienta para la especificación con CTT

En la figura 2.11 se puede observar un ejemplo de especificación de las tareas en la interacción con un teléfono móvil utilizando la notación CTT descrita en este apartado utilizando CTTE (*ConcurTaskTrees Environment*).

Una de las características más interesantes de la herramienta que soporta la creación de una especificación utilizando la notación CTT es la posibilidad

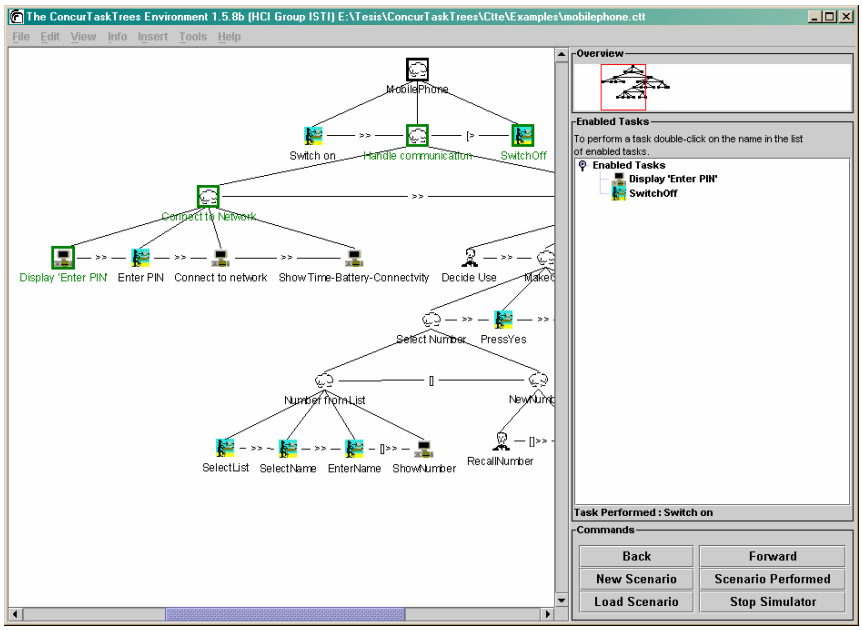


Figura 2.11 Especificación de la interacción en un teléfono móvil con CTT**.**

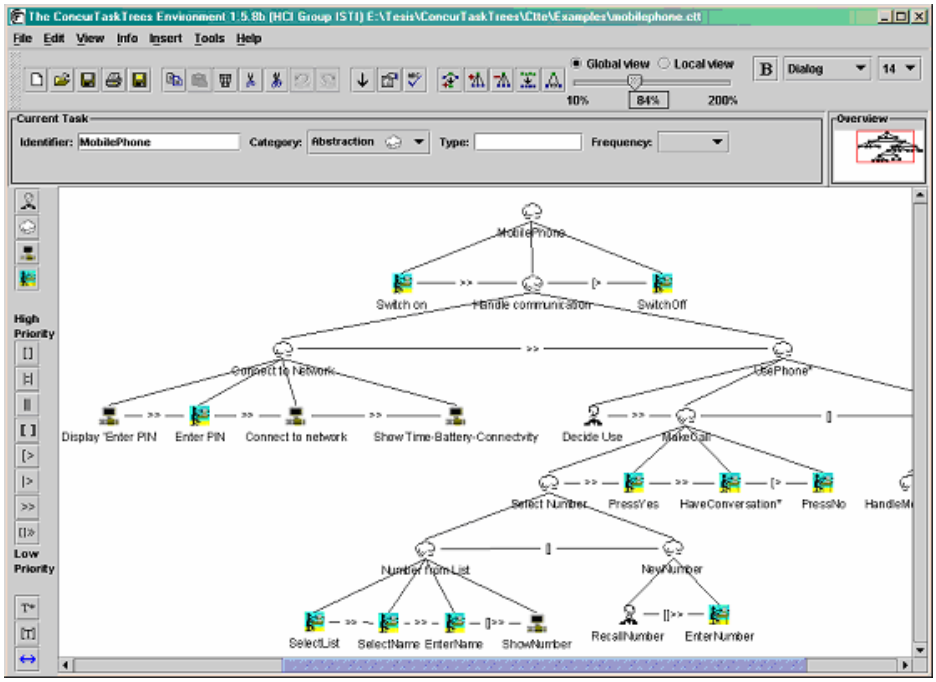


Figura 2.12 Animación del modelo de tareas presentado en la figura 2.11.

de animar el modelo creado para comprobar que el comportamiento es el esperado. En la figura 2.12 se puede observar un ejemplo de animación del modelo de tareas mostrado en la figura 2.11.

La herramienta también permite convertir la notación gráfica al lenguaje LOTOS e incluso ciertas tareas de chequeo de modelos como la comprobación de alcanzabilidad, para comprobar que todas las tareas pueden ser alcanzadas en alguna ocasión.

Actualmente ConcurTaskTrees se ha convertido en la notación para especificación de tareas más utilizada, y va camino de convertirse en el estándar de facto en la especificación de modelos de tareas.

2.2.2.6 MOBI-D

MOBI-D *(Model-Based Interface Designer)* [Pue96][Pue97][Pue98] es un conjunto de herramientas para el desarrollo de interfaces de usuario desarrollado en la Universidad de Stanford por Ángel Puerta y su equipo.

Las fases de diseño de una interfaz de usuario en MOBI-D

MOBI-D está conformado por cuatro herramientas complementarias que cubren todo el ciclo de vida del desarrollo de una interfaz de usuario:

- Elicitación de las tareas del usuario: esta fase es soportada por la herramienta U-Tel.

- Definición de los modelos de tareas, usuario y dominio. La edición de estos modelos es creada con la herramienta MOBI-D.

- Integración de los modelos de tareas del usuario, usuario y dominio. La integración de estos modelos es soportada por la herramienta MOBI-D.

- Análisis de los modelos de diseño de forma interactiva para derivar los potenciales diseños de la presentación y los diálogos. Esta actividad es suportada por la herramienta TIMM.

- Creación de la presentación y los diálogos basados en las tareas. Esta fase es soportada por la herramienta MOBILE.

La arquitectura de MOBI-D

Como se puede ver en la figura 2.13 el modelo de interfaz actúa como repositorio central para el conocimiento sobre el diseño de la interfaz. En este repositorio tenemos una representación declarativa de todos los aspectos relevantes de la interfaz de usuario, incluyendo sus componentes y su diseño.

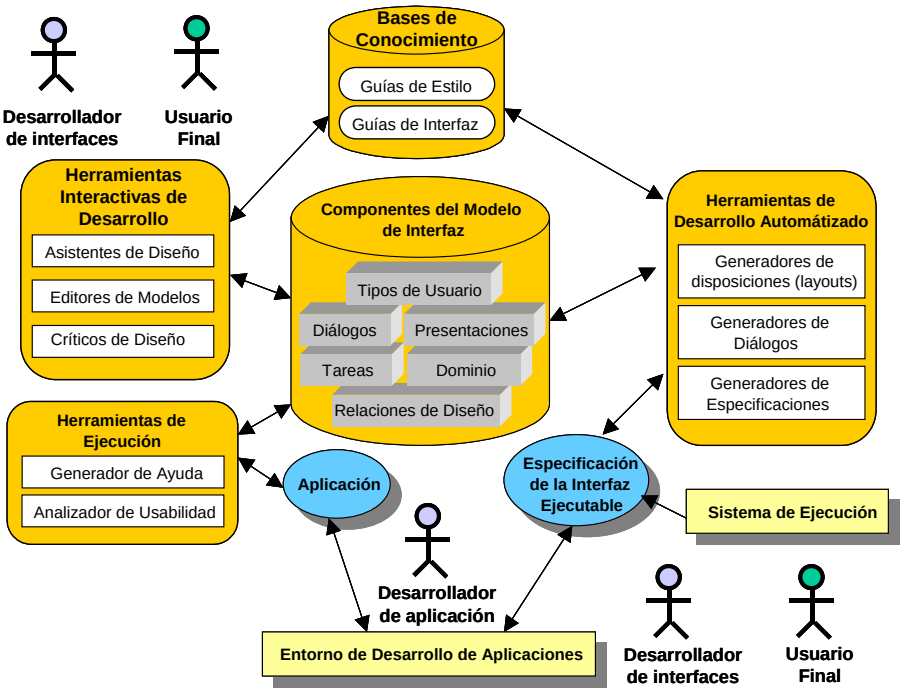


Figura 2.13 Arquitectura de MOBI-D.

Los desarrolladores acceden y modifican el modelo de interfaz a través de las herramientas proporcionadas, las cuales aportan un cierto grado de automatización. Obsérvese como se mantiene una separación entre el desarrollo de la parte funcional de la aplicación y la interfaz de usuario.

MOBI-D ha sido otra de las propuestas basadas en modelos más influyentes en las propuestas actuales, como evolución a su antecesor MECANO, proponiendo un ciclo completo dentro del desarrollo de interfaces de usuario cubierto por un conjunto de herramientas.

2.2.2.7 Wisdom

WISDOM *(Whitewater Interactive System Development with Object Models)* [Nun00][Nun01] es una propuesta metodológica para el desarrollo de interfaces de usuario basada en UML. Este método está diseñado para el desarrollo dentro de pequeñas empresas, y surge como evolución de UCEP *(User-Centered Evolutionary Prototyping)* [Nun98]. UCEP estaba basado en OMT, que fue evolucionando a UML en WISDOM con la aparición de este.

La genealogía de WISDOM puede verse en la figura 2.14 En la parte izquierda se encuentran las influencias en WISDOM procedentes de la Ingeniería del Software, mientras que en la parte derecha pueden encontrarse las influencias provenientes del campo de la interacción persona-ordenador.

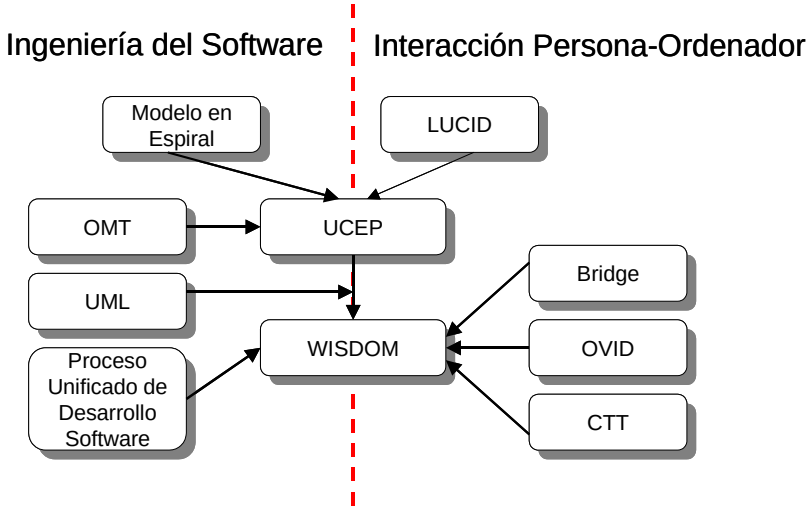


Figura 2.14 Genealogía de WISDOM.

WISDOM toma el modelo en espiral de Bohem [Boh88], mientras que de LUCID [Kre96] toma la perspectiva centrada en el usuario. Como notación para la representación de los modelos propuestos se ha utilizado UML, aprovechando las mejoras que introduce con respecto a OMT. Dentro de este método sus autores también proponen una adaptación a UML del modelo de tareas utilizado en los ConcurTaskTrees de Paternò (ver apartado 2.2.2.5).

WISDOM es una metodología ligera, pensada para ser aprendida y aplicada en un corto espacio de tiempo.

Arquitectura de los modelos en WISDOM

En la figura 2.15 se puede observar la arquitectura y las relaciones entre los distintos modelos que se definen en WISDOM. Los modelos de requisitos (dominio, negocio y casos de uso) representan una vista externa del sistema expresada en el lenguaje del cliente, mientras que el resto representan una vista interna descrita utilizando el lenguaje del desarrollador.

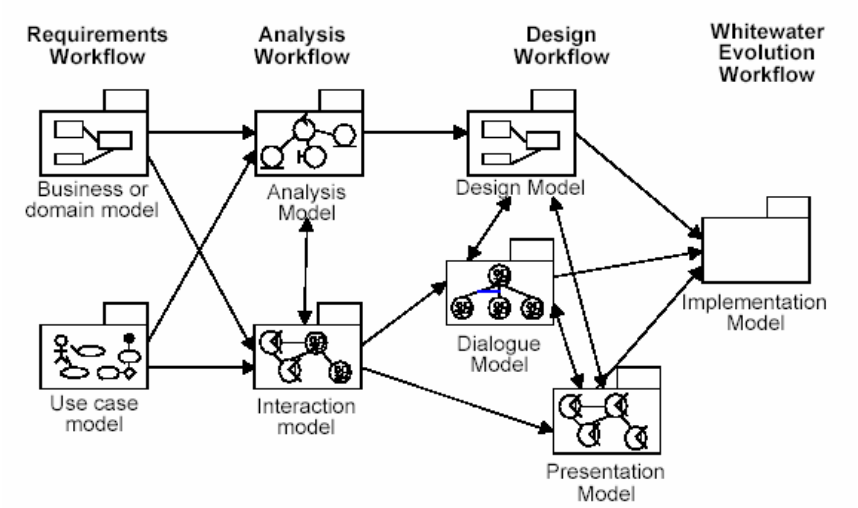


Figura 2.15 Arquitectura de modelos en WISDOM [Nun01].

El modelo de dominio captura los objetos más importantes dentro del contexto del problema. El modelo del negocio sin embargo, describe los procesos que son realizados por los usuarios y las entidades que dichos usuarios manipulan. En WISDOM el modelo de negocio es representado utilizando los casos de uso con la descripción detallada de los flujos de las tareas, mientras que el modelo de dominio es representado usando diagramas de clases.

El modelo de casos de uso describe cuáles son los servicios que el sistema ofrece al usuario. Para su representación se utilizan casos de uso junto con diagramas de actividades (véase la figura 2.16).

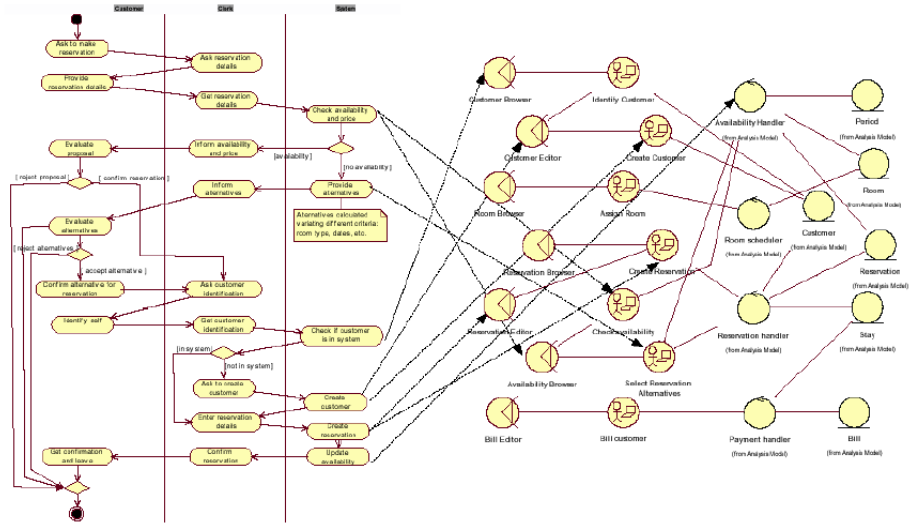


Figura 2.16 Especificación de un caso de uso en WISDOM [Nun01].

Los modelos de diseño reflejan la separación entre la funcionalidad interna de la aplicación y la interfaz de usuario a un bajo nivel de abstracción.

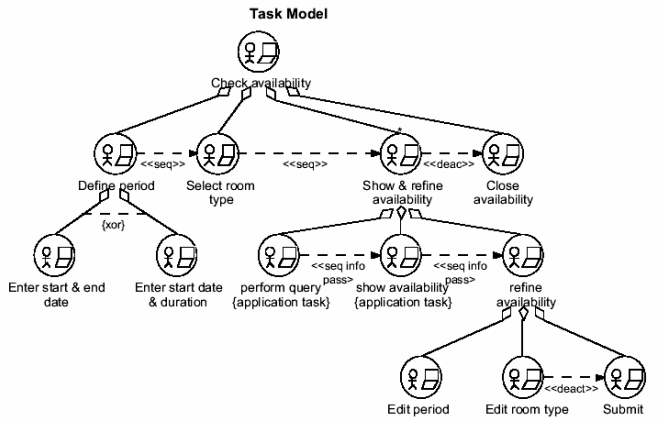


Figura 2.17 Especificación de una tarea en WISDOM [Nun01].

Estos modelos expresan cómo se deben llevar a cabo los casos de uso identificados. Para la especificación de estos modelos en WISDOM se utilizan diagramas de clases con estereotipos dependientes del lenguaje.

El modelo de diálogo es expresado utilizando una adaptación a UML de ConcurTaskTrees (véase la figura 2.17).

El modelo de interacción contiene la descripción de los aspectos técnicos realizados por el equipo de desarrollo de la interfaz de usuario.

El modelo de presentación describe la interfaz de usuario en términos de unos constructores estereotipados que describen la estructura y navegación entre los distintos objetos. En la figura 2.18 puede verse un ejemplo de modelo de presentación para un sistema de reservas de un hotel.

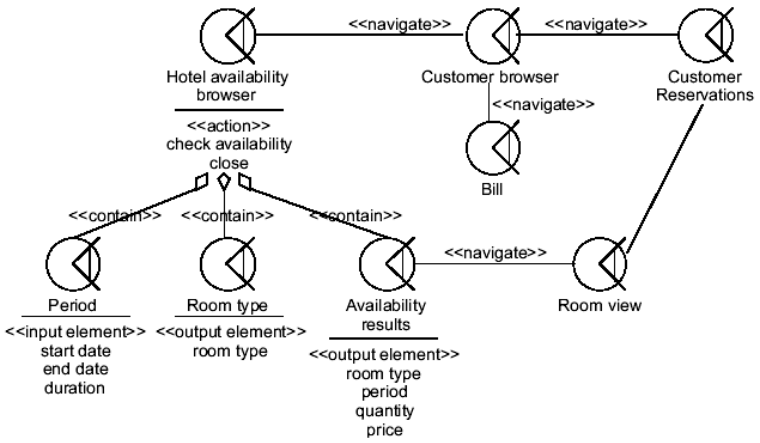


Figura 2.18 Especificación de un modelo de presentación en WISDOM [Nun01].

La transformación del modelo abstracto a una interfaz de usuario se realiza siguiendo el método propuesto en Bridge [Day98]. Actualmente los autores de WISDOM trabajan en el desarrollo de distintos compiladores de modelos que permitan la generación automática de interfaces de usuario a partir del formato de intercambio XMI.

2.2.2.8 Just-UI

El propósito de Just-UI [Mol03b] es la generación automática de interfaces de usuario para distintas plataformas utilizando para ello modelado conceptual. Este método está orientado a la generación de aplicaciones de gestión principalmente.

La principal aportación de Just-UI es la inclusión de patrones conceptuales para la captura de los requisitos necesarios para la generación de la interfaz de usuario de forma automática extendiendo OO-Method [Pas97], un método para la generación automática de aplicaciones basado en el lenguaje formal OASIS [Let98].

Dentro de este método se introducen tres tipos de patrones. Los patrones simples que constituyen las primitivas necesarias para la creación de la interfaz de usuario (como por ejemplo, un *filtro*), y los patrones de presentación, que permiten la agrupación de los AIO que componen las unidades de presentación (véase el apartado 2.2.2.1) en un patrón. Por ejemplo, el patrón *Instancia de Presentación* permite modelar la presentación e interacción con una instancia.

Además, de los dos tipos de patrones anteriormente comentados, también se introduce el concepto de acceso al sistema para modelar los puntos a través de los cuales el usuario puede comenzar su interacción con el sistema.

Para modelar la navegación se crean diagramas de navegación, los cuales consisten en un grafo, cuyos nodos son los patrones identificados.

El método propuesto en esta aproximación es soportado por la herramienta Just-UI/Visio.

2.2.2.9 OO-H

OO-H (*Object Oriented Hypermedia Method*) [Cac03] es un método diseñado para la creación de aplicaciones Web. Para ello se basa en tres modelos principales: el Diagrama de Acceso Navegacional (DAN), el Diagrama de Presentación Abstracta (DPA) y el Diagrama de Diseño Visual (DDV).

El método utiliza diagramas de clases para modelar el dominio del problema, el cual es completado añadiéndole una serie de estereotipos.

El diagrama de acceso navegacional se crea usando cuatro constructores básicos:

- Destino navegacional: es usado para agrupar constructores facilitando el diseño, de forma análoga a como funcionan los paquetes en UML.

- Clase navegacional: son vistas de las clases identificadas en el modelo de dominio. A través de estas vistas se puede especificar la visibilidad de los distintos atributos de las clases.

- Enlace navegacional: permiten describir los caminos que puede seguir el usuario para navegar a través de las vistas creadas con las clases navegacionales.

- Colección: es un agrupamiento de enlaces navegacionales (por ejemplo para crear menús).

El diagrama de presentación abstracta refleja cómo será la estructura y cuáles son las relaciones de cada página, de forma análoga a como sucede en la definición de interfaz de usuario abstracta en [Bod93], [Loz00], [Pin02] o [Cal03].

Finalmente, el diagrama de diseño visual permite crear una presentación más concreta basada en el diagrama de presentación abstracta, donde se puede definir principalmente, cómo se va a realizar la visualización de los distintos constructores usados en diagrama de acceso navegacional.

OO-H permite una personalización de la interfaz de usuario Web generada basándose en un modelo de usuario definido a través de un diagrama de clases. Dentro de dicho diagrama de clases también es posible la inclusión de perfiles que pueden ser usados por los usuarios particulares siguiendo relaciones de herencia.

Todo el proceso de diseño puede ser realizado a través de la herramienta VisualWADE[6].

2.2.2.10 UWE

UWE [Koc01] es una metodología basada en modelos proveniente del ámbito hipermedial. Dicha metodología propone un método que cubre todo el proceso de desarrollo del software. La principal ventaja de UWE es que sólo usa modelos basados en UML para el modelado de la aplicación, introduciendo un nuevo perfil para ello. De esta forma facilitando el acceso a dicho método de los desarrolladores acostumbrados al uso de UML. Además, también es un método conservativo con respecto a los modelos UML que utiliza. Para completar la especificación del sistema, UWE hace uso del lenguaje OCL [War99], lenguaje habitualmente utilizado en la especificación de restricciones dentro de los distintos diagramas UML.

[6] http://www.visualwade.com

UWE propone un método de diseño para el diseño de aplicaciones hipermediales adaptativas, por lo que enfatiza en gran medida el concepto de personalización dentro del metamodelo utilizado.

La metodología propone un proceso compuesto por cuatro actividades, cada una de las cuales determina la especificación de una serie de modelos.

1. Etapa de análisis: dentro de esta etapa se realiza un análisis del sistema construyendo para ello diagramas de casos de uso.

2. Etapa de diseño conceptual: dentro de esta etapa se modela el universo de la aplicación, creando para ello el modelo de dominio.

3. Etapa de diseño navegacional: dentro de esta etapa se define la navegación entre los distintos objetos del dominio. Para ello se construyen los modelos de *Espacio de navegación* y *Estructura de navegación*.

4. Etapa de diseño de la presentación: la presentación se describe en función de distintos modelos estándares UML.

El proceso de desarrollo propuesto es soportado por la aplicación AgoUWE [Kna03], herramienta que se apoya en ArgoUML [Arg03] para su implementación.

2.2.2.11 UML*i*

En UML*i* [Pin02] Pinheiro da Silva propone una ampliación de la notación propuesta dentro de UML para intentar atajar las limitaciones que dicha notación presenta para el diseño de interfaces de usuario.

UML*i* es, al igual que UWE, una ampliación de UML conservativa, facilitando su utilización por parte de desarrolladores familiarizados con UML. El método está centrado en la especificación de interfaces de usuario, y no en la generación de una interfaz de usuario ejecutable a partir de su especificación. Al igual que muchas de las propuestas para el desarrollo de interfaces de usuario basados en modelos, centra su radio de acción principalmente a las interfaces de usuario basadas en formularios. El método se apoya en la notación formal del lenguaje LOTOS [ISO88].

Pinheiro ofrece una serie de primitivas para la especificación abstracta de la interfaz de usuario:

- *Freecontainer*: representan contenedores raíz, que no pueden ser contenidos por ningún otro contenedor.

- *Container*: representan contenedores que podrán contener o ser contenidos por otros contenedores. Estos contenedores también podrán contener a todo el resto de primitivas abstractas (excepto a las *Freecontainer*). Son equivalentes a las unidades de presentación en TRIDENT (véase el apartado 2.2.2.1).

- *Inputter*: representan objetos de interacción abstractos a través de los cuales el usuario puede hacer una operación de entrada (como por ejemplo, obtener un dato a través del teclado).

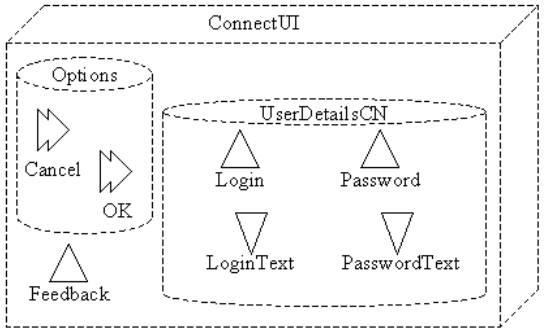


Figura 2.19 Especificación abstracta de una interfaz de usuario en UML*i* [Pin02].

- *Displayer*: representan objetos de interacción abstractos usados para realiza operaciones de salida (como por ejemplo, mostrar un mensaje al usuario).

- *Editor*: representan objetos de interacción abstracta que son a la vez *Inputter* y *Displayer*.

- *ActionInvoker*: representan aquellos objetos de interacción abstractos capaces de recibir eventos y desencadenar una acción (como por ejemplo, un botón).

El modelo de tareas en UML*i* es especificado utilizando una extensión de los diagramas de actividad. Una de las extensiones aplicadas sobre los

diagramas de secuencia usados en la inclusión de nuevos operadores para la selección de estados que permiten especificar los conceptos de repetición, opcional u orden aleatorio.

El método es soportado por ARGO*i*[7], una extensión del entorno ArgoUML [Arg03] de código abierto.

2.2.2.12 IDEAS

IDEAS *(Interface Development Environment within OASIS)* [Loz00] es una metodología de desarrollo de interfaces de usuario basada en modelos que cubre todo el ciclo de vida del desarrollo de una interfaz de usuario. El método permite la especificación de la interfaz de usuario tanto a través de la especificación de una serie de diagramas basados en UML, como formalmente usando el lenguaje de especificación *OASIS* [Let98]. En IDEAS se propone un ciclo de vida iterativo centrado en el usuario. La figura 2.20 muestra los diagramas usados en cada una de las fases del desarrollo de una interfaz de usuario en IDEAS.

El modelo de tareas en el caso de IDEAS es modelado a través de dos tipos de diagramas de secuencia. El modelo de dominio es modelado usando diagramas de clases. Los diagramas de clases son usados además en la especificación del diagrama de roles (usado para la personalización estática de la interfaz de usuario para cada uno de los roles definidos).

El modelo de diálogo es modelado usando dos tipos de diagramas basados en los diagramas de estados. Por una parte se modela las transiciones que pueden ocurrir entre las distintas ventanas (diagrama de interacción de diálogos), y por otra parte se modela cuáles son las transiciones internas que se producen dentro de cada ventana (diagrama de estados internos).

En IDEAS también se plantea una especificación abstracta de la interfaz de usuario(diagrama de especificación de componentes). Esta especificación abstracta es traducida al lenguaje XUL [Moz03] para su visualización.

[7] http://www.cs.man.ac.uk/img/umli/download2.html

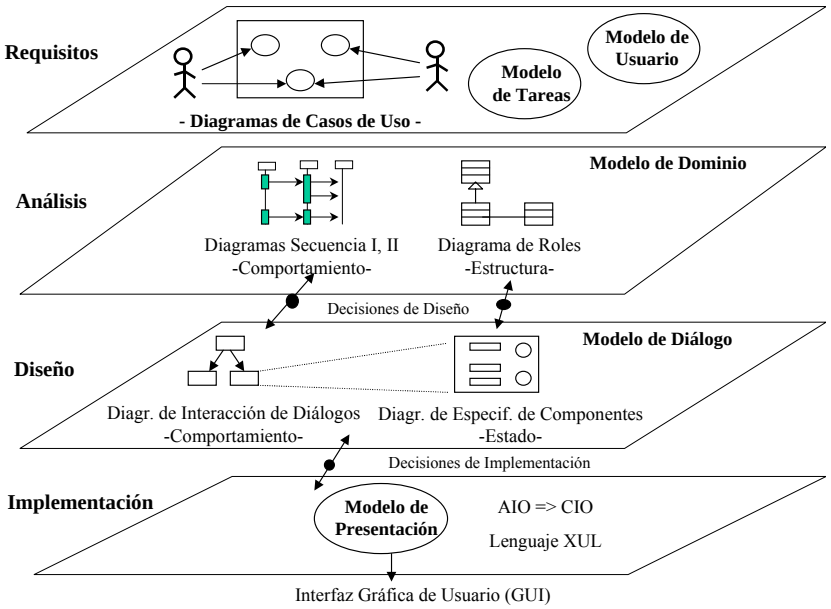


Figura 2.20 Etapas de diseño en IDEAS [Loz00].

2.2.2.13 Otras aproximaciones

Dentro de este capítulo se han recogido una buena parte de las aproximaciones para el diseño de interfaces de usuario basadas en modelos. Sin embargo, existen otras aproximaciones donde se plantean ideas similares a las expuestas a lo largo de la presentación de los métodos descritos en este capítulo.

Desde el mundo de la hipermedia existen otras interesantes aproximaciones como son WebML [Cer00], OOWS [Pas03] o OOHDM [Ros96][Sch95].

WebML, donde se presenta un método de diseño de interfaces de usuario hipermedia de una forma similar a la presentada en OO-H (véase el apartado 2.2.2.9), aunque en este caso usando diagramas entidad/relación en vez de diagramas de clases para modelar el dominio de la aplicación. WebML está soportado por la herramienta comercial WebML[8].

[8] http://www.webml.org/webml/page1.do

OOWS extiende OO-Method [Pas97], un método para el desarrollo de aplicaciones basado en el lenguaje de especificación *OASIS* [Let98] para la generación automática de aplicaciones, para cubrir el desarrollo de aplicaciones Web.

Finalmente, dentro del ámbito de herramientas comerciales, basadas en el paradigma de diseño de interfaces de usuario basado en modelos, podemos encontrar también Olivanova[9], basado en OO-Method [Pas97]. Esta herramienta permite la generación de aplicaciones basadas en formularios automáticamente, a partir de un detallado modelado conceptual, para distintos lenguajes destino. El código generado puede ser entonces compilada para producir la aplicación final.

2.3 Diseño de interfaces de usuario adaptativas basadas en agentes

El diseño de interfaces de usuario adaptativas ha ido ligado a aproximaciones ad-hoc, donde las capacidades de adaptación son incluidas en el código de la aplicación, haciendo difícil, si no imposible su reutilización.

Algunas aproximaciones basadas en modelos han intentado introducir cierto grado de adaptación, aunque normalmente limitándolo a sistemas de ayuda sensibles al contexto, capaces de mostrar una ayuda adaptada al conocimiento del usuario [Suk93]. En FUSE-System [Bau96], se plantea la generación de la ayuda [Lon95] de una aplicación de acuerdo a un reconocimiento de planes basándose en la técnica HIT *(Hierarchic Interaction graph Templates)* [Sch94a] de especificación de interfaces de usuario.

Sin embargo, hasta la fecha no se ha presentado ningún método de diseño de interfaces de usuario donde explícitamente se permita el diseño de las capacidades de adaptación de una interfaz de usuario, incluyendo su diseño dentro del ciclo de vida del diseño de una aplicación. Es este uno de los problemas donde pretende profundizar la presente tesis, aportando un paso hacia delante en la integración total del diseño de la adaptación en el proceso de desarrollo de las interfaces de usuario.

[9] http://www.care-t.com

El diseño de interfaces de usuario adaptativas plantea dos frentes principales: por una parte es necesario modelar los constructores que describen el proceso de adaptación (véase el apartado 2.1.1.2), y por otra parte se hace necesaria la inclusión de algún mecanismo que sea capaz de inferir conocimiento a partir de la información recogida de la interacción con el usuario y su entorno, y que sea capaz de tomar decisiones basándose en dicha información. Estas decisiones pueden ser determinantes en la aceptación o no por parte del usuario de un sistema adaptativo. La mala elección de la adaptación a realizar o la elección de un momento inadecuado para aplicarla puede conducir a una clara degradación de la usabilidad de la interfaz de usuario adaptativa, yendo en contra del principal fin último de la adaptación: la mejor de la usabilidad de las interfaces de usuario.

La toma de decisiones, así como la inferencia de conclusiones a partir de una serie de datos son facultades humanas, las cuales se intenta emular a través de distintas técnicas dentro del mundo de la Inteligencia Artificial. Sistemas basados en reglas [Buc84], reyes Bayesianas [Jen01] o redes neuronales [Fre91] son algunas de las técnicas tradicionalmente usadas para dicho propósito.

Sin embargo, en los últimos años se ha suscitado un gran interés por el concepto de agente software [Woo94], como abstracción para el desarrollo de aplicaciones altamente dinámicas donde puede ser incluido un modelo mental (normalmente el modelo BDI – *Beliefs, Desires, Intentions* [Bra87]) que permita aportar al sistema ciertas capacidades de razonamiento. Esto es particularmente aplicable también dentro del marco de desarrollo de interfaces de usuario, donde distintos proyectos de investigación como *Letizia* [Lie97], *Steve (Soar Training Expert for Virtual Environments)* y *Adele (Agent for Distance Learning: Light Edition)* [Joh98], Lumière [Hor98] o *Collagen* [Ric98] han intentado aplicar el paradigma de agentes en el desarrollo de interfaces de usuario con distintos tipos de adaptación, principalmente en adaptaciones encaminadas a guiar al usuario durante la utilización de la aplicación.

2.3.1 Agentes y Sistemas Multiagente

Hemos nombrado el concepto de agente, pero, ¿qué es un agente?. Esta pregunta no es en absoluto fácil de contestar, y sirva para demostrarlo las definiciones que distintos autores han aportado del concepto de agente que se muestran a continuación.

"*Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed*" [Mae95].

"*Let us define an agent as a persistent software entity dedicated to a specific purpose. 'Persistent' distinguishes agents from subroutines; agents have their own ideas about how to accomplish tasks, their own agendas. 'Special purpose' distinguishes them from entire multifunction applications; agents are typically much smaller.*" [Smi94].

"An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." [Fra96] (véase la figura 2.21)

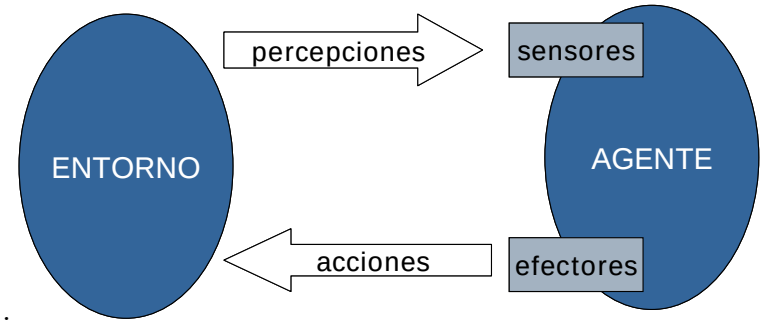


Figura 2.21 Esquema básico de funcionamiento de un agente [Fra96].

La ausencia de una definición de agente consensuada se debe a que a menudo se llama agente a entidades de muy distinta naturaleza. Uno de los principales motivos para la falta de una definición consensuada son los distintos dominio de aplicación y bagaje de los investigadores que utilizan el concepto de agente. Mientras que para algunos un agente es una simple extensión del concepto de objeto, para otros es un ente dotado de ciertas características de inteligente. Como se puede derivar de las definiciones anteriores, aunque no presentan una definición homogénea del concepto, si que proponen unas características comunes. Por ello, habitualmente se suele recurrir a la descripción de un agente como una entidad que cumple una serie de propiedades. Dichas propiedades se suelen clasificar siguiendo dos definiciones: la noción débil de agente y la noción fuerte de agente [Woo95]. Mientras que la noción débil es la habitualmente usada en la Ingeniería del Software Orientada a Agentes, y está más cercana al concepto de programación concurrente orientada a objetos, la noción

fuerte está más ligada al mundo de la Inteligencia Artificial, y añade a la definición débil de agente una serie de propiedades que restringen en gran medida las entidades que pueden ser consideradas como agentes. La noción débil propone las siguientes propiedades como requisitos mínimos para considerar una entidad como agente:

- *Autonomía*: los agentes encapsulan sus estados, de forma que no es accesible al resto de agentes. Basándose en este estado realizan decisiones sobre qué hacer sin la ayuda de un ser humano u otros agentes.

- *Reactividad*: un agente que está situado en un entorno (que puede ser el mundo real, una interfaz gráfica de usuario, una sociedad artificial de agentes, Internet, o una combinación de estos entornos), percibe los cambios que se producen en su entorno y es capaz de reaccionar de acuerdo a su estado, y a la percepción que tiene de su entorno.

- *Iniciativa (pro-actividad)*: a parte de reaccionar ante los cambios que se producen en su entorno, un agente también es capaz de tomar la iniciativa para realizar tareas que acerquen al agente hacia sus objetivos.

- *Sociabilidad*: los agentes también son capaces de interaccionar con su entorno. Dicho entorno puede ser una sociedad artificial de agentes, o incluso una persona humana, como ocurre en el caso de las interfaces de usuario.

La noción débil es la definición usada con más fuerza en el campo de la Ingeniería del Software Orientada a Agentes (AOSE – *Agente Oriented Software Engineering)* [Woo01]. La noción fuerte del concepto de agente añade a la definición débil de agente algunas propiedades más, que hacen la definición más restrictiva. Las características añadidas son las siguientes:

- *Movilidad*: capacidad del agente para moverse a través de una red electrónica.

- *Veracidad*: la seguridad de que el agente no transmitirá información falsa conscientemente.

- *Benevolencia*: se asume que el agente hará aquello para lo que se creó.

- *Racionalidad*: se asume que el agente intentará alcanzar las metas que le fueron asignadas de la mejor forma posible, basándose en la información que posee del mundo exterior.

La noción fuerte de agente está más cercana al mundo de la Ingeniería del Conocimiento, y es por tanto la más aceptada dentro de dicho campo.

Hemos hablado de términos como racionalidad, iniciativa o integración en una sociedad artificial de agentes. Estos términos implican que el agente debe mantener un estado mental. Existen distintas formas para representar el estado mental del agente, pero sin duda la más extendida es la basada en la teoría del filósofo Michael Bratman [Bra87], donde se propone que en el razonamiento humano los factores que priman son las creencias, los deseos y las intenciones (*BDI – Beliefs, Desires, Intentions)*. Las creencias corresponden a la información que el agente tiene del mundo. Dicha información puede ser incompleta o incluso errónea, pero sin embargo es lo que el agente ha percibido del mundo. Los deseos son los estados que al agente le gustaría alcanzar (planes del agente). Finalmente, las intenciones representan los deseos que se ha comprometido a intentar alcanzar (las metas actuales del agente). El agente normalmente no será capaz de alcanzar todos sus deseos, y por lo tanto tendrá que decidir qué deseos intentar conseguir primero y cómo.

Un *Sistema Multiagente* (MAS) es un conjunto de agentes autónomos, generalmente heterogéneos y potencialmente independientes, que trabajan en común (o enfrentándose si no son colaborativos) resolviendo un problema. Pero, ¿cuáles son las razones que han despertado tanto interés por los sistemas multiagente? En primer lugar, los sistemas multiagente son una metáfora natural para el modelado de gran cantidad de dominios, donde aparecen una serie de componentes que interaccionan. En segundo lugar, los sistemas multiagente permiten la distribución de forma natural tanto de los datos como del control de ejecución. En tercer lugar, los sistemas multiagente permiten integrar de forma sencilla sistemas heredados (*legacy systems*), es decir, sistemas que aún siendo obsoletos, deben seguir funcionando por razones de necesidad. Dichos sistemas pueden ser encapsulados dentro de un agente, e integrados dentro del sistema multiagente manteniendo toda la funcionalidad necesaria.

Finalmente, este tipo de sistemas permiten crear sistemas abiertos donde no se conozca en tiempo de diseño los componente exactos que tendrá el sistema, sino que éstos se unirán al sistema en tiempo de ejecución interaccionando con el sistema [Woo01].

La propia sociedad humana podría ser considerada como un inmenso sistema multiagente. La sociedad humana es un modelo útil para obtener modelos de sistemas multiagente, ya que la mayoría de las actividades humanas relacionadas con el carácter inteligente son de carácter social, al menos en origen. El ser humano necesita la sociedad para mejorar su calidad de vida, y está en continua interacción con ella. Por otra parte el conocimiento es en su mayor parte colectivo. Los modelos teórico/prácticos de los sistemas multiagente permiten estudiar fenómenos sociales reales. Existen distintas aproximaciones sobre cómo crear las sociedades en los sistemas multiagente basadas en comportamientos sociales humanos o animales, como es el caso del ecosistema de agentes distribuidos presentado por Nelson Minar [Min98].

En los MAS aparecen conceptos de gran interés:

- Actividades conjuntas y cooperación.
- Conflictos, ¿cómo se resuelven?
- Negociación.
- Compromisos y planificación de actividades.
- Modelo del conocimiento, y su comunicación.

Cooperación es el proceso por el que ciertos agentes participantes generan deberes mutuamente dependientes en actividades conjuntas (planes). En principio el agente tiene un problema que resolver. Inicialmente intenta resolver localmente la parte que pueda. A continuación recurre a agentes del mismo nivel para resolver las tareas restantes. Si es necesario recurrirá a agentes situados en otro nivel de abstracción (en otro grupo).

Los conflictos surgen cuando al resolver un problema se dan una o varias de las siguientes circunstancias:

- El conocimiento local es incorrecto o incompleto.
- Coexisten metas diferentes y divergen en algún momento.
- Hay diferentes criterios de evaluación de soluciones.
- Los recursos son limitados.

Sin embargo, la generación de conflictos también aporta una serie de aspectos positivos:

- Se intercambia información.
- Se mejora la robustez e integración
- Se llega a soluciones globalmente óptimas.

Los conflictos se resuelven o evitan mediante mecanismos de prevención y evitación, sistemas de pizarra, o negociación. El proceso de negociación es iterativo: los agentes afectados ofrecen propuestas o posturas, donde ninguna de estas propuestas es más exigente que la anterior. La negociación finaliza cuando hay acuerdo y se crea un plan conjunto. Cuando hay interbloqueo no hay acuerdo y se requiere otro procedimiento de solución o negociación. También se puede apelar a un agente coordinador que puede modificar metas o relajar las restricciones.

Los compromisos forman un conjunto de restricciones sobre las acciones y creencias de cada agente. Se representan mediante conocimiento compartido y local de cada agente con respecto al resto.
Las interacciones entre agentes en sistemas MAS obedecen a la coexistencia de planes elaborados, preferentemente de forma distribuida.

Cuando hablamos de interacción entre los agentes, sin duda uno de los aspectos a abordar es cómo se comparte el conocimiento. Compartir el conocimiento plantea una serie de requisitos adicionales en el sistema: un modelo de representación del conocimiento común. La comunicación entre los agentes en cualquier caso requiere una pila de protocolos que permita la transmisión del conocimiento y de las distintas peticiones de una forma estándar y que sea entendida por todos los agentes. Para ello los estándares más extendidos son el estándar propuesto por la organización FIPA[10] *(Foundation for Intelligent Physical Agents*) y el estándar propuesto por la organización KSE[11] (*Knowledge Sharing Effort)*. Aunque existen otros, como el MASIF [Mil98], propuesto por OMG *(Object Management Group)* en el marco de los agentes móviles. La utilización de lenguajes basados en XML también puede facilitar la comunicación del

[10] http://www.fipa.org
[11] http://www.cs.umbc.edu/kse/

conocimiento entre agentes, siempre y cuando tengan en común la ontología representada en XML.

2.3.2 Agentes de Interfaz

Las interfaces tradicionales están orientadas hacia las interfaces conversacionales, donde el usuario y el agente van actuando por turnos. Los agentes de interfaz conducen hacia un estilo de diseño distinto, basándose en que el agente es capaz de interaccionar con la interfaz incluso cuando el usuario está interaccionando con la interfaz. El usuario no tiene por qué ser consciente de las actividades que el agente está realizando en cada momento.

Un agente de interfaz debería mostrar, al menos, algunas de las características que asociamos a la inteligencia humana: aprendizaje, inferencia, capacidad de adaptación, independencia, creatividad, etc. El usuario realmente delega en el agente para realizar algunas tareas, en vez de ordenar al agente realizar una tarea. Si el usuario percibe que las acciones que realiza el agente las podría hacer él mismo, aumentará para el usuario la sensación de asistencia. Normalmente no delegamos tareas a un agente porque no seamos capaces de hacerlas nosotros mismo, sino porque, sencillamente, no queremos emplear nuestro tiempo en esa tarea, porque la consideramos tediosa en algún sentido. En este sentido la función de autocompletar con los valores tomados del historial de entradas que anteriormente se hayan realizado sobre una caja de texto de un sitio Web que los navegadotes de Internet actuales presentan, es un ejemplo de tarea tediosa que un agente de interfaz puede realizar automáticamente por el usuario incrementando la satisfacción del usuario con el sistema.

El paradigma de manipulación directa de las interfaces gráficas presenta unas representaciones de objetos físicos o conceptuales que permiten que el usuario pueda ejecutar órdenes que cambian el estado de los objetos. En este paradigma, los cambios en el estado de la interfaz son más o menos uno a uno con las órdenes explícitamente invocadas por el usuario [Lie97]. La interfaz de usuario debe ser capaz de modificar el estado de los objetos de la interfaz utilizando manipulación directa, pero sin que el usuario se lo ordene de forma explícita. El agente de usuario capta la información que el usuario suministra a la interfaz de usuario, y es capaz de hacer cambios sobre los objetos que se muestran en la interfaz, sin que exista una correspondencia uno a uno entre las acciones que ejecuta el

agente y las órdenes ejecutadas por el usuario. El agente puede observar lo que el usuario hace durante un buen rato sin decidir hacer nada, o una sola acción de comunicación del usuario con la interfaz puede desencadenar toda una serie de acciones por parte del agente. Un agente de interfaz puede ser considerado como un robot cuyos sensores son las capacidades de entrada de la interfaz, y sus efectores las capacidades de salida de dicha interfaz. Los sensores permiten al agente percibir su entorno, mientras que los efectores son los medios que el agente posee para actuar sobre su entorno.

Los agentes de interfaz están cobrando cada vez más interés debido a la creciente complejidad de las interfaces de usuario y las tareas para las que son empleados. El crecimiento actual de la complejidad de las interfaces es insostenible. El número de operaciones con menús u otros componentes nos conduce hacia interfaces donde es necesario mantener demasiadas barras de herramientas y menús visibles. No se puede seguir manteniendo la correspondencia uno a uno entre órdenes y elementos de la interfaz de usuario. El paradigma de manipulación directa está llegando a extremos que pronto harán las interfaces inutilizables de seguir con la progresión actual de crecimiento.

2.3.2.1 Agentes de Interfaz Autónomos

Un agente de interfaz autónomo opera de forma paralela al usuario. La autonomía del agente implica que el agente estará siempre ejecutándose mientras esté abierta la aplicación a la que está asociado. El agente puede descubrir algún hecho que determine qué es interesante para el usuario, y por lo tanto comunicárselo. Por otra parte, el agente también puede permanecer activo debido a alguna acción que ha realizado el usuario mucho después de que el usuario invoque la orden o incluso después de que el usuario apague el ordenador.

Los asistentes no son suficientes, ya que necesitan demasiada información explícita del usuario, y una supervisión constante por parte de este. Los asistentes son una herramienta de gran utilidad cuando se les permite actuar de forma independiente y de forma concurrente. Si se permite que el agente de interfaz actúe en paralelo con el usuario, mientras éste presta su atención directa a otras actividades, permite que el usuario realmente delegue tareas al agente.

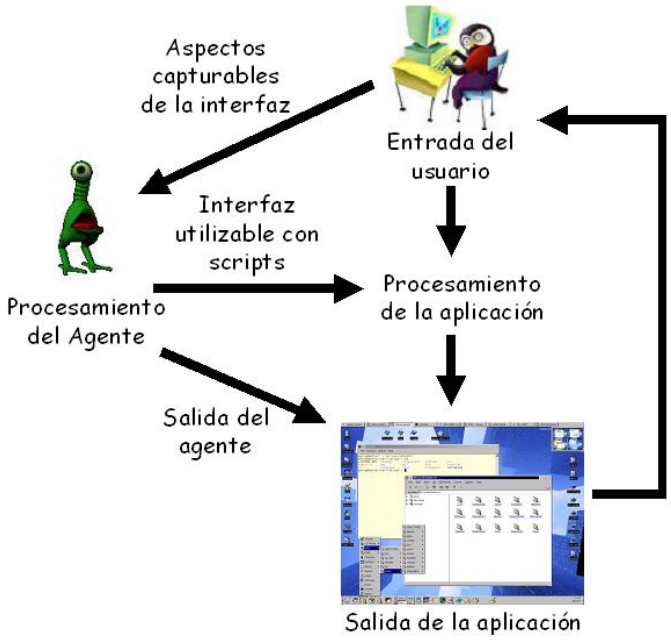


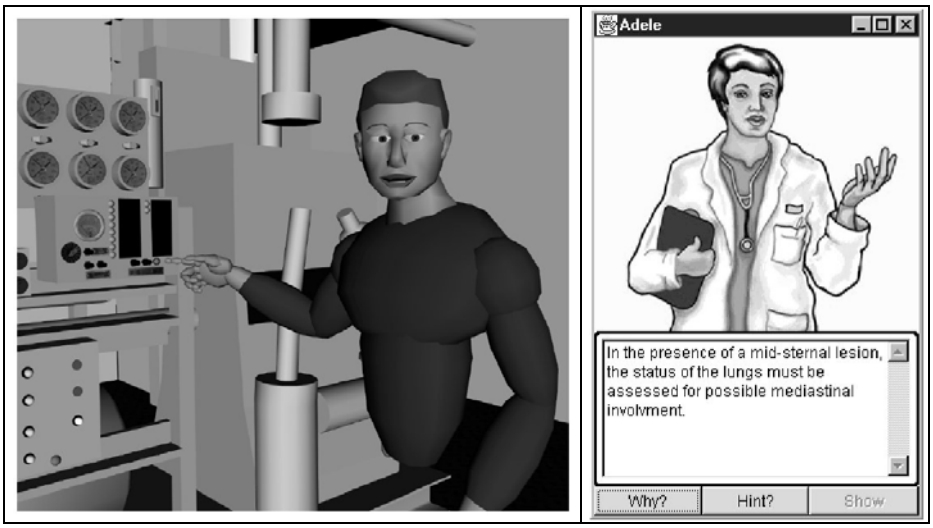
Figura 2.22 Esquema del funcionamiento de un agente de interfaz.

Las interfaces tradicionales (de línea de comandos, o por menús) son conversacionales. El usuario genera un evento de entrada para la interfaz, el sistema lee la entrada, la procesa, ejecuta alguna acción, muestra los resultados y queda esperando la siguiente entrada. El sistema queda inactivo mientras el usuario va preparando la siguiente entrada, o simplemente queda inactivo porque el usuario está deliberando sobre lo que quiere hacer. De esta manera, cada una de las acciones del sistema son iniciadas explícitamente por el usuario. Sin embargo, un agente de interfaz autónomo puede seguir haciendo las acciones que va desencadenando su percepción de la interfaz (su entorno), y que decide realizar de forma autónoma. La figura 2.22 muestra el esquema de funcionamiento básico de un agente de interfaz.

El usuario debe ser capaz de observar las acciones que realizan los agentes de interfaz fruto de su autonomía, mientras que los agentes deben ser capaces de observar las acciones que realiza el usuario, también de forma autónoma.

La Web es un dominio de aplicación especialmente apropiado para los agentes de interfaz, ya que los usuarios de la Web sienten hoy en día la

necesidad de algún tipo de asistente inteligente. Ello se debe principalmente a que ir siguiendo los enlaces manualmente en un navegador no es suficiente para evitar que el usuario sienta que se ahoga en un mar de información.



(a) (b)

Figura 2.23 Agentes en entornos educativos virtuales: (a) Steve. (b) Adele.

Un ejemplo de este tipo de aplicaciones de los agentes de interfaz a la Web es *Letizia* [Lie97]. Este sistema trata de que la búsqueda en Internet se convierta en un proceso cooperativo continuo entre el usuario y un agente de búsqueda. Existen otros proyectos de similares características como WebWatcher [Arm95], Lira [Bal95] o SurfBot[12]. Los agentes de interfaz también han probado ser útiles en entornos educativos. Algunos ejemplos dentro de este campo son Steve *(Soar Training Expert for Virtual Environmnets)* y Adele *(Agent for Distance Learning: Light Edition)* [Joh98]. *Steve* (véase la figura 2.23a) es un agente diseñado para interactuar con estudiantes dentro de un entorno virtual immersivo. Por el contrario, *Adele* (véase la figura 2.23b) ha sido diseñado para interaccionar en interfaces tradicionales.

2.3.2.2 Principios de Diseño para Agentes de Interfaz Autónomos

En el diseño de agentes de interfaz autónomos hay que intentar aprovechar al máximo las posibilidades que estos agentes nos ofrecen.

[12] http://www.surflogic.com/sb30menu.html

Lieberman [Lie97] propone una serie de principios para mejorar el diseño de los agentes de interfaz autónomos:

- *Sugerir es mejor que actuar*. los agentes de interfaz autónomos trabajan mejor en situaciones donde sus decisiones no son críticas. La mayoría de la gente se siente recelosa de permitir que el agente tenga autonomía, porque temen que el agente tome una decisión errónea sin su consentimiento. Sin embargo, existe un gran abanico de posibilidades en la interacción del usuario con el sistema donde el agente puede sugerir, siendo útil sin que sea necesario que tome decisiones críticas.

- *Aprovechar la información que el usuario suministra de forma "gratuita"*: las acciones que realiza el usuario sobre la interfaz de usuario constituyen el material básico sobre el que el sistema puede inferir qué metas o preferencias tiene el usuario, sin que sea necesario ningún tipo de interacción adicional.

- *Aprovechar el tiempo que el usuario utiliza para pensar*: una de las desventajas de las interfaces conversacionales tradicionales es que no realizan ninguna acción mientras el usuario está pensando qué hacer. Si el agente goza de autonomía, mientras el usuario está pensando, puede utilizar ese tiempo de computación para realizar tareas. Esto es especialmente interesante en tareas de búsqueda o exploración, donde el tiempo que el usuario no está interaccionando con la interfaz puede ser usado por el agente para anticiparse al usuario.

- *La atención del usuario puede ser compartida*: una consecuencia lógica del hecho de que los agentes se ejecuten de forma autónoma es que no puede asumir que tiene toda la atención del usuario, tal y como ocurre en la interacción tradicional.

- *Los agentes de interfaz pueden tener un compromiso distinto entre deliberar y actuar*: es bien conocido en el mundo de la Inteligencia Artificial el problema del compromiso entre deliberar y actuar. Ya que pensar sobre un problema requiere tiempo y capacidad de proceso, llega un momento en que hay que detener las deliberaciones y simplemente actuar.

- *Un agente autónomo de interfaz puede no coincidir con el estilo cognitivo de todos los usuarios*: es importante ser consciente de que los usuarios tienen estilos cognitivos distintos, y que un agente de interfaz interactuando con el usuario y la interfaz puede resultar confuso para algunas personas.

2.3.2.3 Características Deseables de los Agentes de Interfaz

En el apartado anterior hemos descrito una serie de principios para el diseño de agentes de interfaz. En este apartado vamos a describir qué características sería interesante que tuvieran los agentes de interfaz. Crabtree et al. [Cra98] proponen las siguientes:

- *Un perfil de usuario común*: el perfil personal de usuario con sus intereses y necesidades debería estar disponible para todos los agentes, de forma que no tenga que ser cada uno de los agentes el que vaya recolectando la información interaccionando con el usuario. Se debería crear algún tipo de repositorio donde el agente pueda acudir a consultar el perfil, y actualizarlo en caso de que sea necesario. Actualmente se está desarrollando un estándar para realizar esta tarea dentro del dominio de las aplicaciones Web llamado *Open Profiling Standard*[13].

- *Capacidad de adaptación*: para hacer un uso óptimo de los agentes de interfaz, el agente debe ser capaz de aprender las preferencias y los hábitos del usuario a lo largo de su utilización. El objetivo debe ser crear un perfil del usuario con un esfuerzo mínimo por parte del usuario.

- *Compartición de la información*: el funcionamiento del agente será normalmente más correcto si éste es capaz de compartir información con otros agentes, en vez de trabajar por su cuenta.

- *Colaboración entre agentes*: los agentes además de compartir información pueden colaborar entre ellos. Por ejemplo, si existen varios agentes de información recabando enlaces interesantes a través de Internet, sería beneficioso que fueran coordinándose para evitar, por ejemplo, recorrer un mismo sitio en busca de enlaces de interés.

[13] http://www.w3.org/TR/NOTE-OPS-FrameWork.html

- *Privacidad de la información*: la privacidad de la información es uno de los puntos más conflictivos a la hora de que el usuario confíe en un agente. Hay que tener en cuenta que los agentes tienen acceso a gran cantidad de información personal del usuario, y que en algunos casos realizan acciones en nombre de usuario. La privacidad es sin duda uno de los mayores obstáculos para que los agentes sean aceptados a lo largo y ancho de la Red.

2.3.3 Ingeniería del Software Orientada a Agentes

La ingeniería del software ha conseguido alcanzar una notable madurez en la comprensión de las características del software complejo. La interacción ha demostrado ser la característica más importante dentro del software con una cierta complejidad. El papel destacado de la interacción ya ha sido resaltado por una gran cantidad de investigadores que afirman que en el futuro, la computación en sí misma será entendida principalmente como un proceso de interacción. El interés en tecnologías relacionadas con agentes suscitado actualmente viene principalmente justificado precisamente porque el paradigma de la ingeniería del software orientada a agentes es una posible solución natural a las interacciones utilizando agentes autónomos capaces de interaccionar con otros agentes para satisfacer los objetivos de diseño [Woo01].

2.3.3.1 ¿Por Qué Ingeniería del Software Orientada a Agentes?

La construcción de software de alta calidad para aplicaciones del mundo real presenta grandes dificultades. Se ha llegado incluso a afirmar que es una de las tareas más complejas (debido tanto al número como a la flexibilidad de los componentes que constituyen el problema, así como sus interacciones). Dicha complejidad se manifiesta debido a la existencia de un gran número de partes integrantes, con una cantidad incluso mayor de interacciones, dentro del software. El papel de la ingeniería del software debe ser, por lo tanto, proporcionar modelos y técnicas que faciliten el manejo de la complejidad del software [Jen00]. El paradigma orientado a agentes pretende precisamente cumplir dicho objetivo.

La concepción de sistemas software como agentes o sistemas multiagente es hoy en día uno de los aspectos que más interés está despertando dentro de la ingeniería del software, como demuestra el enorme esfuerzo que se está realizando en investigación dentro de esta área. La visión de los sistemas multiagente como una metáfora natural de objetos activos en

interacción, la posibilidad de distribuir e incluso desplazar tanto los flujos de datos como de control, la integración de sistemas heredados (*legacy systems*) añadiendo una capa que haga comportarse el sistema heredado como un agente más, y la tremenda potencia de permitir que sean desconocidas en tiempo de diseño las interacciones que se van a producir, introducen unas características que hacen de los sistemas multiagente un paradigma que se ajusta, en gran medida, a las necesidades del software actual (véase la figura 2.24).

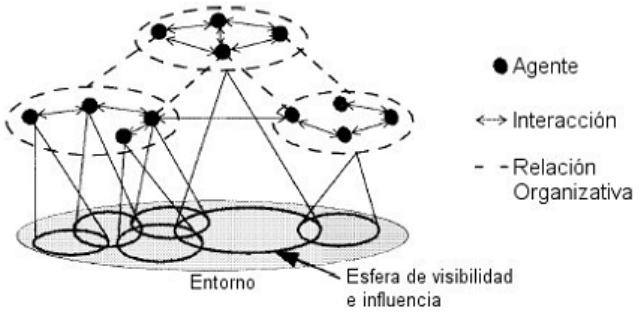


Figura 2.24 Vista de un sistema basado en agentes.

2.3.3.2 Sobre Agentes y Objetos

Cuando una persona novel en el paradigma de orientación a objetos se inicia dentro de aproximaciones orientadas a agentes puede no ser capaz de apreciar las nuevas ideas que este paradigma aporta. Los objetos se definen como entidades computacionales que encapsulan un estado, y son capaces de realizar acciones, o métodos en ese estado, y que se comunican por paso de mensajes. Existe una relación clara entre el concepto de agente y el de objeto. Sin embargo, existen de igual forma diferencias destacables. La primera diferencia la encontramos en el grado de autonomía que uno y otro concepto presentan. Un objeto sí que posee autonomía sobre su estado, y tiene control sobre él. Sin embargo, un objeto no posee, por el contrario ningún control sobre su comportamiento. Si un objeto contiene un método público *método*, cualquier objeto será capaz de invocar dicho método cuando quiera y cuantas veces quiera.

No existe ninguna duda de que cuando se construye un sistema basado en objetos, todos los objetos incluidos en el sistema comparten una meta común. Esto no ocurre en el caso de los agentes. Existe la posibilidad de

diseñar un sistema multi-agente, que compre o venda artículos en nombre de sus respectivos propietarios. En este caso cada agente o grupo de agentes debe defender los intereses de sus propietarios, y por la tanto negociar para conseguir los mayores beneficios [Ant01][Cha96]. Ello implica claramente la imposibilidad de que todos los agentes que interaccionan persigan una meta común. Un agente *i* no ejecutará una acción *a* sólo porque otro agente *j* se lo pida. El agente *j* hará una petición al agente *i* para que ejecute *a*, de forma que la decisión de ejecutar o no la acción será del agente *i*, al contrario de lo que ocurre en el caso de objetos. La filosofía que se persigue puede ser resumida con la siguiente frase: "Los objetos lo hacen gratis; los agentes lo hacen porque quieren hacerlo" [Woo01].

Los agentes presentan además un comportamiento autónomo flexible, para realizar su papel reactivo, proactivo y social. Otra característica que distingue a los agentes de los objetos es que poseen su propio flujo de control. Los agentes están constantemente activos, en un bucle observando su entorno, actualizando su estado interno, y seleccionado y ejecutando una acción. Los objetos, en general, permanecen en letargo hasta que otro objeto invoca alguno los servicios que ofrece. Seguramente, lo más parecido a un agente que existe dentro de la orientación a objetos son los objetos activos [Boo99]. Un objeto activo contiene su propio hilo de ejecución. Además son capaces de manifestar un cierto comportamiento autónomo, sin que necesariamente sea debido a que otro objeto invoque uno de sus servicios.

2.3.3.3 Metodologías de Diseño para Sistemas Multiagente

Una metodología que permita tanto el diseño como la especificación de sistemas de agentes debe proporcionar un marco de trabajo conceptual claro que permite manejar la complejidad del sistema mediante abstracción y descomposición.

Extensiones de Metodologías Orientadas a Objetos

Las metodologías orientadas a objetos [Jac99] abogan por una descomposición del sistema identificando las clases objeto clave en el dominio de la aplicación, centrándose en su comportamiento y sus relaciones con otras clases. Los detalles esenciales del diseño del sistema son capturados utilizando tres tipos de modelos distintos [Kin96]:

1. *Modelo de Objetos*. Captura la información de los objetos dentro del sistema describiendo la estructura de sus datos, sus relaciones y las operaciones que permiten.

2. *Modelo Dinámico*. Describe los estados, transiciones, eventos, acciones, actividades e interacciones que caracterizan el comportamiento del sistema.

3. *Modelo Funcional*. Describe el flujo de datos durante la actividad del sistema, tanto dentro de un mismo componente como entre distintos componentes.

La principal ventaja de la utilización de metodologías que extienden el paradigma de orientación a objetos estriba en las similitudes que presentan agentes y objetos. Los agentes pueden ser considerados como objetos activos con un estado mental. Los dos paradigmas utilizan paso de mensajes para comunicarse y pueden utilizar la herencia y la agregación para definir su arquitectura [Igl99]. La principal diferencia radica tanto en las restricciones que se aplica a los tipos de mensajes en el paradigma orientado a agentes, así como en la definición del estado del agente, basado en sus creencias, deseos e intenciones *(Belief-Desire-Intention)* [Woo00a].

La popularidad de las metodologías orientadas a objetos es otra de las ventajas. Actualmente se están utilizando distintas metodologías orientadas a objetos en la industria con éxito, como por ejemplo: *Object Modeling Technique* (OMT) [Der98], *Object Oriented Software Engineering* (OOSE) [Jac92] o el Proceso Unificado de Desarrollo [Jac99]. Esta experiencia puede ser fundamental para facilitar la integración de la tecnología de agentes, ya que los ingenieros del software no tendrían que aprender una nueva metodología desde cero, y además en una empresa siempre se preferirá utilizar una metodología que ya ha sido sobradamente probada con éxito dentro del campo industrial.

La metodología Tropos

Tropos [Bre04a][Bre04b] es una metodología para el desarrollo de sistemas software basados en agentes, y para las distintas fases de desarrollo utiliza la notación I*[Yu 97].

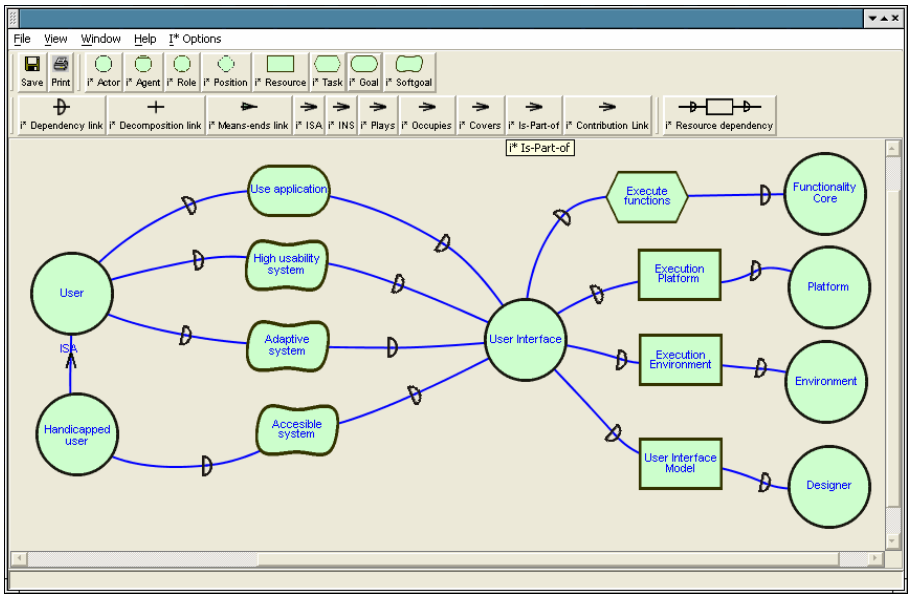


Figura 2.25 Entorno de modelado de la metodología Tropos.

El método propone cuatro fases de desarrollo, que cubren desde las primeras fases de adquisición de requisitos al diseño. Sin embargo, no cubre los aspectos de implementación, ni existen herramientas actualmente que permitan la generación a partir de los modelos creados. Sin embargo, sí que existe una herramienta que permite especificar visualmente los modelos de las fases de desarrollo que sí cubre (véase la figura 2.25).

Existen distintas variantes de la notación I*, la más extendida es la variante GRL [Yu 04], que a continuación se describe detenidamente.

Goal-oriented Requirement Language

Goal-oriented Requirement Language (GRL) es un lenguaje que permite el modelado de los requisitos siguiendo el paradigma de orientación a objetivos. La integración explícita de la representación de objetivos en los modelos de requisitos permite añadir un criterio de completitud a los requisitos – los requisitos han sido completados si son suficientes para mantener el objetivo que refinan [Yue87]. Dentro del marco de *Formal* [Dar91], los refinamientos de los objetivos son capturados usando estructuras de grafos con relaciones AND/OR. Los enlaces AND de un objetivo con un conjunto de subobjetivos representa que si todos los subobjetivos se dan, entonces el objetivo se ha alcanzado. Por otro lado,

los enlaces OR enlazan un objetivo con distintas posibilidades para conseguir alcanzar un objetivo. Otro tipo de objetivo introducido en [Myl92], denominado *softgoal,* representa objetivos para los que no existe una condición clara para decidir si dichos objetivos se han alcanzado o no. GRL toma estas ideas proporcionando tanto una notación XML como una notación gráfica. GRL está compuesto por elementos intencionales y relaciones entre dichos elementos. Entre los elementos intencionales encontramos: *goals* (objetivos), *tasks* (tareas), *softgoals* (objetivos no funcionales), *beliefs* (creencias) y recursos. Entre las relaciones se encuentran: *means-ends* (medios y fines), descomposición, contribución, correlación y dependencia. En este tipo de modelado el diseñador se ocupa principalmente de definir por qué se toman ciertas decisiones sobre el comportamiento y/o la estructura del sistema, pero no está en esta fase todavía interesado en la descripción de los detalles operacionales de los procesos.

Elementos del lenguaje GRL

- ***Goal* (objetivo)**: un objetivo es una condición o estado en el mundo que los actores involucrados desean alcanzar. Sin embargo, no se especifica cómo se debe alcanzar el objetivo, permitiendo de esta forma que distintas alternativas puedan ser consideradas.

- ***Softgoal* (objetivo no funcional)**: un objetivo no funcional es un objetivo cuya satisfacción o cumplimiento no puede ser establecido de manera absoluta, es decir, que en algunos casos solamente se podrá afirmar que se ha cumplido (o incumplido) de manera parcial.

- ***Task* (tarea)**: es una tarea específica, una manera concreta de realizar algo. También puede verse como una solución en el sistema destino para la descripción de cómo se debe satisfacer un objetivo o un objetivo no funcional.

- **Recurso**: un recurso es una entidad (física o no), cuya mayor influencia es si está disponible o no.

- ***Belief* (creencia)**: las creencias permiten representar el por qué de un diseño. Las creencias permiten considerar las características del dominio en el proceso de decisión, para de

esta forma facilitar posteriores revisiones o justificar los cambios al sistema.

- **Actor**: un actor es una entidad que lleva a cabo acciones para alcanzar objetivos haciendo uso de su conocimiento sobre cómo realizarlas.

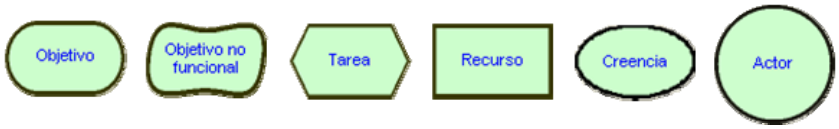


Figura 2.26 Representación gráfica de los elementos del lenguaje GRL.

Relaciones intencionales en GRL

La estructura del modelo es creada en forma de enlaces que representan relaciones intencionales entre los elementos de GRL anteriormente descritos. A continuación se describen los distintos tipos de relaciones que pueden establecerse.

- **Relaciones de medios-fines**: este tipo de relaciones es utilizado para describir cómo se deben alcanzar los objetivos definidos. En la notación gráfica de GRL, este tipo de enlaces une un nodo con los medios disponibles para alcanzarlo.

- **Relación de descomposición**: permite descomponer los elementos necesarios para alcanzar un objetivo o tarea.

Tabla 2. Tipos de relaciones de contribución en GRL.

Nombre	Contribución
AND	La contribución de cada uno de los elementos es positiva y necesaria.
OR	La contribución de cada uno de los elementos es positiva y suficiente.
MAKE	La contribución del elemento es positiva y suficiente.
BREAK	La contribución del elemento es negativa y suficiente.
HELP	La contribución del elemento es positiva pero insuficiente.
HURT	La contribución del elemento es negativa pero insuficiente.
SOME+	La contribución del elemento es positiva pero su aportación desconocida.
SOME-	La contribución del elemento es negativa pero su aportación desconocida.
EQUAL	Existe una aportación igual en ambos sentidos.
UNKNOWN	Existe una contribución, pero tanto la aportación como el sentido (positivo o negativa) son desconocidos.

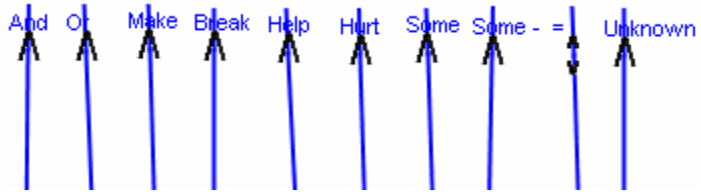


Figura 2.27 Representación gráfica de los distintos tipos de contribuciones.

- **Relación de contribución**: describe cómo unos elementos contribuyen a otros. Por ejemplo, se puede describir cómo una tarea contribuye a la consecución de un objetivo. Esta contribución no necesita ser necesariamente positiva, sino que también puede ser negativa, o incluso desconocida. En la tabla 2 se muestran los distintos tipos de contribuciones posibles y en la figura 2.27 se describe la notación gráfica asociada a cada tipo de contribución.

- **Relación de dependencia**: una relación de dependencia establece una relación entre dos actores o entre un actor y cualquier otro elemento de GRL (por ejemplo un recurso) (véase la figura 2.28).

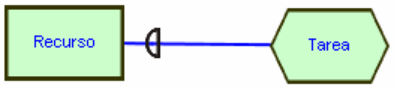


Figura 2.28 Representación gráfica de una dependencia entre una tarea y un recurso.

- **Relación de correlación**: una relación de correlación expresa una interacción entre dos elementos. Una relación de correlación es similar a una relación de contribución, aunque en este caso la contribución no es fruto de un deseo explícito de que exista, si no que es un efecto colateral. Su representación gráfica es similar a la de las relaciones de contribución, aunque en este caso las líneas que unen los elementos implicados son dibujadas con línea discontinua. Las relaciones de correlación se representan gráficamente de forma análoga como se representan las relaciones de contribución (véase la figura 2.27), aunque en este caso la línea que une los dos nodos implicados será discontinua.

En Tropos el método comienza con la etapa de adquisición de requisitos tempranos, donde se describen las razones que han conducido al

desarrollo del sistema, centrándose por lo tanto en los actores involucrados en el sistema y sus intenciones. Las intenciones son modeladas como objetivos, mientras que los actores involucrados son representados como actores de I*. Los actores dependen unos de otros para conseguir alcanzar las metas, realizar las tareas o conseguir un recurso. La descripción de los requisitos tempranos se realiza usando un modelo estratégico de dependencias. El modelo estratégico de dependencia es un grafo donde se describen los actores y las dependencias entre dichos actores. Cada dependencia especifica un acuerdo entre dos actores, donde el tipo de dependencia define el tipo de acuerdo. Las dependencias de los objetivos, y de los objetivos no funcionales *(softgoals)*, representan delegación de responsabilidades para alcanzar una meta. Las dependencias de tareas especifican que un actor debe realizar una actividad. Finalmente, las dependencias de recursos especifican que el actor necesita un recurso.

La siguiente etapa es el análisis de requisitos tardíos. En esta etapa se describen todos los requisitos funcionales y no funcionales del futuro sistema. Esta etapa se realiza especificando un modelo estratégico de dependencias en el que participan uno o más actores que representan el sistema. La especificación de cómo los objetivos identificados en la etapa de requisitos tempranos pueden ser alcanzados se realiza usando el modelo estratégico de motivaciones *(rationale)*. Un modelo estratégico de motivaciones es un grafo que contiene nodos de tipo objetivo, objetivo no funcional, recurso y tarea, relacionados mediante enlaces de descomposición y de medios-fines. Este modelo describe las relaciones entre las metas de cada actor y las dependencias a través de las cuales el actor espera alcanzar sus objetivos.

A continuación se realiza un diseño arquitectónico que describe como los distintos componentes del sistema colaboran para alcanzar los objetivos del sistema. En Tropos este diseño de basa en una serie de estilos arquitectónicos organizacionales [Kol02], especialmente dirigidos al diseño de sistemas multi-agente.

La última etapa descrita en Tropos es el diseño detallado, donde se describe en detalle cada uno de los componentes de la arquitectura. En esta etapa se especifica cómo los agentes llevan a cabo las metas asignadas a cada actor. Ello lleva consigo también la especificación de la comunicación y el comportamiento de los agentes.

La principal aportación de Tropos es el comienzo del desarrollo a partir de la adquisición de los requisitos tempranos, captando las razones que conducen a la definición del sistema usando una aproximación guiada por objetivos, sin embargo la propuesta es poco detallada en su fase de diseño e implementación.

La Metodología AAII

La metodología AAII ha sido desarrollada por el Instituto Australiano de Inteligencia Artificial (AAII) basada en la experiencia adquirida en el desarrollo de distintos proyectos de gran envergadura, como el sistema OASIS [Gab94] para el control del tráfico aéreo del aeropuerto de Sidney. Basada en las metodologías orientadas a objetos actuales, las enriquece añadiéndoles algunos de los conceptos de los sistemas basados en agentes. El objetivo de la metodología es la construcción de un conjunto de modelos, que totalmente completados, definen la especificación del sistema de agentes [Kin96].

La metodología define dos niveles de abstracción, descomponiendo el sistema desde un punto de vista externo y desde un punto de vista interno.

Desde el punto de vista externo, la metodología propone una descomposición del sistema basada en los papeles más significativos en la aplicación. La identificación de los papeles y sus relaciones guía la especificación de la jerarquía de clases de los agentes. Los agentes serán en consecuencia instancias concretas de dichas clases. El análisis de las responsabilidades de cada clase de agente conduce a la identificación de los servicios que ofrecen, así como de aquellos que usan de otros agentes. Ello permite identificar las interacciones externas de los agentes. Otros aspectos a tratar son tanto la creación y duración de los papeles, como de sus interacciones para determinar las relaciones de control entre las distintas clases de agentes.

Todos estos detalles son capturados utilizando dos modelos:

1. El *Modelo de Agente* que describe las relaciones jerárquicas entre distintas clases de agentes, e identifica las instancias de los agentes que pueden existir dentro del sistema, su multiplicidad, y cuando son creadas.

2. El *Modelo de Interacción* describe las responsabilidades de una clase de agente, los servicios que ofrece, sus interacciones asociadas, y las relaciones de control entre las clases de agentes.

Desde el punto de vista interno, la metodología se ajusta al paradigma BDI *(Beliefs-Desires-Intentions)*. De acuerdo a este modelo los agentes poseen ciertas actitudes mentales que representan la información que poseen (creencias), sus motivaciones (deseos) y sus deliberaciones (intenciones). Los agentes son descritos de acuerdo a los eventos que perciben, las acciones que realizan, las creencias que poseen, las metas que adoptan, y los planes que definen sus intenciones.

Los puntos anteriormente descritos son modelados utilizando tres tipos de modelos:

- El *Modelo de Creencias (Beliefs)* describe la información sobre el entorno y el estado interno de una clase de agentes, así como las acciones que puede realizar. Se describen además las creencias de agentes y sus propiedades. Las propiedades de las creencias se engloban en conjuntos de creencias *(belief set)*. Las instancias de los conjuntos de creencias son denominadas estados de las creencias *(belief states)*, que pueden ser utilizados para especificar el *estado mental* inicial del agente.

- El *Modelo de Metas (Goals)* describe las metas que un agente puede adoptar, y los eventos a los que responde. También existen los conjuntos de metas *(goal set)* que especifican la meta, el dominio del evento y uno o más estados de metas *(goal states)* que es utilizado en la especificación del estado *mental inicial* del agente.

- El *Modelo de Planes (Plans)* describe los planes que un agente podría utilizar para alcanzar sus metas. Las propiedades y la estructura de control de planes individuales son agrupadas en conjuntos de planes *(plan set)*.

La Metodología Gaia

Gaia [Woo00b] es una metodología orientada a sistemas multiagente agentes donde se asumen ciertas restricciones sobre el sistema. En primer lugar se supone que los agentes son sistemas computacionales con una cierta granularidad, por lo que hacen uso de una cantidad de recursos

significativa. En segundo lugar asume que se intenta maximizar alguna característica, es decir, no está diseñada para sistemas donde cada agente juegue por su propio interés. En tercer lugar, está pensada para sistemas que contengan un número relativamente pequeño de agentes (menos de 100).

Se pretende crear una metodología donde el analista pueda llegar de forma sistemática desde los requisitos a un diseño lo suficientemente detallado para que pueda ser implementado directamente.

Se realiza una división de los conceptos clasificándolos en abstractos y concretos. La principal diferencia radica en que los conceptos abstractos, aunque usados en el análisis, no tienen porque tener un reflejo real en el sistema final. Mientras que los concretos si que tendrán una representación directa en el diseño final. Para ello se propone la jerarquía mostrada en la figura 2.29.

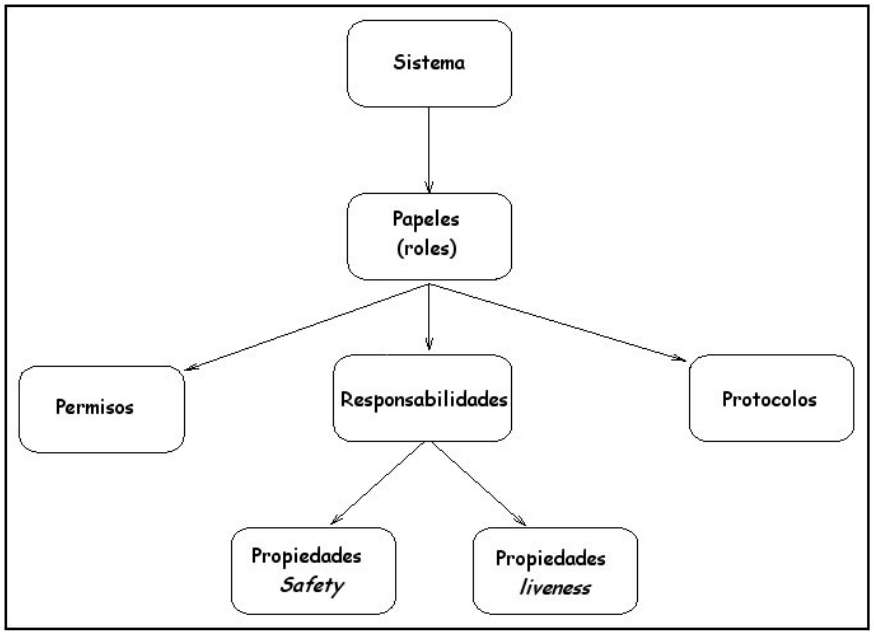


Figura 2.29 Conceptos para el Análisis en Gaia.

La idea más abstracta es la entidad *sistema*. Aunque conserva las habituales connotaciones de la palabra sistema, también debe entenderse desde el punto de vista de las sociedades artificiales y organizaciones que se dan en los sistemas multiagente. De la idea de un sistema como una sociedad de

agentes surge el concepto de *papel* (*rol*) – el concepto de papel debe entenderse dentro del contexto de una obra de teatro o una película. Los *papeles* que desempeñan los distintos agentes componen el sistema. Estos *papeles* podrán ser instanciados tanto por un solo agente como por varios, adicionalmente un agente podrá ejercer varios *papeles*, tal y como ocurre en la vida real en una empresa, donde una mismo persona desempeña muy a menudo distintos *papeles*.

En la propuesta inicial de la metodología [Woo99] se definían los *papeles* usando los atributos de *responsabilidad*, *permisos*, y *protocolos*. Finalmente, se comprobó que era necesario añadir también el concepto de *actividades*, para modelar los procesos que pueden ocurrir dentro de un agente sin que se produzca ninguna interacción con otros agentes. De alguna manera se trata de añadir algo parecido al concepto de método privado en el paradigma de orientación o objetos.

Las *responsabilidades* de un *papel* indican la funcionalidad de este. Están divididas en dos tipos: las propiedades *safety* y las propiedades *liveness*. Las propiedades *safety* indican condiciones que debe respetarse a lo largo de toda la existencia del *papel*. Las propiedades *liveness*, por el contrario, definen estados que el agente debe adoptar dadas unas condiciones ambientales determinadas.

Sin embargo, para llevar a cabo las *responsabilidades* que se le encomiendan a un *papel* es necesario adjudicarle a ese *papel* los *permisos* necesarios sobre los recursos. Una de las limitaciones actuales de esta metodología es que limita los *permisos* a fuentes de información.

Finalmente, los *protocolos* definen la forma es que un *papel* interacciona con otros *papeles*.

De la descripción de los *papeles* se deriva la necesidad de dos modelos distintos dentro de la organización en Gaia: el *modelo de papeles* y el *modelo de interacción*.

El *modelo de papeles* describe de forma abstracta las funcionalidades que se espera que posea un *papel*. El *modelo de interacción* por su parte representa las dependencias y relaciones que existen entre los distintos *papeles* de un sistema multiagente.

Una vez definidos los modelos anteriores hay que realizar la etapa de diseño para transformas los modelos definidos en la etapa de diseño en modelos con un nivel de abstracción lo suficientemente bajo para que puedan ser implementados con facilidad. En Gaia se realiza generando tres modelos distintos: el *modelo de agente (agent model)*, el *modelo de servicios (services model)* y el *modelo de conocidos (acquaintance model)*. El *modelo de agente* define los tipos de agentes que compondrán el sistema, así como las instancias que serán creadas para estos tipos de agentes. El *modelo de servicios* identifica los servicios que son necesarios para que el agente interprete el *papel*. Para finalizar, el *modelo de conocidos* modela las líneas de comunicación entre los distintos agentes. La figura 2.30 muestra los distintos modelos dentro de Gaia así como sus relaciones.

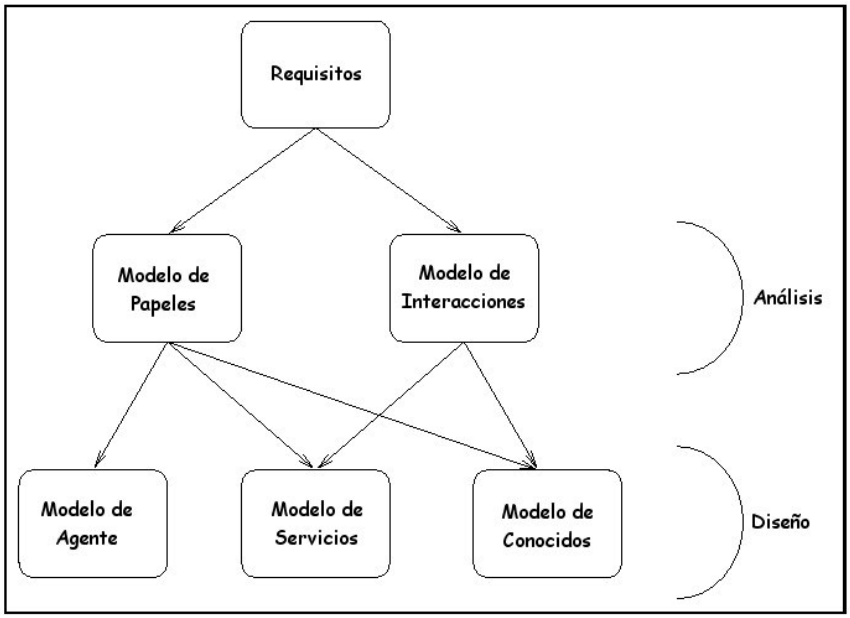


Figura 2.30 Relaciones entre los distintos modelos en Gaia.

UML para Agentes

En la última década se han producido una gran cantidad de propuestas metodológicas para el modelado de sistemas bajo el paradigma de la orientación a objetos. Sin embargo, el lenguaje de especificación que se ha impuesto como estándar ha sido UML [Boo99]. UML no es en sí una metodología. La metodología propuesta basada en UML es el Proceso Unificado de Desarrollo [Jac99]. Siendo UML el lenguaje más extendido

en el campo de la especificación, no es extraño que tanto la FIPA *(Foundation for Intelligent Physical Agents)* como la OMG *(Object Management Group)* estén impulsando la extensión de UML para su utilización con agentes.

Odell y otros [Ode00] han propuesto extensiones a UML, principalmente modificaciones de los diagramas estándar aprovechando las posibilidades, que como sistema abierto, ofrece UML. Para ello han modificado distintos tipos de diagramas, como los casos de uso o los diagramas de secuencia y colaboración. Bauer y otros [Bau01a][Bau01b] proponen la adaptación de UML para la especificación de agentes basándose en la extensión del concepto de *Objeto Activo* que aparece en UML.

Los principales escollos que hay que superar para adaptar UML a los agentes son conseguir que los agentes sean capaces de procesar peticiones externas, así como plasmar los conceptos *sociales* que aparecen en las sociedades artificiales de los sistemas multiagente.

Metodología Orientada a Agentes para Modelado de Empresas

Kendall y otros [Ken95] proponen una metodología basada en el paradigma de orientación a objetos Se apoyan para ello en la definición de modelado y análisis de flujo de trabajo *(workflow)* presentada por el IDEF *(Integration Definition for Function modelling)*, y el marco de trabajo para modelado empresarial CIMOSA *(Computer Integrated Manufacturing Open System Architecture)*.

El *modelo de funciones* describe las funciones utilizando diagramas $IDEF_0$ que contienen la selección de los distintos métodos dependiendo de las entradas y del control. El *modelo de casos de uso* describe qué actores intervienen en cada función. Para ello utiliza los diagramas habituales de casos de uso del paradigma de orientación a objetos. El *modelo dinámico* representa las interacciones entre los distintos objetos.

El sistema será una composición de una identificación de agentes, unos protocolos de coordinación, unos planes, así como de un modelo de entradas donde las creencias *(beliefs)* se obtienen de objetos utilizando sensores, y las metas alcanzadas son modeladas como cambios en las creencias o modificaciones a través de efectores.

Especificación de Agentes en OASIS

Oasis [Let98] *(Open and Active Specification for Information Systems)* es una aproximación formal basada en el paradigma de orientación a objetos para el modelado conceptual. En Oasis los sistemas de información son entendidos con una sociedad de objetos autónomos concurrentes que interaccionan entre ellos ejecutando acciones.

Una especificación en OASIS consiste en una estructura de definiciones de clases. Dichas clases pueden ser simples o complejas. Una clase compleja será una clase que es definida utilizando otras clases simples o complejas, a través de relaciones de agregación o herencia. Un objeto es un proceso observable, y que tiene una vida caracterizada por la ocurrencia de acciones que pueden ser servicios que se proporcionan o peticiones.

Los servicios proporcionados por el objeto, de forma atómica, son denominados eventos. Cada objeto tiene un evento de creación, que determina el comienzo de su vida, y opcionalmente uno de destrucción, que finaliza su vida. Además de la semántica utilizada para especificar procesos, también existe una semántica para determinar la distinción entre los protocolos y las operaciones. Una operación es un servicio que se oferta a otro objeto a un nivel más alto. Una transacción es una operación que contiene implícitamente la política todo o nada. Los protocolos no permiten la ejecución de ciertas secuencias de acciones durante la vida del objeto.

La visibilidad entre los distintos objetos de la sociedad de objetos descrita es determinada utilizando un mecanismo de interfaces. Cada objeto encapsula tanto su estado como sus reglas de comportamiento. El estado del objeto en un momento concreto son los valores de las propiedades del objeto, y tal como ocurre en la filosofía de orientación a objetos, dicho estado varía a través del tiempo debido a las acciones que se producen. La actividad de los objetos puede ser descrita más concretamente utilizando precondiciones, restricciones de integridad, disparadores *(triggers)*, protocolos y operaciones. Cada objeto tendrá u identificador único asignado por el sistema, y que no podrá ser reutilizado incluso cuando el objeto haya sido destruido. OASIS también incorpora el concepto de *metaclase*. La metaclase permite la creación de los distintos tipos de clases en OASIS (véase la figura 2.31) y permiten la modificación de las características de una clase cambiando el estado de los *metaobjetos* que las

componen. Las metaclases son el mecanismo utilizado en OASIS para la evolución del esquema.

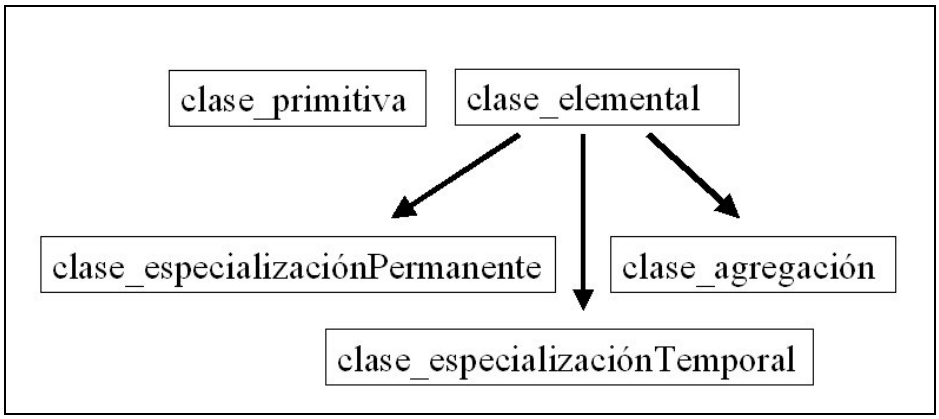


Figura 2.31 Tipos de clases en OASIS.

Dentro de este marco, en [Ram01] se propone la modelización del concepto de agente de acuerdo al modelo de ejecución de procesos de OASIS. Para ello se realiza una especificación del concepto de agente de acuerdo a la definición de agente proporcionada por Michael Wooldridge y Nick Jennings [Woo95] de acuerdo a los atributos: autonomía, reactividad, sociabilidad e iniciativa (pro-actividad).

- La autonomía es modelada considerando al agente como un proceso continuo y concurrente que actúa sobre su entorno comunicándose con el resto de agentes y herramientas.

- El concepto de reactividad es muy similar al concepto de orientación a objetos en el que un objeto ejecuta una acción en respuesta a una petición, por lo que su especificación en OASIS es análoga.

- La sociabilidad consiste en que los agentes deben comunicarse entre sí. Este concepto es muy similar al paso de mensajes tradicional en orientación a objetos. Para especificarlo se debe hacer uso de la metaclase, ya que es la única que conoce la estructura de clases y puede actuar como un enrutador para los mensajes entre los distintos agentes.

- El concepto de iniciativa (pro-actividad) implica que el agente es capaz de ejecutar acciones sin que ningún objeto le haga una petición. Esto marca una clara diferencia entre el concepto de agente y el concepto de objeto, donde la ejecución de un método se realiza como respuesta a una petición de otro objeto. Para especificar este concepto se utiliza el concepto de vista que determina qué parte del entorno que lo rodea es capaz de observar el agente.

Las propiedades que describen el estado mental del agente son especificadas de acuerdo al modelo BDI *(Beliefs, Desires, Intentions)*.

Extensiones de Metodologías de Ingeniería del Conocimiento

Existen, como hemos visto, múltiples aproximaciones desde el punto de vista del paradigma de la orientación a objetos. Sin embargo, no debemos olvidar la clara componente de inteligencia artificial que presenta el mundo de los agentes. Por ello también se puede abordar el problema metodológico de los sistemas multiagente desde el punto de vista de la ingeniería del conocimiento.

La ingeniería del conocimiento aporta una amplia experiencia en el desarrollo de sistemas donde se presentan características de inteligencia artificial, y especialmente sistemas en los que se plantean problemas de aprendizaje, como los planteados en los sistemas multiagente.

Intentaremos a continuación describir algunas de las soluciones metodológicas para el desarrollo de sistemas multiagente, que han aparecido extendiendo de alguna manera metodologías de ingeniería del conocimiento.

Prometheus: Un Método de Diseño de Sistemas Multiagente

Prometheus [Pad02] es una metodología para la creación de sistemas multi-agente que describe cómo crear el sistema multi-agente desde las primeras fases al diseño detallado.

El método propone las siguientes etapas dentro del ciclo de vida de diseño:

- Especificación del sistema

- Determinar las interfaces externas: acciones, percepciones y datos necesarios y producidos por las funcionalidades del sistema. Las acciones representan aquellas cosas que los agentes hacen y que afectan su entorno. Las percepciones describen aquellos estímulos del exterior que son significativos para el sistema.

- Determinar las metas del sistema, sus funcionalidades y escenarios.

- Diseño arquitectónico
  - Definir los agentes, los incidentes y las interacciones. Compartir las estructuras de datos.

- Diseño detallado
  - Definir las capacidades, planes, estructura de las creencias y los eventos.

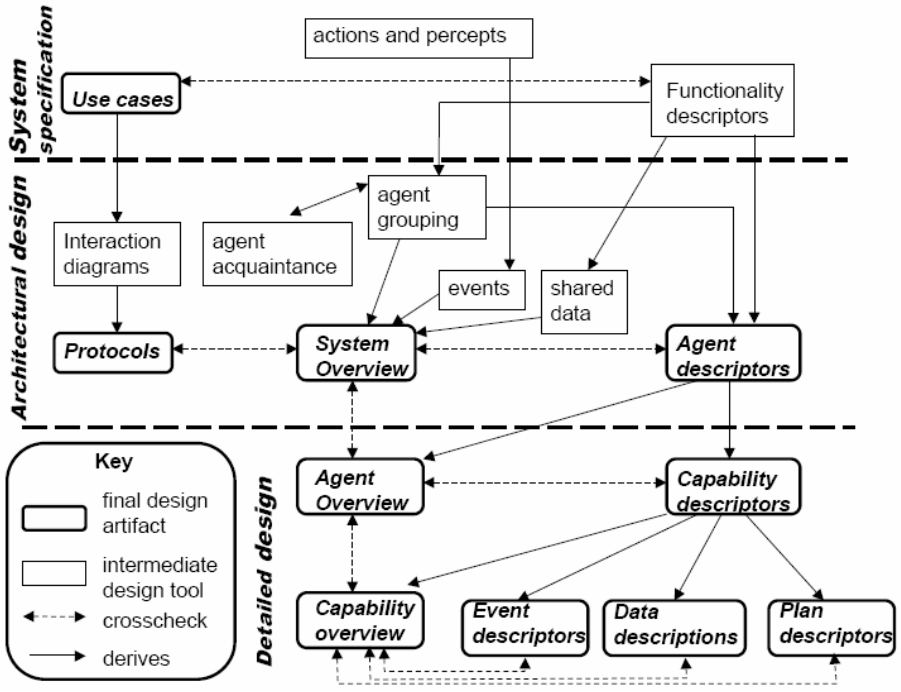


Figura 2.32 Ciclo de desarrollo propuesto en Prometheus [Pad02].

Diseño arquitectónico en Prometheus

El diseño arquitectónico comienza definiendo los agentes que compondrán el sistema. Para ello es necesario agrupar las funcionalidades identificadas de acuerdo a la relación que exista entre las distintas funcionalidades (funcionalidades relacionadas deberían estar unidas). De igual manera aquellas funcionalidades que compartan datos también deberían ser agrupadas en un mismo agente. Para evitar sobrecargar las comunicaciones entre los agentes, las funcionalidades que interacciones mucho deben ser agrupadas en un mismo agente. Prometheus propone guiar al diseñador en la agrupación de las funcionalidades mediante el diagrama de acoplamiento, donde se muestran las relaciones entres los distintos componentes.

De igual forma, se propone en Prometheus un diagrama de conocidos *(acquaintance)* donde se describen que agentes conoce cada agente. Es interesante que cada agente tenga que relacionarse con el número mínimo de agentes para simplificar la comunicación.

Una vez identificados los agentes se debe especificar qué agente responderá a cada percepción, y cada acción.

La interacción entre los agentes es descrita a través de escenarios, es decir, una descripción textual que describe la interacción y un diagrama de secuencia adaptado para la especificación de agentes [Bau01a] que lo especifica. Las secuencias de interacciones permitidas son descritas mediante protocolos.

Diseño detallado

El diseño detallado en Prometheus describe con precisión la estructura de los agentes. Los agentes se describen especificando sus planes, los eventos a los que responde y los que produce, y finalmente, los datos que lee o escribe. Todo ello puede ser agrupado en capacidades, de forma que una misma capacidad pueda ser asignada a distintos agentes, de esta forma reutilizando el diseño.

Todo el proceso de desarrollo propuesto en Prometheus puede ser especificado utilizando la herramienta *Prometheus Design Tools*[14]. La estructura de la especificación del diseño detallado en Prometheus puede

[14] http://www.cs.rmit.edu.au/agents/pdt/

ser diseñada visualmente mediante la herramienta JDE *(Jack Development Environment)* [Bus99], que luego permite la generación de código para el lenguaje de programación de agente JACK[15].

Cassiopeia

Cassiopeia es una metodología desarrollada por Collinot y otros [Col96] con la que se pretende seguir un ciclo de vida de acuerdo a una visión ascendente *(bottom-up)*.

Una de las mayores dificultades que nos encontramos a la hora de desarrollar un sistema multiagente es lograr transformar una especificación global de una tarea colectiva en comportamientos individuales, que será lo que finalmente será asignado a los agentes para completar la tarea. Estas dificultades nos llevan hacia un problema de organización dinámica para alcanzar nuestros objetivos. Precisamente la noción de organización es el concepto sobre el que la metodología Cassiopeia está basada.

En Cassiopeia se propone un diseño en tres niveles de comportamientos: (1) papeles individuales, (2) papeles relacionales, (3) papeles organizacionales. El componente organizacional aparece debido a que la activación de un comportamiento para alcanzar un objetivo afecta a otros comportamientos activados por otros agentes, y a su vez es afectado por dichos comportamientos. Los papeles (*roles*) individuales son los comportamientos que los agentes individualmente son capaces de realizar. Los papeles relacionales describen cómo interaccionan entre sí los agentes, con respecto a las mutuas dependencias que surgen en sus papeles individuales. Finalmente, los papeles organizacionales describen cómo maneja el agente sus interacciones con otros agentes para mantener la organización.

Como primer paso se propone la elaboración de una lista de los comportamientos elementales necesarios para alcanzar la tarea colectiva que estamos estudiando. A continuación es necesario analizar la estructura organizativa que aparece de acuerdo a las dependencias entre los distintos comportamientos elementales. El siguiente paso intenta capturar los aspectos dinámicos de la organización. Es decir, se deben especificar los comportamientos que harán que los agentes gestionen tanto la formación,

[15] http://www.agent-software.com

como la duración y disolución de los distintos grupos de agentes que irán apareciendo.

Uno de los puntos a favor de Cassiopeia es el intento de aunar en una sola metodología tanto el diseño del sistema multiagente como los métodos de aprendizaje que el agente utiliza para conseguir su adaptación al medio. Para ello se ha creado una ampliación denominada *Andromeda* [Dro98] donde se intenta dar una solución a este problema.

Para utilizar aprendizaje automático distribuido es necesario definir el nivel al que se supone el aprendizaje, las tareas de aprendizaje y su evaluación, y finalmente los protocolos de aprendizaje. Andromeda realiza una descomposición del problema en metas, y no en comportamientos como ocurre en Cassiopeia. Para ello es necesario que el conocimiento de defina de acuerdo a las metas, y no de funcionalidades y comportamientos. Los comportamientos adaptativos o los comportamientos aprendidos deben ser definidos en función de sus metas.

La primera fase de Andromeda define las acciones individuales, habilidades, y papeles individuales. Para ello se utiliza une lenguaje para describir roles llamado IRL (*Individual Role Language*). A este nivel, la mayor diferencia con Cassiopeia es que aquí los papeles que puede desempeñar un agente se ven como planificadores dinámicos de metas. Para alcanzar estas metas, los agentes pueden crear nuevos comportamientos o ajustar otros ya existentes. Desde un punto de vista metodológico, el objetivo de esta fase es ayudar al diseñador en la definición de los papeles individuales permitiendo a los agentes aprender o ajustar dichos papeles.

A continuación se construyen las dependencias uno a uno entre los papeles individuales. En este caso se utiliza el lenguaje IDL *(Influences Definition Language)*. El beneficiario del aprendizaje que se produce en esta fase es un solo agente, como ocurría en la primera fase. Ahora el agente debe aprender, en IDL, las dependencias que los papeles con los que interacciona ejercen sobre él. En este caso, el interés metodológico es mantener sólo las dependencias que expresan la influencia entre dos agentes cuando desempeñan un determinado papel.

En el siguiente paso se crean las relacionen entre los papeles. El lenguaje para definir las relaciones entre los papeles es RRL *(Relational Roles*

Language). En esta fase la mayor diferencia con Cassiopeia es que las relaciones entre los papeles son definidas utilizando metas, lo que permite que las relaciones entre los papeles se aprendan en diferente contextos. Las metas que se persiguen cuando se aprenden las relaciones entre los papeles son:

1. Aprender como indicar y cuando una influencia entre dos papeles.

2. Aprender como elegir entre los distintos signos utilizados para indicar la influencia

3. Aprender como reaccionar ante un determinado signo, es decir, cómo seleccionar el papel individual apropiado.

En este caso se beneficia del aprendizaje todo un conjunto de agentes, y no sólo un agente como ocurría en las fases anteriores. Un grupo de agentes aprende a coordinarse de acuerdo a sus dependencias.

Es necesario a continuación crear los grupos de agentes. Para ello se utiliza el lenguaje GDL *(Group Definition Language)*. Como ocurre en Cassiopeia, un grupo de agentes se crea de acuerdo a las dependencias uno a uno y las correspondencias en la ruta del grafo de dependencias. Se trata de permitir que los agentes cooperen explícitamente identificando las pautas en las activaciones simultáneas de sus papeles individuales debido a sus interrelaciones. De nuevo en esta fase, el beneficiario del aprendizaje es un grupo de agentes.

Finalmente, de realiza la definición de los papeles organizacionales. En este caso se utilizará el lenguaje ORL *(Organizational Role Language)*. En este caso no se trata tan sólo de seleccionar los papeles individuales a partir de las interrelaciones entre los papeles, sino de identificar en qué situaciones pueden gestionar, tanto activa como dinámicamente, la selección colectiva de un grupo de papeles.

El beneficiario del aprendizaje son teóricamente todos los grupos de agentes. Sin embargo, para ajustar el comportamiento global del sistema multiagente el diseñador debería ser capaz de definir uno o más funciones de utilidad, lo que no siempre es posible. Esta es la razón de que en la

práctica los beneficiarios son los grupos de agentes para los que sí se puede definir dicha función de utilidad.

La idea básica en Andromeda es dejar que la experiencia del aprendizaje se produzca en cada una de fases de definición de Cassiopeia. El objetivo de la metodología es mantener la máxima flexibilidad en el proceso de diseño, de forma que sea posible introducir conocimiento en los distintos niveles, a la vez que se ayuda al diseñador con el sistema de aprendizaje para reducir la complejidad del diseño del sistema.

CommonKADS

CommonKADS [Sch94b] ha servido como base para la creación de metodologías para sistemas multiagente. Aunque esta metodología no fue diseñada específicamente para el desarrollo de sistemas multiagente, si que contiene el concepto de agente dentro de sus métodos, aunque no contempla las principales características achacables a un agente, como la iniciativa *(pro-activity)*, reactividad, sociabilidad o adaptación. El concepto de agente que parece en CommonKADS está muy encaminado a la comunicación entre el usuario y el sistema experto. A continuación se dará una rápida visión de la estructura básica que se propone en dicha metodología.

CommonKADS se creó como una revisión de la metodología KADS *(Knowledge Acquisition Data System)* financiada por la Unión Europea, y se ha convertido en un estándar de facto para la Ingeniería del Conocimiento. CommonKADS abarca todo el ciclo de desarrollo del software (se extiende no solamente a sistemas basados en conocimiento, sino al software en general) mediante siete modelos: Organización, Tareas, Agentes, Experiencia, Comunicación y Diseño.

El modelo de organización es utilizado para analizar la organización donde se va incluir el sistema. El modelo de tarea, por su parte, describe las tareas que deben ser realizadas dentro del entorno organizativo y sirve como punto de partida para la distribución de tareas entre los agentes. El modelo de agente es la entidad que realiza una tarea, y puede ser humano, software o cualquier otro tipo de entidad capaz de realizar una tarea. El modelo de comunicaciones precisa cómo se produce el intercambio de información entre los agentes involucrados en la ejecución de las tareas especificadas en el modelo de tareas. El modelo de experiencia es el núcleo básico de CommonKADS, y modela el conocimiento de

resolución de problemas utilizado por los agentes para realizar las tareas. Finalmente, el modelo de diseño describe la arquitectura y diseño del sistema como paso previo a su implementación.

Como ya sea ha comentado anteriormente, CommonKADS presenta ciertas limitaciones para su utilización en entornos multiagente, especialmente en los aspectos del modelo de agente, y en los modelos de comunicación y organización, que no están pensados para modelar el tipo de comunicaciones e interacciones que se producen dentro de un sistema multiagente.

Basándose en CommonKADS se han producido distintos intentos de incluir de forma sencilla los sistemas multiagente dentro de la metodología modificándola con nuevas características que permitan modelar los conceptos distintivos de los sistemas multiagente. Entre ellas podemos encontrar MAS-CommonKADS [Igl98] y CoMoMAS [Gla96].

MAS-CommonKADS

MAS-CommonKADS [Igl98] intenta suplir las carencias de CommonKADS. Para ello como primer paso propone la modificación del modelo de agente para permitir que se puedan especificar las características necesarias para el modelado de agentes: capacidades de razonamiento, habilidades, servicios, sensores, efectores, grupos de agentes y clases de agente. Ahora el modelo de organización no sólo se utiliza para analizar la organización en que se va a introducir el sistema, sino que también se usa para describir la organización de los agentes software y su relación con el entorno. El modelo de tareas contempla también los métodos de resolución de los problemas para cada objetivo. El modelo de experiencia describe el conocimiento necesario para que los agentes alcancen sus metas.

MAS-CommonKADS utiliza el modelo de comunicación para describir las interacciones entre los agentes humanos y los agentes software, mientras que para describir las interacciones entre los agentes software añade un nuevo modelo: el modelo de coordinación. El modelo de diseño al igual que ocurría en CommonKADS es utilizado para describir la arquitectura y el diseño del sistema multiagente como paso previo a la implementación. La figura 2.33 muestra las relaciones existentes entre los distintos modelos en MAS-CommonKADS.

MAS-CommonKADS propone combinar el análisis ascendente y el descendente. El análisis descendente permite reutilizar y diseñar, mientras que el análisis ascendente permite reutilizar y diseñar. La identificación de los agentes utiliza un análisis ascendente utilizando casos de uso internos, mientras que el análisis descendente es utilizado para desarrollar el modelo de tareas.

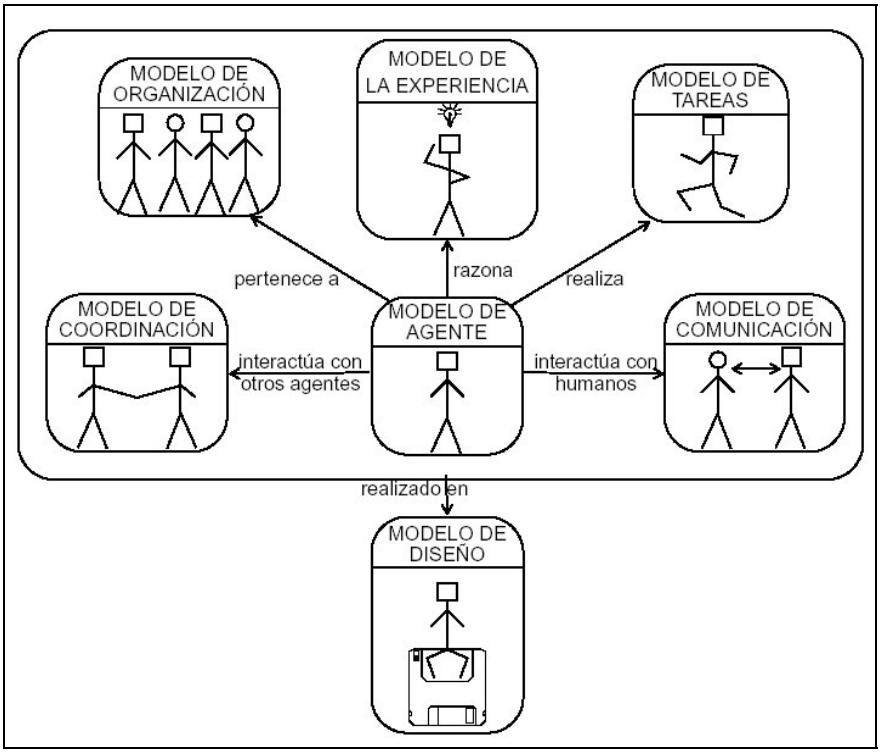


Figura 2.33 Modelos de MAS-CommonKADS [Igl98].

Después de la fase de análisis se procede a una fase de diseño, la cual se subdivide en tres actividades distintas:

- Diseño de la aplicación: se descompone el sistema en subsistemas. Para una arquitectura multiagente se define la arquitectura más adecuada para cada agente involucrado en el sistema.

- Diseño de arquitectura: se selecciona una arquitectura multiagente. Se determina la infraestructura del sistema multiagente (modelo de red) y los agentes que mantendrán dicha infraestructura.

- Diseño de la plataforma: se define qué software y qué hardware será necesario para desarrollar el sistema.

Para la codificación se puede utilizar un lenguaje de propósito general o un lenguaje formal específico para sistemas multiagente, con la posibilidad de que dicho lenguaje sea directamente ejecutable mediante traducción o un intérprete.
Dada la naturaleza no determinista de los sistemas multiagente, no es posible determinar en la fase de análisis la conducta global del sistema, ya que dependerá de los acuerdos a los que vayan llegando los agentes durante la ejecución. Por ello se suele recurrir a la simulación.

Una de las grandes ventajas de los sistemas multiagente es que el mantenimiento se facilita por el carácter modular intrínseco a los sistemas multiagente.

MAS-CommonKADS propone una metodología completa y que ha sido probada con éxito en distintos proyectos.

CoMoMAS

CoMoMAS [Gla96] es una aproximación que incluye las estructuras del conocimiento de la reactividad, cognitivas, de cooperación y las habilidades sociales de los agentes autónomos. La metodología respeta la autonomía de cada agente, los cuales están compuestos de cinco modelos distintos que representan el conocimiento necesario para la creación del sistema multiagente. La autonomía en CoMoMAS se entiende en el sentido en que cada agente tiene sus habilidades y su conocimiento específico, es capaz de actuar independientemente y toma decisiones por sí mismo.

En esta metodología los agentes se describen como entidades autónomas e inteligentes que actúan por sí mismas usando su conocimiento individual. Las cuales, además, crean sociedades con otros agentes. Cada uno de estos agentes es capaz de realizar planificaciones, navegar, resolver tareas y posee algún tipo de mecanismo de seguridad. Adicionalmente, recogen y actualizan su conocimiento en forma de planes, comportamientos reactivos, mapas del mundo donde están inscritos y comportamientos. Como parte activa de un sistema multiagente, necesitan capacidades sociales para cooperar, comunicarse e interaccionar con sus compañeros. El agente tiene cuatro tipos de conocimiento que

representan sus capacidades de reacción, cognitivas, de cooperación y sociales. Por otro lado un agente tiene conocimiento de control para describir sus mecanismos internos de control, para especificar cómo como comportarse, sus métodos de resolución de problemas (PSM – *Problem Solving Methods)* y las estrategias para el trabajo conjunto con otros agentes para alcanzar las metas propuestas.

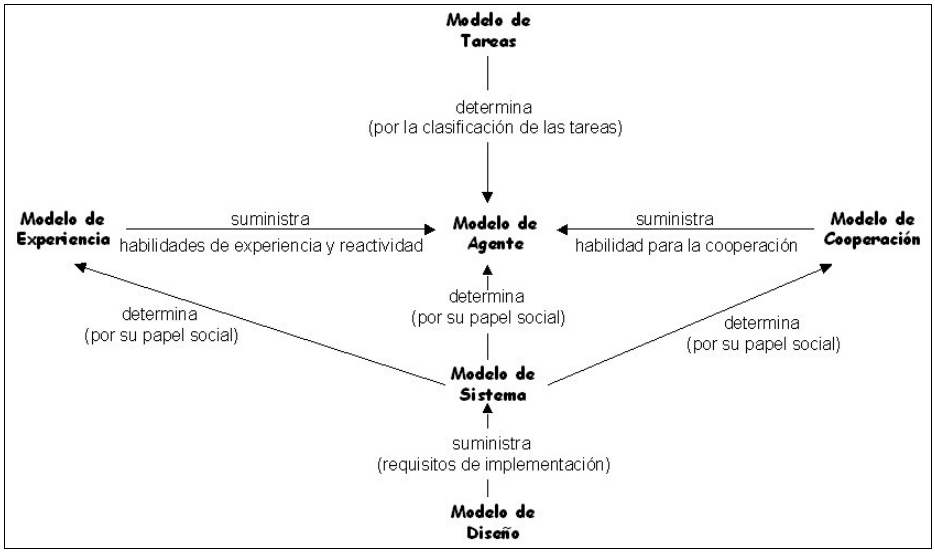


Figura 2.34 Modelos para el desarrollo de sistemas multiagente en CoMoMAS.

Para realizar la fase de análisis la metodología propone cinco tareas:

1. Es necesario realizar un análisis funcional para identificar las tareas que debe resolver el sistema multiagente. Se construye una jerarquía con las tareas a las que pueden dar una solución tanto el usuario como el agente, dando forma a lo que se llama el modelo de tareas.

2. El análisis de requisitos determina los requisitos de diseño del sistema multiagente. Incluye una lista de los requisitos, la determinación de sus dependencias y las prioridades.

3. El análisis de capacidades identifica las habilidades cognitivas y reactivas que el sistema debe proporcionar para resolver las tareas especificadas en los pasos anteriores.

4. El análisis de cooperación identifica los protocolos de cooperación, los métodos de cooperación y los métodos de resolución de conflictos para los agentes. Una vez identificadas las habilidades necesarias para alcanzar una meta, es necesario especificar qué medios serán usados para la interacción entre los agentes.

5. El análisis social identifica la organización y la arquitectura del sistema multiagente y sus agentes, además de las capacidades sociales para que los agentes actúen de forma correcta durante la cooperación y la interacción. El análisis social persigue la elaboración del modelo de sistema.

El resultado de la fase de análisis es la elaboración de cinco modelos conceptuales que son utilizados para crear los modelos de agente. La figura 2.34 muestra los distintos modelos de CoMoMAS, así como las relaciones que existen entre los distintos modelos. Las mayores diferencias entre CoMoMAS y CommonKADS se encuentran tanto en el modelo de agente como en el modelo de experiencia.

A continuación se describen los modelos y la aplicación que de dichos modelos propone CoMoMAS. El modelo de agente define la arquitectura del agente y el conocimiento que posee el agente (social, de cooperación, cognitivo, reactivo y de control). El modelo de experiencia por su parte describe las habilidades cognitivas y reactivas del agente. El modelo de tareas describe cómo se deben descomponer las tareas, indicando si dichas tareas pueden ser realizadas por el usuario o por el agente. El modelo de cooperación especifica cómo se produce la cooperación entre los agentes de la sociedad de agentes. En este modelo se especifican los métodos de negociación y cooperación. El modelo de sistema presenta cómo se organiza la sociedad de agentes y describe la arquitectura del sistema multiagente y de cada uno de los agentes que lo componen. Finalmente, el modelo de diseño describe como pasar de los modelos de las fases de análisis a un código ejecutable. Esta es una de las mayores diferencias de CoMoMAS con CommonKADS, ya que se pone mucho énfasis en este último paso.

CoMoMAS es pues una metodología que propone una posible adaptación al estándar europeo de metodología para la Ingeniería del Conocimiento

(CommonKADS), donde se describen todos los pasos desde las primeras fases de análisis a la generación de código ejecutable.

DESIRE

DESIRE [Bra97] *(framework for Design and Specification of Interacting Reasoning components)*, modela explícitamente el conocimiento, la interacción y la coordinación de las tareas complejas y las capacidades de razonamientos de los sistemas de agentes. Esta metodología persigue conseguir que los sistemas desarrollados sean robustos, fiables y que se adecuen al propósito para el que fueron creados.

DESIRE permite al diseñador del sistema especificar de forma precisa y explícita tanto las funcionalidades internas (las funcionalidades para realizar las tareas para alcanzar sus objetivos) y las funcionalidades externas (aquellas que se producen entre los distintos agentes del sistema, como la coordinación, la cooperación u otras pautas de interacción social). Aunque DESIRE fue creada originalmente para especificar sistemas software complejos, ya que su filosofía consiste en ver los sistemas como una serie de componentes que interaccionan, es ideal para la especificación de sistemas multiagente. Uno de los objetivos principales de esta metodología es crear un marco de trabajo que ofrezca las herramientas necesarias para permitir que se puedan modelar explícitamente los patrones de razonamiento.

El entorno de modelado de alto nivel de DESIRE puede generar automáticamente aplicaciones directamente a partir de las especificaciones. Uno de los puntos fuertes de DESIRE es que ya ha sido probado con éxito en distintos proyectos para desarrollar sistemas operacionales para distintas tareas complejas (sistemas de diagnosis, diseño y planificación). Además las especificaciones, y la semántica de éstas, realizadas en DESIRE pueden convertirse a un marco formal utilizando para ello una lógica temporal. Esto permite que se puedan comprobar distintas propiedades sobre el sistema durante las fases de verificación y validación.

Entre las aplicaciones prácticas donde se ha utilizado DESIRE destaca el proyecto para la gestión de la red de transporte de electricidad, desarrollado dentro del proyecto ARCHON y que está actualmente funcionando en el norte de España [Jen95].

Los modelos de tareas definen la estructura de las arquitecturas composicionales. Los componentes en una arquitectura composicional están directamente relacionados con la (des)composición de tareas y subtareas Las estructuras jerárquicas de las tareas, la interacción y el conocimiento son conservadas dentro de la arquitectura composicional.

En DESIRE se propone un marco de trabajo formal para el modelado de las tareas de los sistemas multiagente siguiendo las siguientes fases:

1. (des)Composición de tareas.
2. Intercambio de información.
3. Secuencia de (sub)tareas.
4. Delegación de subtareas.
5. Estructuras de conocimiento.

La (des)composición de tareas consiste en modelar y especificar las tareas basándose en una jerarquía de tareas, qué información necesita la tarea como entrada, qué información de salida (resultado) debe generar la tarea, y finalmente qué (sub)tareas razonan sobre otras (sub)tareas.

La información que necesita/produce la (sub)tarea se denomina firma de entrada/salida de un componente. Las unidades de información se representan usando los átomos definidos en la firma (que está definida de acuerdo a una lógica de predicados).

El intercambio de información entre las tareas se especifica utilizando enlaces (*link*) entre los componentes. Cada enlace de información relaciona la salida de un componente con la entrada de otro, especificando qué valor de un átomo específico de salida esta enlazado con qué átomo de entrada.

Las secuencias de tareas se modelan explícitamente dentro de los componentes como conocimiento de control de la tarea. El conocimiento de control de la tarea incluye el conocimiento sobre qué subtareas deben activarse, cuándo y cómo, además de las metas asociadas a la activación de las tareas y la cantidad de esfuerzo que puede ser empleado en intentar alcanzar una meta.

Durante el desarrollo del proceso de modelado se realizan decisiones sobre qué (sub)tareas son realizadas mejor por qué determinados agentes.

Este proceso puede producirse también en tiempo de ejecución, lo que puede producir que cierta (sub)tarea deba delegarse a otro agente involucrado en la ejecución de la tarea.

En el proceso de adquisición de conocimiento es necesario definir la estructura apropiada para el dominio del conocimiento. Es necesario definir el significado de los conceptos utilizados para describir el dominio y las relaciones entre los conceptos y los grupos de conceptos. Los conceptos y las relaciones entre estos se definen en forma de jerarquías y reglas.

En DESIRE se propone un modelo genérico de agente que posee ciertas tareas comunes a todos los agentes: control de sus propios procesos, actualizar la información del estado del mundo donde está incrustado, y gestionar las comunicaciones con otros agentes. Además, cada agente tienen asignada una o más tareas que debe realizar. Tanto la especificación como la instanciación de cada una de las tareas genéricas del agente difiere mucho de un agente a otro, ya que cada agente actúa de acuerdo a un tipo distinto de información e interacción, y no tienen por qué tener una vista común del mundo exterior.

Los enlaces de información modelan el intercambio de información entre los componentes dentro del agente. La información que se comunica al agente se envía directamente al componente que gestiona la comunicación del agente.

El control de los agentes así como la interacción entre los agentes se especifica exactamente de la misma manera que el control y la interacción entre los componentes dentro del agente. Dentro del agente los componentes pueden ser autónomos o controlados. El conocimiento de control de la tarea del agente especifica cuándo y cómo se activan los distintos componentes y sus enlaces.

Aparte de intercambiar información con otros agentes, los agentes también pueden intercambiar información con el mundo exterior. La interacción entre el agente y el mundo exterior se modela exactamente igual que la interacción entre agentes. Cuando un agente desea observar qué ocurre en el mundo exterior se pueden modelar como una petición de información del agente sobre el mundo exterior, y desde el mundo exterior. Los patrones de comunicación entre los agentes se pueden

especificar usando enlaces de información y el conocimiento del agente sobre cuando hay que enviar la información.

La semántica formal del lenguaje de especificación está basada en una lógica temporal, lo que permite que se validen y verifiquen ciertas propiedades de gran interés en sistemas donde la seguridad es un factor crítico.

Especificación de Agentes en Z

Michael Luck y Mark d'Inverno [Luc95] presentan una representación formal tanto del concepto de agente, como del concepto de agente autónomo basándose en el lenguaje de especificación Z, ampliamente extendido dentro de la Inteligencia Artificial.

Las razones que aportan para la utilización del lenguaje Z son por un lado que la modularidad, la abstracción y la expresividad que proporciona son suficientes para permitir el modelado de la estructura de un sistema y sus operaciones. Y por otro lado, que los esquemas del lenguaje Z son especialmente adecuados para cuadrar el modelo formal con la implementación, lo que permitirá una transición desde las especificaciones al programa.

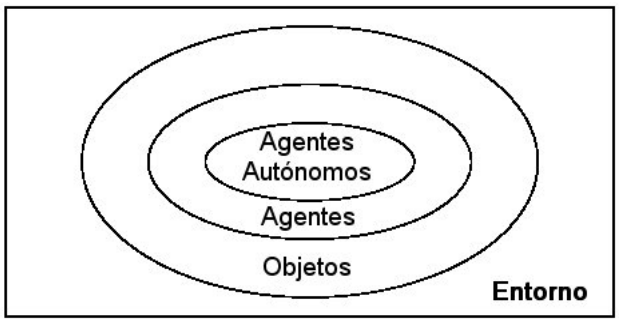


Figura 2.35 Jerarquía de las Entidades.

Es necesario crear un modelo que describa, inicialmente, sólo el entorno, para luego incrementalmente ir aumentando el nivel de detalle para definir objetos, agentes y agentes autónomos. Se propone una jerarquía de tres capas: objetos, agentes y agentes autónomos. La idea básica es que todas las entidades son objetos, algunos de estos objetos serán agentes, y finalmente, algunos de estos agentes serán agentes autónomos. Un objeto es un refinamiento del entorno, un agente es un refinamiento de un

agente, y lógicamente, un agente autónomo es un refinamiento de un agente. La figura 2.35 muestra la jerarquía descrita.

El entorno es la descripción del mundo más abstracta, y realiza una descripción del mundo en función de sus atributos. Los atributos son simplemente características del mundo que no necesitan ser percibidas por ninguna entidad en particular. Un atributo es por lo tanto una característica perceptible.
Un objeto agrupa una serie de atributos del entorno. Además, un objeto también contiene una descripción de sus capacidades. Las capacidades del objeto se definen en función de un conjunto de primitivas de acciones. Una acción es un evento discreto que cambio el estado del entorno. Para definir cómo será el comportamiento del objeto se define la función *objectactions*, que determina el conjunto de acciones que va a realizar el objeto dentro de un entorno dado. A continuación es necesario describir el estado del objeto dentro de un entorno particular. Para ello se define la variable *History* que representa la secuencia de conjuntos de acciones que el objeto ya ha realizado, y la variable *WillDo* que presenta el conjunto de acciones que el objeto va a realizar a continuación. Cuando se realizan acciones se producen interacciones. Cuando un objeto interacciona con su entorno y realiza una acción, tanto el estado del objeto como el estado del entorno cambian. El concepto de objeto proporciona el bloque de construcción básico para definir la noción de agente.

Un agente es un objeto, que además tienen asociadas una serie de metas. Según Luck y d'Inverno los agentes son sólo objetos con ciertas intenciones. Un agente puede ser un agente a lo largo de todo su ciclo de existencia, o puede volver a ser un objeto bajo ciertas circunstancias. Una meta es una serie de objetivos que deben ser alcanzados en el entorno. Una agente será pues una instanciación de un objeto junto con sus metas asociadas. Un mismo objeto podrá, por la tanto, dar lugar a distintas instanciaciones de agentes. La instancia del agente se crea como respuesta a la petición de otro agente. Ello conlleva que el estado de agente es de carácter transitorio, y que en algún momento el agente puede volver a ser un objeto.

Para modelar el hecho de que cada agente puede tener una vista diferente del mundo, Luck & d'Inverno introducen el concepto de vista (*view*). La vista representa el conjunto de atributos que representan el mundo tal y como el agente lo ve. Para especificar qué es lo que el agente realmente ve

se define la variable *PerceivingActions*, que es un subconjunto de las capacidades del agente. *CanPerceive* determina los atributos que el agente puede percibir. Los atributos que percibe el agente realmente son definidos por la función *WillPerceive*.

Para alcanzar una meta el agente debe decidir que acciones son necesarias, de entre las que es capaz de desarrollar. La decisión estará basada tanto en las metas que tiene asociadas el agente como en la percepción actual que tiene de su entorno. Para ello se define la función *AgentActions*.

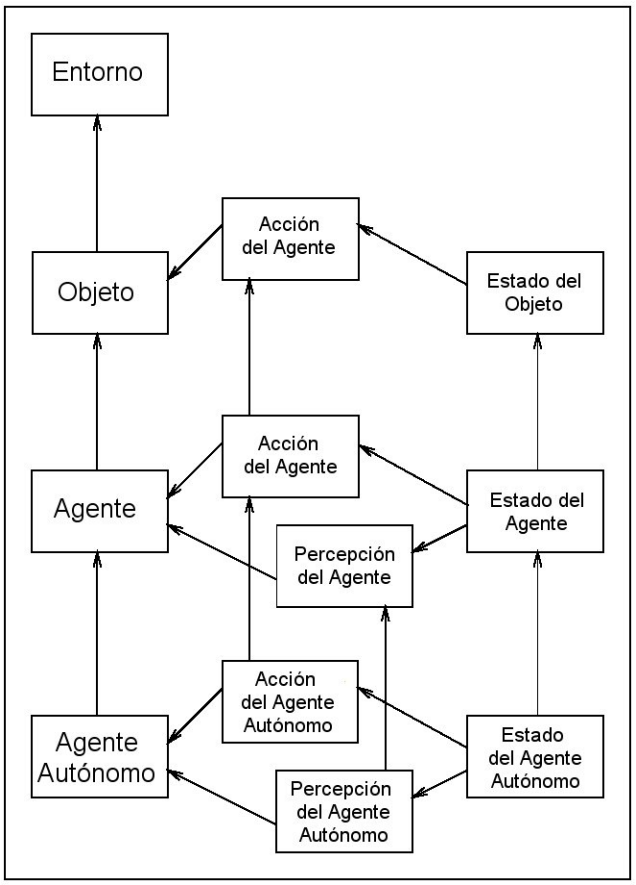


Figura 2.36 Esquema de la visión conceptual de Z.

Los agentes autónomos van un paso más allá. Se trata de aquellos agentes que son autónomos, es decir, de aquellos agentes que siguen su propia agenda de tareas en vez de funcionar bajo en control de otro agente. Una agente autónomo se representa como una instanciación de un agente junto

con un conjunto de motivaciones asociadas. Una motivación es una preferencia que puede conducir a adoptar nuevas metas, y que influye en el comportamiento del agente en la ejecución de la tarea asociada a esas metas. Tanto las motivaciones como las metas determinan cómo percibe su entorno el agente autónomo. Por ello se añade la función *AutoWillPerceive* que añade este hecho a la función *WillPerceive* que aparecía en los agentes.
Luck y d'Inverno presentan una especificación del concepto de agente que permite modelar sistemas multiagente de una manera formal. La especificación presentada no pretende ser completa, sino que es un punto de partida para llegar a una definición más detallada a partir de sucesivos refinamientos.

2.4 Conclusiones del capítulo

El diseño de interfaces de usuario adaptativas ha sido afrontado, en su mayor parte, aplicando soluciones adhoc, que no se integran dentro del ciclo de vida del desarrollo habitual de una aplicación. La utilización de este tipo de métodos adhoc hace que el coste de diseño de una interfaz de usuario con ciertas capacidades de adaptación sea muy elevado, debido al bajo nivel de reutilización, tanto del código como de la experiencia adquirida durante el desarrollo de las capacidades de adaptación.

Las aproximaciones basadas en modelos han proporcionado al desarrollo de interfaces de usuario la sistematización necesaria para la reutilización tanto del código como en parte de la experiencia, y han permitido la generación automática o semiautomática de las interfaces de usuario para distintas plataformas.

Dentro de estas aproximaciones se han incluido ciertas capacidades de adaptación, aunque principalmente estática. Es decir, durante la etapa de desarrollo se han generado distintas interfaces de usuario adaptadas a distintos perfiles o roles, como por ejemplo en *IDEAS* [Loz00]. Métodos como *UIDE* [Suk93] o *FUSE-System* [Lon95] incluyen adaptación dinámica, pero sin embargo lo limitan a la generación de la ayuda de forma dinámica de acuerdo a la tarea actual del usuario.

Dentro del ámbito hipermedial, *UWE* [Koc01] sí que permite el modelado de las técnicas de adaptación [Bru03] más habituales dentro de este tipo de sistemas, pero no está diseñado para la creación de nuevos tipos de adaptaciones.

Desgraciadamente no existe ningún método que haya integrado el proceso de diseño de la adaptación dentro del proceso de desarrollo de la interfaz de usuario, permitiendo el modelado de las capacidades de adaptación desde un punto de vista ingenieril.

Es en este punto donde esta tesis aporta una solución para el diseño de todos los constructores involucrados en el diseño de una interfaz de usuario adaptativa, desde la fase de captura de requisitos a la implementación, permitiendo la reutilización tanto del código como de la experiencia acumulada durante cada desarrollo de una interfaz adaptativa.

Por otra parte la necesidad de que la adaptación de la interfaz de usuario sea precisa, y redunde en una mejora de la usabilidad del sistema que sea palpable por le usuario, es necesario proporcionar una arquitectura que interprete la especificación creada durante el diseño, y que presente un cierto grado de "inteligencia" en la toma de decisiones durante el proceso de adaptación, especialmente en las fases de detección de la posible necesidad de adaptación y en la elección de cuál es la mejor adaptación a aplicar. En este sentido, dentro de este trabajo se propone una arquitectura basada en el concepto de agente como medio para introducir ese grado de inteligencia necesario citado, a través de la colaboración de una serie de agentes software en busca de la consecución de un resultado óptimo en la aplicación del proceso de adaptación, y que es en sí mismo adaptativo, mejorando las heurísticas aplicadas a lo largo del tiempo de acuerdo a la experiencia adquirida durante la interacción con los usuarios de la aplicación adaptativa.

A lo largo de este capítulo se han descrito distintos métodos para el modelado de sistemas multi-agente. Para la especificación de la arquitectura multi-agente para la adaptación de interfaces de usuario se ha escogido la metodología *Prometheus*, ya que cubre todo el ciclo de vida, está ampliamente extendida en la comunidad de investigación de sistemas multi-agente, proporciona una herramientas visual para el diseño del sistema multi-agente, y además dicha herramienta es capaz de generar código e integrarse con los dos lenguajes de programación de agentes más extendidos: JACK [Bus99] y JADE[16].

[16] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

CAPÍTULO 3
DISEÑO E IMPLEMENTACIÓN DE INTERFACES DE USUARIO ADAPTATIVAS

"En principio la investigación necesita más cabezas que medios."
(Severo Ochoa)

3.1 Introducción

El diseño de interfaces de usuario es de por sí complejo, lo cual ha suscitado un gran interés por la exploración de métodos de diseño de interfaces de usuario capaces de facilitar la creación de interfaces de usuario, y donde se intentan extrapolar las técnicas habituales en el diseño de la parte funcional del sistema para conseguir los principales beneficios de un proceso de desarrollo bien definido: generación automática/semiautomática del código, reutilización del código, buena mantenibilidad o la reutilización de la experiencia.

Para ello, los métodos de diseño de interfaces de usuario basados en modelos aplican las técnicas de las arquitecturas dirigidas por modelos *(Model-driven Architecture – MDA*) en busca de proporcionar todas las ventajas anteriormente descritas.

Sin embargo, la aplicación de las aproximaciones basadas en modelos para el diseño de interfaces de usuario debe afrontar un gran reto que habitualmente no aparece en el diseño de la parte funcional de una aplicación: los requisitos no funcionales, y especialmente la usabilidad. Este es actualmente el talón de Aquiles de los métodos de diseño de interfaces de usuario basados en modelos, ya que actualmente la usabilidad de las interfaces de usuario generadas no es todavía suficientemente madura para ciertos tipos de aplicaciones.
Por otra parte, como ya se ha descrito durante el capítulo anterior, el diseño de las capacidades de adaptación que hoy en día las interfaces de

usuario necesitan para afrontar las exigencias de usabilidad que los usuarios demandan, no ha sido incluido dentro de las aproximaciones basadas en modelos de diseño de interfaces de usuario, haciendo necesario parchear de alguna manera la interfaz de usuario creada para añadir las capacidades de adaptación deseadas. Ello supone la ruptura de gran parte de los beneficios que supone la adopción de métodos basados en modelos y especialmente de la mantenibilidad y la reutilización.

Nuestra propuesta pasa por la inclusión del diseño de las capacidades de adaptación de la interfaz de usuario como una entidad de primer orden en el diseño de la interfaz, permitiendo el modelado de todos sus constructores, e integrándolo con la especificación del resto de la interfaz de usuario, para de esta forma obtener todos beneficios derivados de la aplicación de un método de diseño basado en modelos.

A continuación, una descripción de la definición de adaptación propuesta es descrita.

3.2 La Adaptación como proceso de iniciativa mixta

De acuerdo a Horvitz [Hor99][Hor99b], iniciativa mixta (*mixed-iniative*) son "*los métodos que explícitamente permiten unas contribuciones eficientes e intercaladas del usuario y de servicios automatizados cuyo objetivo es converger en la solución de problemas*". De la definición de iniciativa mixta, se deduce que cualquiera de las etapas dentro del proceso de adaptación podrá ser realizada en colaboración entre varios agentes (humanos, máquinas o una combinación de humanos y máquinas). A partir de esta definición se desprenden algunos problemas relacionados con la colaboración entre el usuario y el sistema:

- ¿Cuándo debe el sistema iniciar la colaboración con el usuario?
- ¿Cuál es la mejor manera de contribuir en la resolución de un problema?
- ¿Cuándo debe el sistema devolver el control al usuario para que refine la solución?
- ¿Cuándo debe el sistema pedir al usuario información adicional para resolver un problema?

El momento en el que el sistema interacción con el usuario para realizar una tarea colaborativa es un factor importante. Si el sistema comienza una colaboración, aplica una adaptación o pide información en un momento inoportuno el usuario tendrá una percepción mala del sistema, y a menudo redundará en la desactivación de las capacidades de adaptación de la interfaz de usuario. Por lo tanto, es necesario aportar algún tipo de modelo que describa la atención del usuario para permitir al sistema actuar en el momento apropiado, y de esta forma incrementar la eficiencia del sistema adaptativo. Otro factor que introduce una gran influencia en la percepción de la calidad del sistema por parte del usuario es la latencia del proceso de adaptación. En sistemas altamente interactivos, una alta latencia hará que el sistema sea realmente inutilizable debido a los tiempos de espera que el usuario tendrá que soportar entre sus operaciones y la aplicación de las adaptaciones.

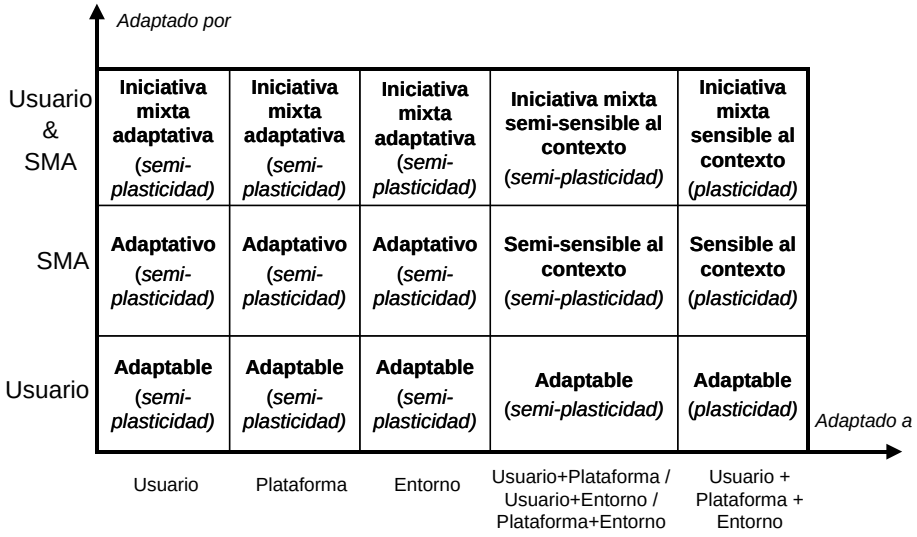


Figura 3.1 Clasificación de la IU en función de sus capacidades de adaptación y de los actores involucrados en el proceso de adaptación.

En la figura 3.1 se describe la clasificación de las interfaces de usuario propuesta de acuerdo a sus capacidades de adaptación y a quienes son los actores que realizan la adaptación que extiende la propuesta en [Cal01a].

El proceso de adaptación de una interfaz de usuario puede ser realizado en nuestro caso por tres combinaciones distintas de agentes. Por una parte podemos tener casos en los que la adaptación sea llevada a cabo por el usuario, en cuyo caso tendremos interfaces de usuario adaptables. Si por el

contrario es el sistema multiagente el que realiza la adaptación estaremos entonces ante un sistema adaptativo. Finalmente, si el proceso de adaptación es realizado en colaboración por el usuario y el sistema multi-agente obtendremos una interfaz de usuario adaptativa con iniciativa mixta.

Por otro lado, la interfaz de usuario puede ser plástica o semiplástica. Una interfaz plástica es aquella capaz de adaptarse a cambios tanto al usuario que está usando la aplicación, a la plataforma donde se está ejecutando la aplicación y al entorno donde se produce la interacción manteniendo la usabilidad del sistema. Si por le contrario nos encontramos con que el sistema sólo es capaz de adaptarse a alguna de estas tres componentes, entonces se tiene una interfaz de usuario semiplástica.

Finalmente, en el caso de que el sistema sea capaz de adaptarse a cambios en el usuario, la plataforma y el entorno, pero no garantice la usabilidad del sistema, tendemos un sistema sensible al contexto. Si el sistema es capaz tan solo de adaptarse a algunos de las componentes del contexto, es denominado semi-sensible al contexto.

3.2.1 Un modelo de adaptividad en interfaces de usuario con iniciativa mixta

El usuario o el sistema pueden iniciar el proceso de adaptación. En nuestra propuesta, el sistema es presentado como un sistema multi-agente (SMA) que realiza la adquisición de datos necesaria para que el sistema sea capaz de reacción y realizar las adaptaciones, es capaz de inferir la necesidad de una adaptación, la decisión de seleccionar qué adaptación se aplicará de entre las posibles, y finalmente aplicar la adaptación.

Cuando el usuario o el sistema multi-agente inician el proceso de adaptación, el primer paso será averiguar el estado actual en que se encuentra la interacción con el usuario, es decir, capturar el estado actual del contexto de uso (véase la sección 3.4.1.2). Para aplicar una adaptación que sea apropiada, y produzca un beneficio, y no una degradación de la usabilidad del sistema, es aconsejable intentar saber cuál es la tarea actual que el usuario está realizando con el sistema, de forma que el sistema puede determinar cuáles son las posibles necesidades que tiene el usuario dado el contexto de uso actual y la tarea que se está realizando. Por ejemplo, en una interfaz de usuario adaptativa diseñada para mejorar la experiencia en la conducción de un coche, si un coche entra dentro de un

túnel en una autopista con unas malas condiciones de iluminación, el sistema debería detectar la tarea "conduciendo en una autopista". Si el sistema no es capaz por sí mismo de detectar la tarea actual o el contexto, puede recurrir a agente externos, como podría ser en este caso un servicio de GPS, para detectar a través de qué carretera y en qué punto exacto de dicha carretera se encuentra el coche actualmente. En este momento los sensores del coche comunicarían las malas condiciones de iluminación, y el sistema inferiría la necesidad de mejorar la iluminación para mejorar la seguridad del usuario durante la conducción por el túnel. En el caso de que el túnel estuviera iluminado suficientemente para hacer la conducción segura, el sistema podría consultar a un agente externo, por ejemplo un sitio Web sobre la legislación de tráfico para averiguar si en el país donde actualmente se encuentra conduciendo el usuario es obligatorio o no encender los faros del coche cuando se conduce a través de un túnel.

Una vez que la tarea actual, y las necesidades para esa tarea en el contexto actual de uso han sido detectadas, el sistema debe elaborar una lista de adaptaciones que sean factibles para la tarea actual, las necesidades detectadas y el contexto de uso actual. De nuevo el sistema puede recurrir a agentes externos para ampliar y mejorar la calidad del repertorio de posibles adaptaciones aplicables. En nuestro ejemplo de la interfaz de usuario para un coche, el sistema seguramente sugeriría al usuario/conductor encender las luces con un alto grado de certeza. En el caso de que según las leyes sea obligatorio encender las luces dentro del túnel, el sistema podría incluso encenderlas por sí mismo. Si no es obligatorio encender las luces dentro del túnel, otra posible adaptación que el sistema podría sugerir al usuario sería que reduzca la velocidad del automóvil, para que sea adecuada a las condiciones de visibilidad dentro del túnel. El proceso de adaptación descrito se haya especificado en la figura 3.2.

Es en este momento cuando el sistema ha confeccionado una lista de las adaptaciones más adecuadas de acuerdo a los datos de entrada recogidos.

Por lo tanto, es necesario decidir qué adaptación o adaptaciones serán realmente aplicadas (si es que alguna es aplicada). Para decidir cuál o cuáles son las mejores adaptaciones a aplicar, el sistema debe calcular el beneficio o perjuicio que una adaptación va a producir para el usuario.

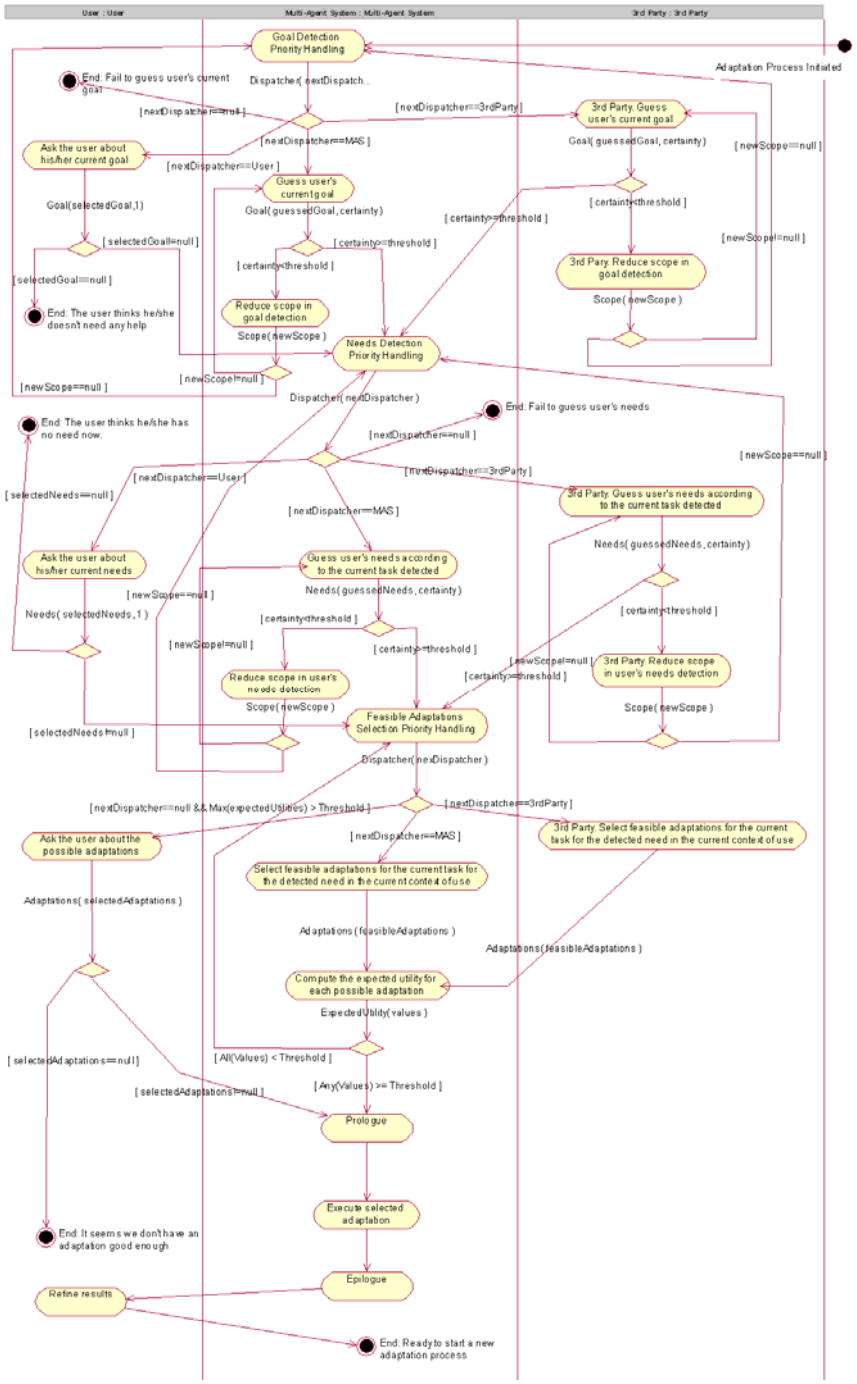


Figura 3.2. Modelo del proceso de adaptación con iniciativa mixta propuesto.

Horvitz [Hor99] propone utilizar la teoría de la utilidad esperada propuesta por Von Newman y Morgenstern [Von44] para este propósito. En nuestro caso el cálculo se realiza evaluando tanto el beneficio como el perjuicio que cada adaptación tendría en caso de su aplicación (véase la sección 5.2.6.1).

Una vez calculada la bondad de cada posible adaptación aplicable se debe tomar la decisión sobre qué adaptación o adaptaciones aplicar. La estrategia más lógica es escoger para su aplicación aquella adaptación con un mayor valor de bondad. Sin embargo, se debe tener en cuenta que si el máximo de los valores de bondad de las adaptaciones es demasiado bajo se puede optar por no aplicar ninguna adaptación. Es mejor no realizar ninguna adaptación que realizar una que no satisfaga al usuario. Por otra parte, si se obtienen valores cercanos a la media sería aconsejable preguntar al usuario qué adaptación prefiere, si es que prefiere alguna. Si se obtienen valores de bondad suficientemente altos significa que el sistema tiene un alto grado de confianza en la aplicación de dichas adaptaciones, por lo que se podrían aplicar sin consultar al usuario.

Una descripción detallada de cómo se realiza de forma más concreta cada una de las fases del proceso de adaptación puede encontrarse en el capítulo 5.

3.3 Adaptación vs. Usabilidad

La agregación a una interfaz de usuario de capacidades de adaptación no garantiza por si misma la mejora de la usabilidad de sistema, sino que incluso puede conducir a un deterioro de la misma. Es por lo tanto, por una parte necesario realizar el proceso de adaptación de una forma adecuada de tal manera que se minimice el impacto sobre el usuario, y por otra parte será necesario garantiza que cualquier modificación que se realice sobre la interfaz de usuario mantiene la usabilidad del sistema. Por ejemplo, si una adaptación necesita reducir los contenidos que se muestran en pantalla porque se ha reducido el espacio de presentación, debería ser muy cuidadoso a la hora de elegir qué datos ocultar, de forma que nunca se oculten datos necesarios para la finalización de la tarea en curso.

Cuando se realiza una adaptación es necesario tener en cuenta que la adaptación debe realizarse en el momento apropiado, ya que si se realizara en un momento erróneo la percepción del usuario del sistema se

degradaría de forma importante. Por ejemplo, si se realiza una adaptación mientras el usuario está realizando una tarea de introducción de datos, es fácil que el usuario se frustre al ser interrumpido el proceso de introducción de datos (por supuesto, existen excepciones a este caso, por ejemplo, si se desea facilitar la introducción de datos con funciones de autocompletar).

Aplicando estas ideas al ejemplo de la interfaz de usuario del coche, en el caso en que la propuesta de adaptación con un mayor valor de bondad sugiera al conductor la reducción de la velocidad del vehículo, el sistema mostraría un mensaje al usuario proponiéndole la reducción de la velocidad. Sin embargo, si existe información adicional del contexto, que por ejemplo, permitan saber si el conductor se encuentra conduciendo en este momento por una recta o una curva, el sistema debería escoger el momento apropiado para mostrar el mensaje. En este caso debería esperar a que termine la curva, ya que la adaptación no está relacionada directamente con la curva, de forma que el sistema minimice la distracción de la atención del usuario durante la conducción.

Si por el contrario la adaptación que se ha estimado como más apropiada fuera encender los faros, el sistema puede colaborar con el usuario para refinar la solución propuesta. El sistema podría encender las luces cortas, y sugerir al usuario que refine la solución encendiendo las luces largas si él piensa que es necesario.

3.4 AB-UIDE: Una aproximación para el diseño de interfaces de usuario adaptativas

AB-UIDE [Lop04] (*Agent-Based User Interface Development Environment*) pretende dar una solución a la integración del diseño de interfaces de usuario adaptativas dentro de las actuales aproximaciones basadas en modelos, añadiendo los aspectos del proceso de adaptación que no se han contemplado dentro de los métodos basados en modelos habituales.

Para ello, AB-UIDE aporta los resortes necesarios para la descripción de los constructores que conforman la definición de un proceso de adaptación.

3.4.1 Descripción de la adaptatividad en AB-UIDE

La descripción de los constructores del proceso de adaptación determina la manera en que se modelan los distintos elementos que influyen en el proceso de adaptación.

3.4.1.1 Constituyentes

Los constituyentes son los elementos de la interfaz de usuario que pueden ser adaptados (contenidos, modalidad, navegación, técnica de interacción, ...). Tradicionalmente dentro del mundo hipermedial las adaptaciones se han clasificado en tres tipos [Bru03]:

1. La **adaptación de los contenidos** persigue la modificación de los contenidos que se muestran al usuario de acuerdo a sus preferencias, objetivos o conocimientos sobre el tema del cual se están mostrando los contenidos. Este tipo de adaptaciones permiten proporcionar material adicional a ciertos usuarios, eliminar o atenuar fragmentos de código de menor relevancia u ordenar los fragmentos de los contenidos poniendo los más relevantes primero. Para ello se utilizan técnicas como el filtrado condicional de texto o el *stretchtext*. El *stretchtext* es texto elástico que el usuario o el sistema pueden desplegar o comprimir.

2. La **adaptación de la presentación** donde el contenido de la información permanece inalterado (idealmente), y sólo se modifica el formato y disposición de los objetos. La adaptación de la presentación es habitualmente implementada alrededor de las hojas de estilo en cascada (CSS).

3. La **adaptación de la navegación** en el mundo hipermedial se plasma principalmente en la modificación de los enlaces mediante distintas técnicas, como pueden ser: la ordenación adaptativa de enlaces, la ocultación adaptativa de enlaces o la generación adaptativa de enlaces.

Sin embargo, el propósito de AB-UIDE no es permitir la inclusión de un número concreto de técnicas de adaptación como ocurre en UWE [Koc01], sino proporcionar los mecanismos necesarios para que el

diseñador sea capaz de definir sus propios mecanismos de adaptación y reutilizarlo si así lo cree conveniente en distintos proyectos.

Es por ello, que en AB-UIDE los constituyentes son cualquier elemento de la interfaz de usuario, es decir, cualquier información recogida en los modelos que describen la interfaz de usuario (dominio, tareas, usuario, plataforma, presentación, ...). Para ello, se ha escogido un modelo ontológico de descripción de interfaces de usuario [Sou03] denominado usiXML[17][Lim04][Lim04b] que ha sido extendido con los resortes necesarios para su integración en el proceso de especificación de la adaptación definido en AB-UIDE.

UsiXML: Una ontología para descripción de interfaces de usuario

usiXML es un lenguaje para la descripción de interfaces de usuario que permite la especificación de las características más habituales usadas en el desarrollo de interfaces de usuario basadas en modelos, y almacenarlas en un fichero en formato XML. El lenguaje se basa en el marco de desarrollo de interfaces de usuario desarrollado dentro del proyecto europeo Cameleon [Cal03], cuyos niveles de abstracción son ilustrados en la figura 3.3.

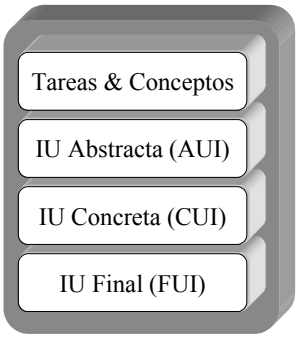


Figura 3.3 Marco de desarrollo de interfaces de usuario definido en Cameleon [Cal03].

- Las **tareas y los conceptos** representan las tareas que el usuario podrá realizar a través de la interfaz de usuario y los objetos del dominio que dichas tareas deben manipular para ser llevadas a cabo por el usuario.

- La **interfaz de usuario abstracta (AUI – *Abstract User Interface*)** describe la interfaz de usuario con la cual el usuario va

[17] usiXML: http://www.usixml.org

a interaccionar, pero usando un alto nivel de abstracción. Este alto nivel de abstracción hace que la especificación de la interfaz de usuario abstracta sea independiente tanto de la plataforma final donde será ejecutada como de la modalidad usada para interaccionar con la interfaz de usuario.

- La **interfaz de usuario concreta (CUI – *Concrete User Interface*)** representa la interfaz de usuario de una forma abstracta, pero en un nivel de abstracción menor al usado en la interfaz de usuario abstracta. En este caso la especificación de la interfaz de usuario concreta es independiente, de la plataforma donde se ejecutará la aplicación, pero es dependiente de la modalidad que será usada (gráfica, vocal o textual).

- La **interfaz de usuario final (FUI – *Final User Interface*)** representa el código que será ejecutado en la plataforma destino para mostrar la interfaz de usuario. Por lo tanto, esta versión de la interfaz de usuario es dependiente tanto de la plataforma donde se ejecutará la interfaz como de la modalidad que será usada para interactuar con ella.

Una interfaz de usuario en usiXML (véase la figura 3.4) es descrita utilizando una serie de modelos que representan los modelos más habitualmente usados en los desarrollos de interfaces de usuario basados en modelos, y adicionalmente algunos modelos que permiten representar posibles transformaciones que se puedan aplicar sobre un modelo, siguiendo el paradigma de MDA. A continuación se describen los modelos incluidos en usiXML.

Modelo de interfaz de usuario

El modelo de interfaz de usuario representa toda la información disponible de la interfaz de usuario, y actúa por lo tanto como contenedor del resto de modelos.

Modelo de tareas

El modelo de tareas en usiXML permite representar un modelo de tareas en forma de árbol, siguiendo una notación muy similar a la propuesta en CTT [Pat99], con una representación en XML.

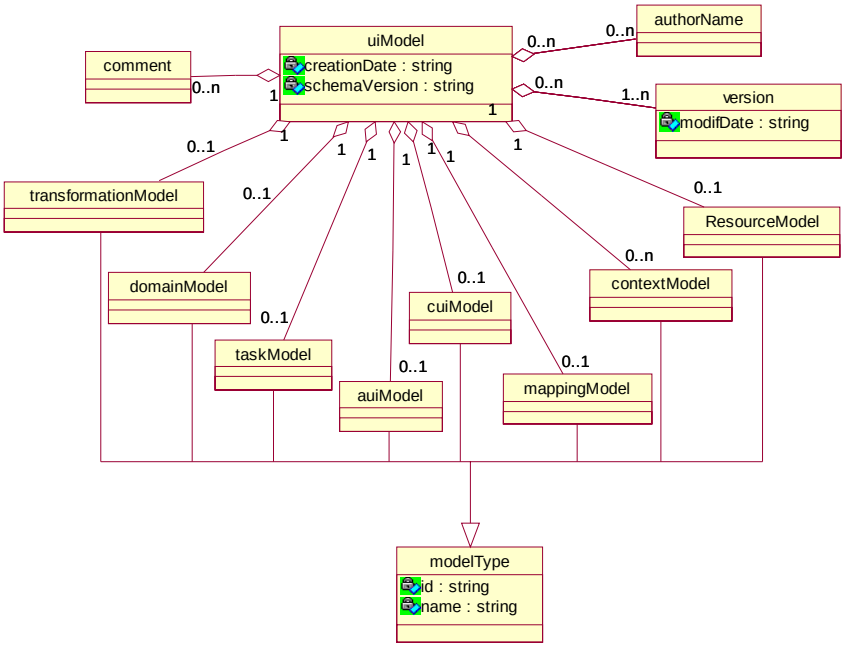


Figura 3.4 Modelo de interfaz de usuario de usiXML.

Modelo de dominio

El modelo de dominio representa los objetos del dominio con los que interactúan las tareas que el usuario podría realizar con el sistema. usiXML permite expresar un diagrama de clases y objetos de UML con una representación XML, incluidas las relaciones entre las distintas clases y objetos. Adicionalmente, también permite especificar ciertas propiedades de los atributos de las clases y objetos que permiten generar la interfaz de usuario de una forma más precisa.

Modelo de interfaz abstracta (AUI)

El modelo de interfaz abstracta permite representar la interfaz de usuario utilizando un alto nivel de abstracción. Básicamente, la interfaz se describe usando contenedores abstractos y elementos individuales abstractos. A los elementos individuales abstractos se les puede asociar facetas que describen las capacidades de interacción que presentan (entrada, salida, navegación o control). De igual forma, también es posible la definición de relaciones abstractas entre los distintos elementos abstractos

(*abstractContainment*, *abstractAdjacency*, ...). Esta descripción abstracta es independiente tanto de la plataforma destino como de la modalidad.

Modelo de interfaz concreta (CUI)

Describe la interfaz de usuario de una forma menos abstracta que el modelo de interfaz de usuario abstracta, y de una manera dependiente de la modalidad elegida, aunque independiente de la plataforma destino. El modelo de interfaz de usuario concreta permite describir la interfaz de usuario en función de contenedores y elementos individuales, y relaciones entre estos. Los contenedores pueden ser cajas, tablas, cuadros de diálogo, etc. Los elementos individuales pueden ser elementos de texto, imágenes, botones de distintos tipos, etc. Actualmente usiXML contiene una gran expresividad en el nivel concreto para la representación de interfaces gráficas, y está en curso el desarrollo de los elementos para la representación de interfaces de usuario auditivas.

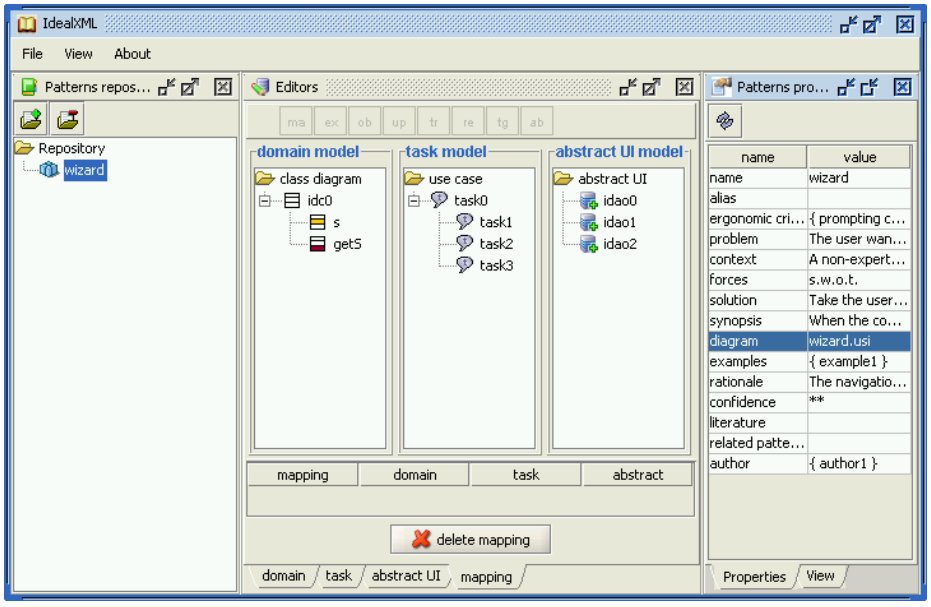


Figura 3.5 IDEALXML: un editor de modelos de usiXML [Mon05].

*Modelo de relaciones intermodelo (***mapping***)*

El modelo de relaciones intermodelo permite almacenar las relaciones que aparecen entre unos modelos y otros. Por ejemplo, se puede representar en qué contenedor se visualizará una tarea concreta. Las relaciones

intermodelo pueden ser creadas usando idealXML[18] [Mon05] (véase la figura 3.5), herramienta desarrollada por Francisco Montero que permite además crear los modelos de tareas, dominio y abstracto de usiXML.

Modelo de transformación

El modelo de transformación en usiXML describe el proceso de transformación entre los distintos modelos de usiXML propuesto en [Lim04b]. La idea principal detrás de este modelo es permitir la especificación de la transformación de modelos de usiXML en otros, de forma que se pueda generar una interfaz de usuario a partir de una serie de modelos base. Una de las novedades del modelo de transformación propuesto es la posibilidad de comenzar el proceso de desarrollo por distintos puntos de entrada, siguiendo lo que se ha denominado un desarrollo multi-ruta de interfaces de usuario.

Modelo de contexto

El modelo de contexto de usiXML describe las componentes del contexto consideradas habitualmente en el desarrollo de interfaces de usuario: el usuario, la plataforma y el entorno. El lenguaje ofrece un detallado modelo para la plataforma basado en el estándar CC/PP[19] para la descripción de perfiles de plataformas.

Modelo de recursos

El modelo de recursos permite el almacenamiento de los recursos de forma separada a los elementos que los utilizan, permitiendo de forma sencilla la internacionalización de las aplicaciones.

Por lo tanto, los constituyentes en AB-UIDE son todos aquellos elementos que pueden ser expresados utilizando usiXML, lo cual dota al método de un gran potencial de expresividad de los posibles elementos que pueden ser adaptados en la interfaz de usuario.

3.4.1.2 Determinantes: el contexto de uso en AB-UIDE

Los determinantes son los factores que guían la adaptatividad (características del usuario, tareas, entorno). En nuestro caso los determinantes son aquellas características del contexto de uso que pueden producir la ejecución de un proceso de adaptación.

[18] http://www.info-ab.uclm.es/personal/fmontero/idealXML.htm
[19] http://www.w3.org/Mobile/CCPP/

Habitualmente cuando se trata de describir el contexto de uso donde se desarrollará la interacción para el diseño de una interfaz de usuario, se crea un modelo del usuario, de la plataforma, y del entorno. El modelo de contexto construido será uno de los pilares fundamentales para la definición de las posibles adaptaciones.

El modelo de usuario en AB-UIDE

El modelo de usuario describe a los usuarios potenciales de la aplicación. Éste puede ser tanto estático como dinámico. El modelo estático se crea durante el diseño de la aplicación de una forma más o menos detallada, dependiendo de las necesidades de la aplicación. Sin embargo, si el modelo de usuario es estático, este no cambiará durante el uso de la aplicación. Por supuesto, el modelo podría ser modificado en sucesivas revisiones de la aplicación, pero nunca durante el uso de ésta.

Los modelos de usuario dinámicos evolucionan en tiempo de ejecución para intentar adecuarse con la mayor exactitud posible a la percepción que se tiene del usuario, o a la propia evolución del usuario a lo largo del tiempo. Por ejemplo, un usuario que al inicio del uso de una aplicación puede considerarse un novato, podrá evolucionar hasta convertirse en un experto en el uso de la aplicación con el paso del tiempo. Si se desea hacer que el modelo de usuario evolucione a lo largo del tiempo, es necesario algún tipo de motor capaz de recoger datos del usuario, y de inferir cambios en el modelo.

Otro hecho a tener en cuenta durante el desarrollo del modelo de usuario es si se diseña para cada usuario específico o para cada grupo, o estereotipo de usuarios. Si el modelo de usuario se aplicara sobre grupos es necesario especificar la técnica que se utilizará para deducir a qué grupo pertenece cada usuario. Para ello se pueden utilizar algoritmos de clasificación, como por ejemplo, los algoritmos de agrupación de datos que miden la distancia entre dos instancias de un modelo.

El modelo de usuario en AB-UIDE está compuesto por dos tipos de atributos del usuario. Por una parte existen atributos independientes de la aplicación, como pueden ser las características motrices del usuario, y por otra parte existen atributos dependientes de cada aplicación concreta, como puede ser el conocimiento que el usuario tiene sobre una aplicación concreta. Los atributos independientes de la aplicación sólo se deben diseñar una sola vez, mientras que los atributos dependientes de la

aplicación tendrán que ser diseñados para cada aplicación o dominio de aplicaciones.

Para poder optimizar la reutilización de las características del usuario independientes de la aplicación es necesaria la creación de un repositorio capaz de almacenarlas, y que sea accesible por las distintas aplicaciones. Sin embargo, aunque varias iniciativas han intentado ahondar en la definición de un repositorio de tales características, no parece que ninguna iniciativa haya conseguido tener éxito, debido principalmente a la desconfianza de los usuarios a ceder los datos sobre su perfil a una empresa. Otro de los problemas principales para este tipo de iniciativas ha sido la desconfianza de unas empresas con otras, por el tremendo potencial económico que la posesión de los datos de miles o millones de usuario puede reportar. Este ha sido uno de los principales factores en el fracaso de la tecnología *Passport* de Microsoft.

El modelo de plataforma en AB-UIDE

El modelo de plataforma describe donde se va a ejecutar la aplicación. El modelo incluye tanto la plataforma hardware donde se ejecuta la aplicación, como el software que permite la ejecución de la aplicación (por ejemplo, qué versión del JRE de Java incluye el sistema, o el sistema operativo sobre el que se ejecuta la aplicación). Algunas de las características más habituales en un modelo de plataforma son el tamaño de la pantalla del dispositivo, el número de colores que pueden ser visualizados simultáneamente, la potencia de cálculo o el ancho de banda de comunicación a través de la red.

El modelo de entorno en AB-UIDE

El modelo de entorno describe las características del entorno que pueden influir en la utilización de la interfaz de usuario. Teóricamente, puede estar compuesto de una gran cantidad de información, por lo que será el diseñador el que tenga que decidir qué factores del entorno afecta el uso de su aplicación y cuáles no.

El modelo de entorno en AB-UIDE no es un simple conjunto de atributos, sino que también incluye las relaciones entre los atributos que componen el entorno y las tareas sobre las que influye, así como las relaciones entre los atributos que componen el entorno, es decir, como afectan unas características del modelo a otras. A continuación se describe

un ejemplo de dichas relaciones para el ejemplo de la interfaz de usuario para la mejora de la experiencia de conducción de un coche:

- Condiciones de iluminación: { *alta*, *normal*, *baja*, *muy baja* }
  - **Afecta al Modelo de Usuario en**
    - Reduce Usuario.AgudezaVisual.
  - **Afecta la realización de las tareas**
    - Conducción.

La tarea actual como componente del contexto de uso en AB-UIDE

En las aproximaciones habituales para el diseño de interfaces de usuario basadas en modelos se considera el contexto de uso como la composición del usuario que utiliza la aplicación, el entorno donde se realiza la interacción, y la plataforma sobre la que se ejecuta la aplicación.

Sin embargo, dentro de AB-UIDE se ha introducido también la tarea actual en ejecución. La tarea actual en ejecución suministra información necesaria para ofrecer al usuario adaptaciones precisas y útiles. Por ejemplo, si desea ayudar al usuario con un sistema de ayuda sensible al contexto, la tarea actual que el usuario está intentando realizar es un factor crucial a la hora de aportar una ayuda que realmente sea útil.

Figura 3.6 Contexto de uso en AB-UIDE.

Pero para que sea útil se hace necesaria la introducción en el sistema de algún tipo de mecanismo que sea capaz de inferir cuál es la tarea actual en ejecución. El mecanismo diseñado para la ejecución de especificaciones creadas con AB-UIDE puede verse en el capítulo 5.

3.4.1.3 Descripción de los objetivos de la adaptación en AB-UIDE

Los objetivos de la adaptación son las metas que el proceso de adaptación debe intentar perseguir (minimizar el número de errores, optimizar la

eficiencia, facilidad de uso, mostrar al usuario que desea ver, acelerar el uso, considerar la experiencia del usuario, etc).

La especificación de las metas que debe perseguir el proceso de adaptación es uno de los puntos menos tratados en la literatura. Dentro de AB-UIDE se ha optado por la especificación de dichas metas utilizando una notación basada en I*[Yu 97], ya que dicha notación fue diseñada precisamente para la captura de los requisitos de futuros sistemas utilizando un paradigma orientado a metas [Yue87].

La especificación de las metas de la adaptación es especialmente interesante dentro del proceso de adaptación, ya que permite establecer los criterios que permitan discernir qué adaptaciones van en contra del diseño establecido, y por lo tanto no deben aplicarse. Dentro de AB-UIDE esta especificación recibe el nombre de compromiso de usabilidad, ya que describe qué factores de usabilidad deben prevalecer sobre otros. Por ejemplo, si se reduce la superficie de visualización de la interfaz de usuario caben distintas alternativas: por una parte, por ejemplo, se podrían hacer más pequeñas las fuentes de texto, pero sin embargo eso podría contradecir los criterios de accesibilidad del sistema si las fuentes se hacen demasiado reducidas. Es necesario especificar en este caso, qué criterio debería prevalecer.

Una descripción detallada de la especificación del compromiso de usabilidad puede encontrarse en la sección 4.2.2.3.

3.4.1.4 Reglas de adaptación en AB-UIDE

Una adaptación representa un cambio en la interfaz de usuario, un cambio de sus contenidos, su presentación, etc. Por lo tanto, cualquier aproximación de diseño de interfaces de usuario que desee reflejar el diseño de la adaptación dentro de su método tendrá que permitir la especificación de los posibles cambios que se puedan aplicar a la interfaz de usuario, es decir, tendrá que permitir la descripción de las reglas de adaptación aplicables sobre la interfaz de usuario para producir una adaptación.

Dentro de AB-UIDE las reglas de adaptación se describen de acuerdo a los determinantes que las producen, una especificación de los valores que deben tomar los determinantes para disparar la adaptación, los datos que necesita la adaptación para ejecutarse, y finalmente, la propia adaptación.

En la sección puede encontrarse una descripción detallada del meta-modelo de regla de adaptación propuesto para AB-UIDE.

La especificación de la propia adaptación de la interfaz de usuario sigue un modelo transformacional como el propuesto en [Lim04][Lim04b][Lim05], en concreto un modelo basado en la transformación de grafos, mediante reescritura condicional de los mismos.

Adaptación de la interfaz de usuario por transformación de grafos

La adaptación de la interfaz de usuario por transformación de grafos consiste en la transformación de un grafo origen, que en nuestro caso representa la interfaz de usuario en su estado actual, en un grafo destino a través de la aplicación de una serie de reglas mediante la aplicación de técnicas de reescritura de grafos.

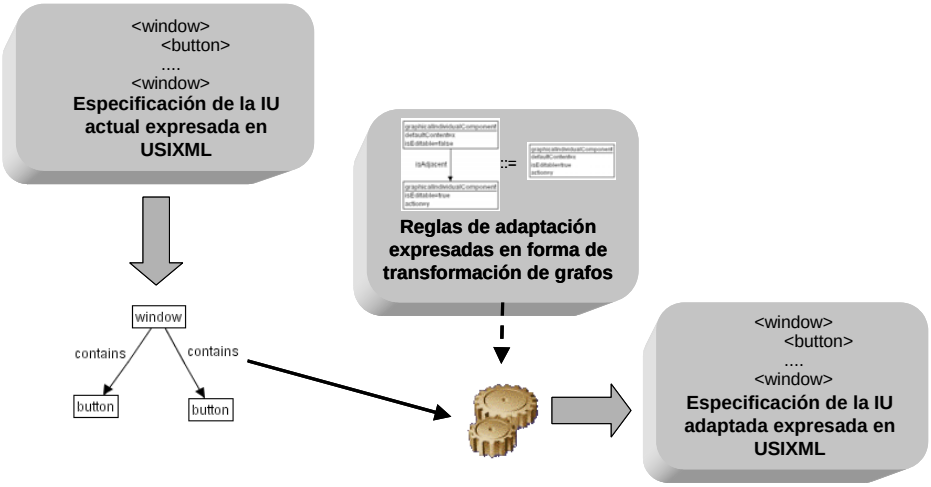


Figura 3.7 Modelo de adaptación de la interfaz de usuario por transformación de grafos.

La figura 3.7 describe le proceso de adaptación de una interfaz de usuario siguiendo un modelo de transformación de grafos. Como ya se ha descrito anteriormente, el lenguaje de especificación de interfaces de usuario usiXML sirve como ontología de interfaz de usuario donde se almacenan las interfaces de usuario diseñadas mediante AB-UIDE. Obsérvese que cualquier lenguaje basado en XML presenta una estructura jerárquica y es, por lo tanto, susceptible de ser representado en forma de grafo. En el caso de usiXML la representación en forma de grafo debe plasmar las relaciones. Por ejemplo, de contenencia o adyacencia que se den entre los distintos elementos de la interfaz de usuario. Nótese que algunas

relaciones entre los elementos de la especificación serán explícitamente especificadas en el modelo XML, mientras que otras serán implícitas (véase la figura 3.8). El grafo generado debe ser lógicamente un grafo con atributos y etiquetado, es decir, un grafo cuyos nodos y enlaces puedan contener atributos, y cuyos nodos y enlaces pueden ser etiquetados. En la conversión de una especificación de una interfaz de usuario expresada en un lenguaje basado en XML en un grafo, normalmente, cada elemento de la interfaz de usuario se convertirá en un nodo del grafo. Los atributos de cada elemento en la especificación XML conformarán los atributos del nodo correspondiente en el grafo. Las relacionen, explícitas o implícitas (como las relaciones de contenencia en el ejemplo de la figura 3.8), se transforman en enlaces dentro del grafo, donde de igual forma podrán o no tener atributos.

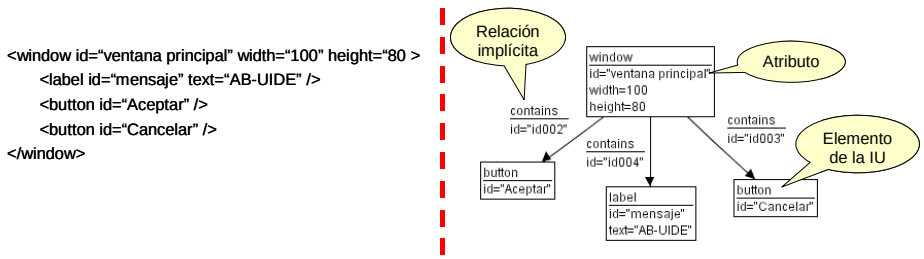


Figura 3.8 Ejemplo de conversión de una especificación de IU en un grafo con atributos.

Gramáticas de grafos

Las gramáticas de grafos fueron propuestas durante los años 70 como una manera de generalizar las gramáticas de cadenas propuestas por Chomsky [Cho57]. La idea que se perseguía era la extensión de la concatenación de cadenas a la concatenación de grafos. Para la especificación de los cambios que puede sufrir un grafo a lo largo del tiempo, se deben especificar una serie de reglas de producción que describan de qué maneras puede ser modificado el grafo. Por lo tanto, una gramática de grafos se define como: $GG=(G_0, P)$, donde G_0, será el grafo inicial sobre el que se aplican las reglas de producción, y P el conjunto de reglas de producción aplicables a a G_0.

Una regla de transformación de grafos o producción (L, K, R) [Löw93] consiste en tres grafos L, K y R.. El grafo L es habitualmente denominado

parte izquierda de la regla (*LHS – Left Hand-Side*), mientras que al grafo *R* se le suele denominar parte derecha de la regla (*RHS – Right Hand-Side*).

Una producción (*L*, *K*, *R*) es aplicable a un grafo *G* si *G* contiene un subgrafo que es una imagen de *L*. El grafo L que constituye la parte izquierda de la producción, y formula las condiciones bajo las cuales resulta aplicable la producción, es decir, es un subconjunto del conjunto de objetos (nodos y arcos) del universo. El grafo de contacto K, que generalmente es un subgrafo de L y de R, describe los subobjetos de la parte izquierda que se deben preservar en el proceso de aplicación de la producción. Por lo tanto, la diferencia entre L y K: L-K, contiene a todos aquellos subobjetos que se deben eliminar al aplicar la producción. Análogamente, la diferencia R-K contiene a todos aquellos subobjetos que se deberán agregar al aplicar la producción. Este grafo intermedio K describe el contexto en el cual se integran los subobjetos agregados. Nótese que por lo tanto mediante la definición de reglas de producción se pueden modificar los valores de los atributos de los nodos y arcos, y añadir o eliminar nuevos nodos o arcos.

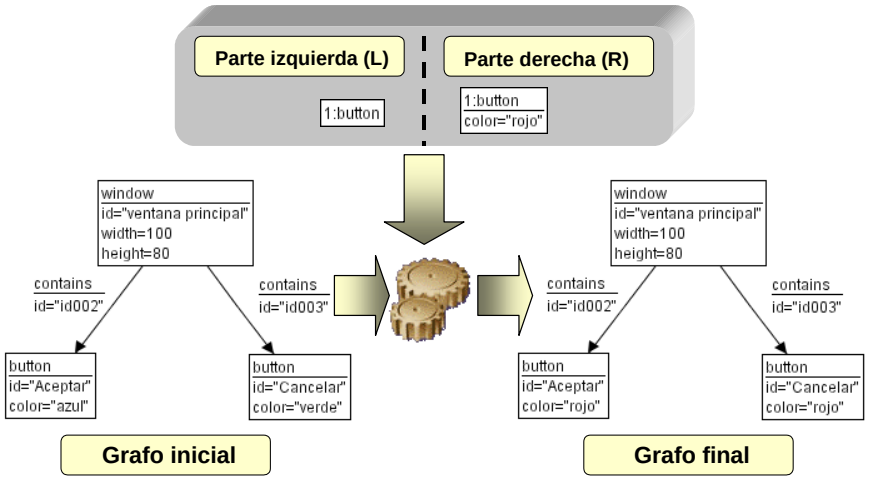


Figura 3.9 Ejemplo de regla de producción.

En la figura 3.9 se muestra un ejemplo de regla de producción, donde se especifica que cada nodo de tipo *button* (especificado en la parte izquierda), debe ser sustituido por un nodo de tipo *button* cuyo color será "rojo" (especificado en la parte derecha). Ya que en el grafo inicial existen dos nodos de tipo *button*, la regla de producción será aplicable sobre ambos. Y por lo tanto el nodo de tipo *button* cuyo atributo "color" era "verde"

parará a ser "rojo", y de forma análoga el nodo de tipo *button* cuyo atributo "color" era "azul" pasará a ser "rojo". Obsérvese que el orden de aplicación de la regla sobre dos nodos de tipo *button* no es determinista. La regla descrita permitiría asignar el color "rojo" a todos los botones de una interfaz.

Dentro de las gramáticas de grafos existen dos aproximaciones principales a la hora de la aplicación de una regla de producción: *double push-out*, *single push-out* [Löw93]. El enfoque llamado *single push-out* estudia la generalización del *double push-out*, relajando las condiciones de aplicación de las producciones. Como no existe la condición de contacto, este enfoque permite identificar objetos que sean candidatos a ser eliminados, y elimina los arcos que queden pendientes, garantizando así que el resultado siempre sea un grafo. Este enfoque interpreta una producción del enfoque *double push-out* con dos morfismos (totales) de grafos, como un único morfismo (parcial) de la parte izquierda a la parte derecha de la producción. Ello hace que la especificación de una regla siguiendo el modelo de *single push-out* sea más clara y sencilla. Por lo tanto, a lo largo de este trabajo se ha optado por utilizar el método de aplicación de reglas de producción basado en *single push-out*.

La reescritura de grafos mediante la aplicación de reglas de producción, mejora su capacidad de expresividad cuando se permite la especificación de condiciones que determinan cuando una regla de producción debe aplicarse, permitiendo la especificación, por ejemplo, de condiciones de parada en la aplicación iterativa de una regla de producción. La reescritura de grafos que permite la especificación de este tipo de condiciones es denominada reescritura de grafos condicional, y será la utilizada dentro de este trabajo [Hab96]. Estas condiciones pueden venir expresadas en forma de:

- **Precondiciones de aplicación positiva** (*PAC – Positive Application Condition*): son precondiciones que deben darse para que una regla pueda ser aplicable (por supuesto a continuación también se deben dar las condiciones expresadas en la parte izquierda de la regla). Habitualmente este tipo de precondiciones no es muy utilizado, ya que la mayoría de estas precondiciones pueden expresarse en la parte izquierda de la regla. En la figura 3.10 puede observarse una regla donde se ha incluido este tipo de precondición. Nótese que ahora el grafo final resultante no será el

mismo que en la regla especificada en la figura 3.9, ya que antes todos los botones pasaban a ser de color rojo independientemente del color que tuvieran inicialmente, mientras que ahora sólo pasarán a ser de color "rojo" aquellos cuyo color inicial fuera "verde", ya que primero se comprobará que se da la condición de aplicación positiva.

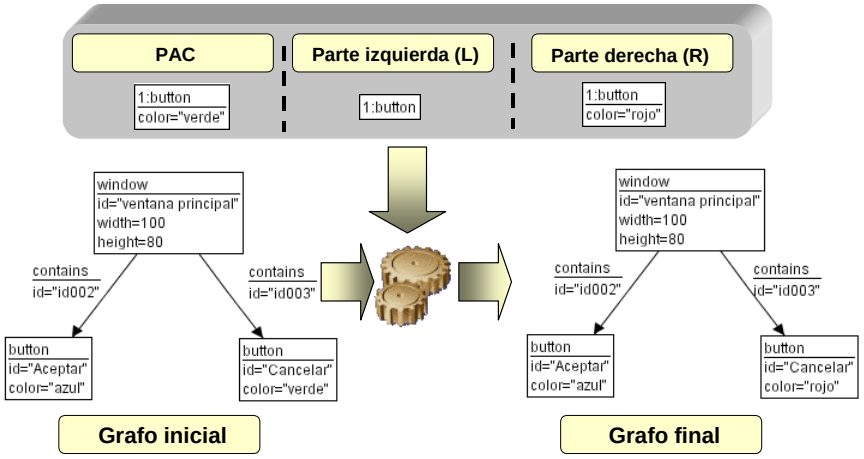


Figura 3.10 Ejemplo de regla de producción con PAC.

- **Precondiciones de aplicación negativa** (*NAC - Negative Application Conditions*): las condiciones de aplicación negativa funcionan de forma análoga a como funcionan las precondiciones de aplicación positiva. La única diferencia será que ahora para que una la regla que contiene la precondición de aplicación negativa sea aplicada, la precondición especificada no se debe dar en el grafo inicial. Obsérvese que ahora el grafo inicial y final son idénticos, ya que la regla de producción nunca llega a ejecutarse ya que la precondición especificada en el NAC, en la que se indica que no debe encontrarse un nodo de tipo *button* cuyo atributo "color" tenga el valor "verde", sí que se da en el grafo inicial.

- **Poscondiciones de aplicación positiva:** son poscondiciones que deben darse una vez aplicada la regla de producción.

- **Poscondiciones de aplicación negativa**: son poscondiciones que no deben darse una vez aplicada la regla de producción.

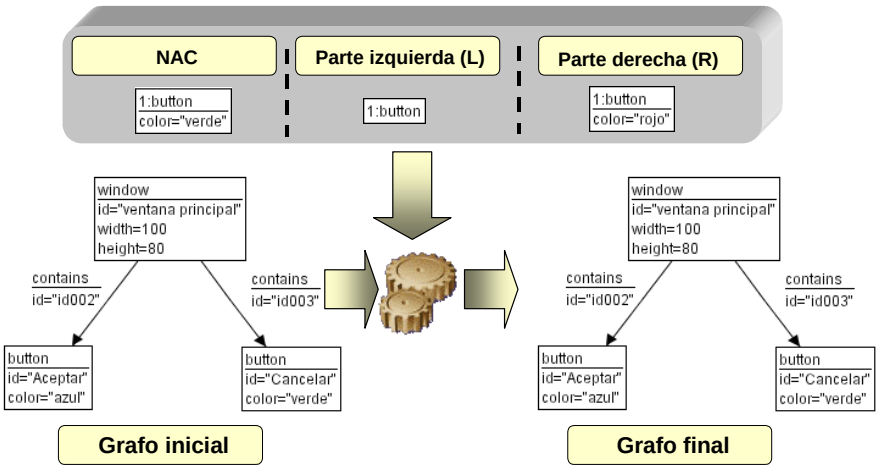


Figura 3.11 Ejemplo de regla de producción con NAC.

Con todo ello, la ejecución de una regla de producción dentro de un sistema de transformación de grafos con escritura condicional seguiría los siguiente pasos:

1. Encontrar una ocurrencia de la parte izquierda de la regla en el grafo inicial (*G*), es decir encontrar un morfismo entre *L* y el grafo inicial. Nótese, que por ejemplo en el caso de la regla de la figura 3.10 se encontrarían dos posibles emparejamientos (los dos botones). La elección de uno u otro primero será al azar.

2. Comprobar las precondiciones de aplicación positiva y negativa. En el caso de que no se cumplan la ejecución de la producción terminaría.

3. Eliminar del grafo inicial aquella parte que es igual a *L-K*. Como ya se describió anteriormente, *K* contiene aquellos elementos de la regla de producción que deben ser preservados por la transformación. $G_1 = G - (L\text{-}K)$.

4. Hacer $G_2 = G_1 - (R\text{-}K)$.

5. Incorporar (*R-K*) en G_1, tal como se especifique en la relación entre *R-K* y *K*.

6. Comprobar las poscondiciones de aplicación positiva y negativa. Si no se cumplen las poscondiciones, se tendrán que deshacer los pasos anteriores.

Especificación de adaptaciones en forma de transformaciones de grafos

En general se puede especificar cualquier tipo de adaptación utilizando la expresividad de las reglas de producción. Hay que tener en cuenta que la expresividad de las gramáticas de grafos con reescritura condicional es equivalente a la expresividad de cualquier lenguaje de programación habitual. A continuación se ejemplifican las operaciones más habituales dentro de la programación usando reescritura condicional de grafos.

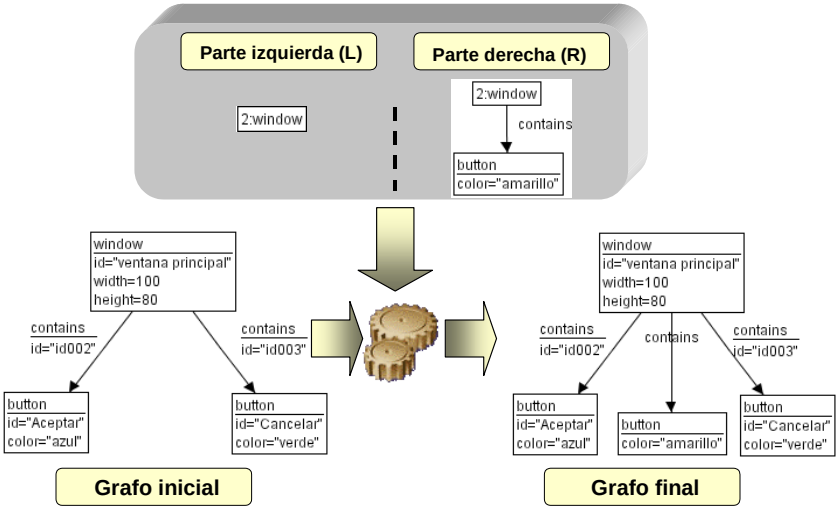


Figura 3.12 Regla con creación de nodos y enlaces.

- **Creación de nodos o enlaces**: todos los elementos que aparezcan en la parte derecha de una regla de producción, y que no tengan una correspondencia (*mapping*) con un elemento de la parte izquierda de la regla, serán añadidos al grafo inicial. El número dos junto al elemento *window* denota que ambos elementos se corresponden, es decir, que el elemento *window* que se use en la parte derecha de la regla debe ser el mismo que se deba encontrar al hacer el emparedamiento de la parte izquierda.

- **Eliminación de nodos o enlaces**: todos los elementos que aparezcan en la parte izquierda de una regla, y no aparezcan en la parte derecha, serán eliminados del grafo inicial. En la figura 3.13

se muestra un ejemplo de este tipo de comportamiento. En este caso, existe un nodo de tipo *window* que no es eliminado, mientras que todos los nodos de tipo *button* que tengan una relación de tipo *contains* con el nodo *window* emparejado se irán eliminando en sucesivas ejecuciones de la regla.

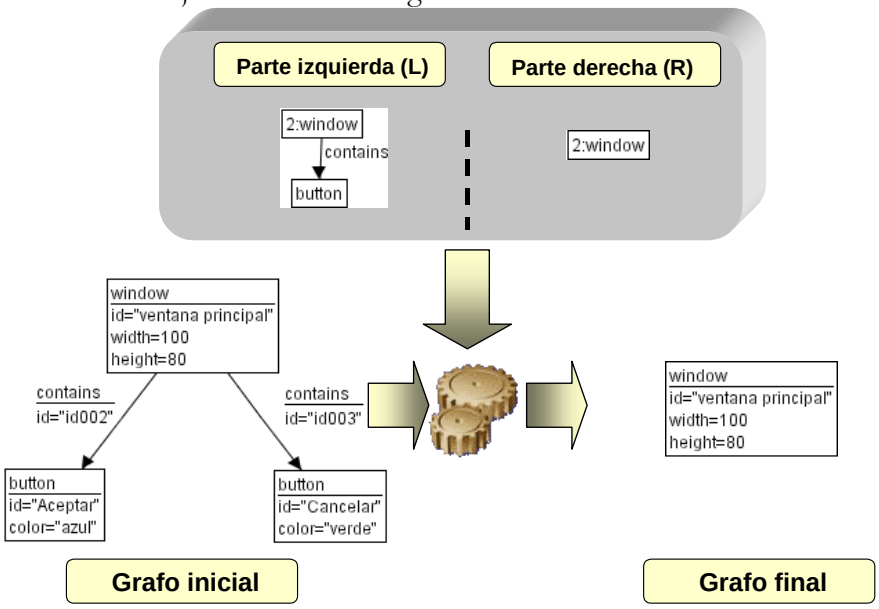


Figura 3.13 Regla con eliminación de nodos y enlaces.

- **Modificación de nodos o enlaces**: para modificar un nodo o un enlace, estos deberán preservarse en la aplicación de la regla, es decir, tendrá que existir una correspondencia (*mapping*). A continuación, los atributos de los nodos de la parte derecha que tengan una correspondencia con algún nodo de la parte izquierda, pueden tomar valores, que sobrescribirán los valores que tuviera el nodo emparejado en la parte izquierda inicialmente (véase la figura 3.14). Nótese que aquellos atributos que se dejen en blanco en el nodo de la parte derecha con el que existe la correspondencia no serán modificados, manteniendo los valores que tuvieran en el grafo inicial.

- **Utilización de variables**: la utilización de variables en la especificación de reglas permite realizar comparaciones entre los atributos de los distintos nodos y enlaces, y transferir los valores de unos atributos a otros, e incluso aplicarles operaciones. La figura 3.15 describe una transformación donde la altura del nodo

window será el doble de la anchura que dicho nodo tiene, además se crea un nuevo botón de color amarillo, y cuyo identificador es el identificador del nodo *window* que lo contien.

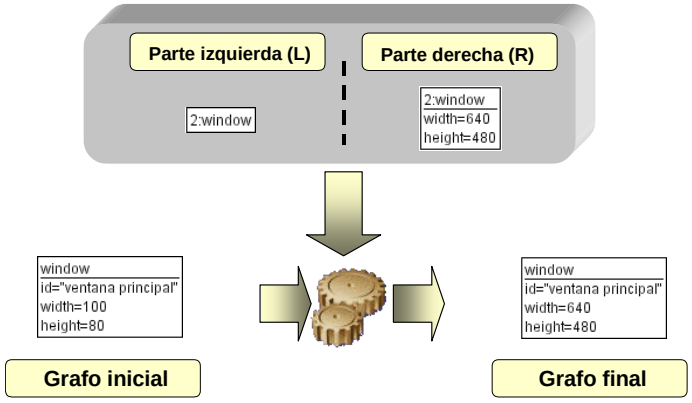


Figura 3.14 Regla con modificación de atributos.

Una vez mostrada una vista general de los constructores básicos utilizados para la descripción del método AB-UIDE, que será descrito con profundidad en el capítulo 4, a continuación se describirán los constructores básicos de la arquitectura de la interfaz de usuario diseñada para la ejecución de las interfaces de usuario adaptativas creada siguiendo el método AB-UIDE. Dicha arquitectura será descrita en detalle en el capítulo 5.

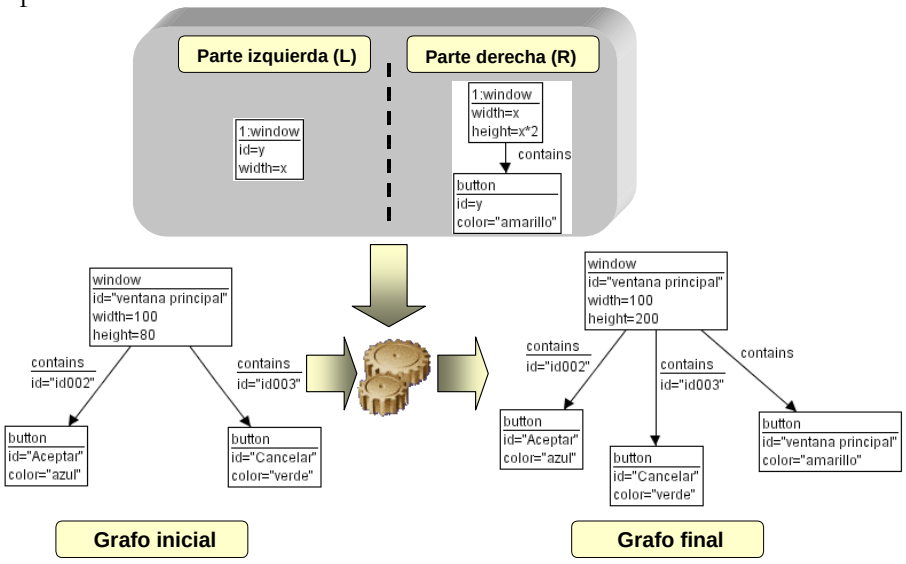


Figura 3.15 Regla con uso de variables.

3.4.2 Una Arquitectura de ejecución de interfaces de usuario adaptativas basadas en un sistema multi-agente

El término arquitectura es utilizado dentro de distintos contextos en la Ingeniería del Software, y es en general aplicable a todo tipo de sistemas. Dentro de esta tesis, el término será aplicado a la descripción de la interfaz de usuario.

Una arquitectura identifica los componentes y sus interrelaciones, e incluye el conocimiento de diseño relativo a la estructura de un sistema interactivo. Una arquitectura debería, idealmente, reunir el conocimiento de diseño y permitir su reutilización. Conforme la arquitectura se va haciendo más concreta, identifican componentes reutilizables y mecanismos para la comunicación de los datos y el control.

Dentro de esta tesis se propone una arquitectura de ejecución de interfaces de usuario adaptativas, creadas siguiendo el método propuesto: AB-UIDE. El requisito principal de esta arquitectura será la ejecución de las cuatro fases del proceso de adaptación (véase la sección 2.1.1). Por ello, debe ser capaz de detectar los cambios en el contexto de uso de la aplicación, reaccionar antes estos proponiendo alternativas de adaptación, seleccionar las mejores adaptaciones a aplicar, y finalmente, aplicarlas.

Debido a la necesidad de la toma de decisiones que dicha arquitectura necesita realizar, se ha propuesto una arquitectura basada en un sistema multi-agente (véase la sección 2.5), que permite la incorporación de un modelo mental de razonamiento basado en la teoría del razonamiento humano propuesta por Bratman [Bra87].

La arquitectura propuesta explota el conocimiento de la interfaz de usuario recopilado durante la construcción de la interfaz, de forma que se integra el diseño de la interfaz de usuario basado en modelos dentro del proceso de adaptación.

La figura 3.16 describe la estructura general propuesta para una aplicación adaptativa. La arquitectura está basada en el modelo clásico cliente/servidor. La plataforma cliente es la encargada de la visualización de la interfaz de usuario y de la captura de los cambios en el contexto de uso, a través de los sensores. Los sensores comunicarán cualquier cambio en el contexto a la plataforma servidor, diseñada en forma de sistema multi-agente, que decide la iniciación o no del proceso de adaptación, y en

caso afirmativo, realiza las fases de proposición, selección y ejecución de las adaptaciones. Para realizar estas fases, el sistema multi-agente aprovecha el conocimiento adquirido de la interfaz de usuario durante el proceso de diseño, y el conocimiento adquirido durante el uso de la aplicación (por ejemplo, la frecuencia de realización de una tarea).

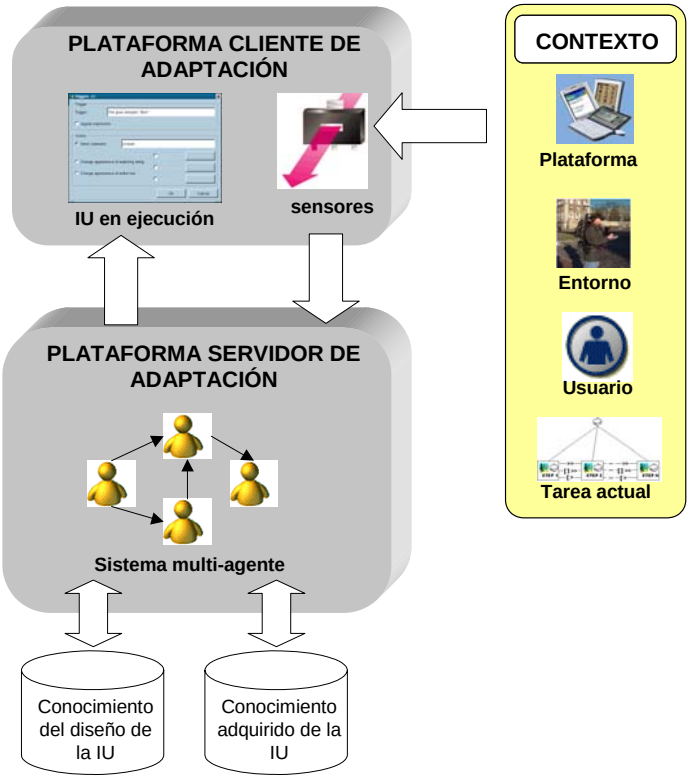


Figura 3.16 Arquitectura general propuesta para una aplicación adaptativa.

3.5 Conclusiones del capítulo

Los contenidos de este tercer capítulo ilustran de forma general la aproximación propuesta, tanto para el método de diseño de interfaces de usuario propuesto, como para la arquitectura para la ejecución de interfaces de usuario adaptativas basada en un sistema multi-agente necesaria para la ejecución de las interfaces de usuario adaptativas diseñadas.

CAPÍTULO 4
AB-UIDE: DESARROLLO BASADO EN MODELOS DE INTERFACES DE USUARIO ADAPTATIVAS

"El hombre razonable se adapta al mundo; el hombre no razonable se obstina en intentar adaptar el mundo a sí mismo. Todo progreso depende, pues, del hombre no razonable."
(George Bernard Shaw)

4.1 Introducción

No son desconocidas las ventajas de la aplicación de un método sistemático en la creación de cualquier aplicación software. Las interfaces de usuario son una parte del software especialmente difícil de diseñar [Mye93], y es por tanto, si cabe, más importante la aplicación de métodos que permita el diseño de una interfaz de usuario de calidad de una manera sistemática y guiada.

El paradigma de diseño basado en las arquitecturas guiadas por modelos, y más concretamente dentro del mundo de la interacción, de los métodos basados en modelos, intenta atajar las dificultades que afloran durante el diseño de una interfaz de usuario, y a su vez aporta nuevos beneficios: automatización o semiautomatización de la creación de la interfaz de usuario, reutilización de la experiencia, etc.

Dentro del paradigma de diseño basado en modelos de interfaces de usuario se encuadra el método propuesto, que está diseñado con el objetivo de complementar las aproximaciones basadas en modelos clásicas [Bod93][Pue97][Koc01][Loz00], con las características necesarias para la descripción de los constructores necesarios para la realización del proceso de adaptación.

En este capítulo se presenta AB-UIDE (*Agent-Based User Interface Development Environment*), una aproximación que extiende las actuales propuesta para el diseño de interfaces de usuario basadas en modelos para permitir la especificación de intefaces de usuario con capacidades de adaptación, y cuya salida sirve como entrada a una arquitectura de ejecución de interfaces de usuario adaptativas, cuyas capacidades de adaptación se basan en las los modelo diseñados durante el proceso propuesto en AB-UIDE.

4.2 AB-UIDE: un método basado en modelos para el desarrollo de interfaces de usuario adaptativas

AB-UIDE [Lop04] (*Agent-Based User Interface Development Environment*) es una aproximación basada en modelos que persigue extender los actuales métodos basados en modelos con los artefactos necesarios para el diseño de los elementos que permitan incluir un cierto grado de adaptatividad en una interfaz de usuario. El método permite el aprovechamiento de las ventajas que las aproximaciones basadas en modelos proporcionan en el diseño de las capacidades de adaptación de una interfaz de usuario, permitiendo, entre otras cosas, la generación automática de la implementación de los componentes encargados del proceso de adaptación, y la reutilización de la experiencia adquirida durante el diseño de la adaptación.

AB-UIDE se encuadra dentro de los métodos centrados en el usuario [Nor86]. Dentro de dichos métodos se trata de involucrar al usuario dentro del ciclo de vida de desarrollo de la interfaz de usuario en las primeras etapas del desarrollo, la evaluación empírica de las interfaces generadas, todo ello dentro de un ciclo de vida iterativo.

Sin embargo, hay que tener en cuenta, que un ciclo de vida iterativo, donde se itere sobre un diseño inicialmente de baja calidad, producirá habitualmente un resultado final peor que un diseño no iterativo donde se haya mantenido una alta calidad en el proceso de diseño. Es por ello que, dentro de AB-UIDE, se combinan las prácticas propuestas para el diseño centrado en el usuario con el diseño centrado en el uso [Con99], que trata de asegurar la calidad, no sólo mediante la evaluación con el usuario, sino también a través de la evaluación de la calidad del software generado utilizando métricas.

La figura 4.1 muestra las etapas de diseño propuestas en AB-UIDE. El proceso comienza con la etapa de adquisición de requisitos, guiada por los casos de uso identificados, donde se capturan los requisitos tanto funcionales como no funcionales para la futura aplicación. La adquisición de requisitos guiará el análisis del futuro sistema, donde se traducen los requisitos capturados en modelos que permitan la traducción de los requisitos en una representación homogénea y consistente, y en concreto describiendo el modelo de dominio de la aplicación, los perfiles de usuario del sistema, y finalmente el compromiso de usabilidad. La etapa de diseño crea una descripción abstracta detallada de la futura interfaz de usuario, mediante el modelado de las tareas de interacción, los objetos implicados en dichas tareas, la interfaz de usuario abstracto, su representación concreta, y las relaciones entre todos esos elementos, representadas usando el concepto de conector. Finalmente, la etapa de implementación genera el código de la aplicación que será ejecutado dentro de la arquitectura adaptativa propuesta (véase el capítulo 5).

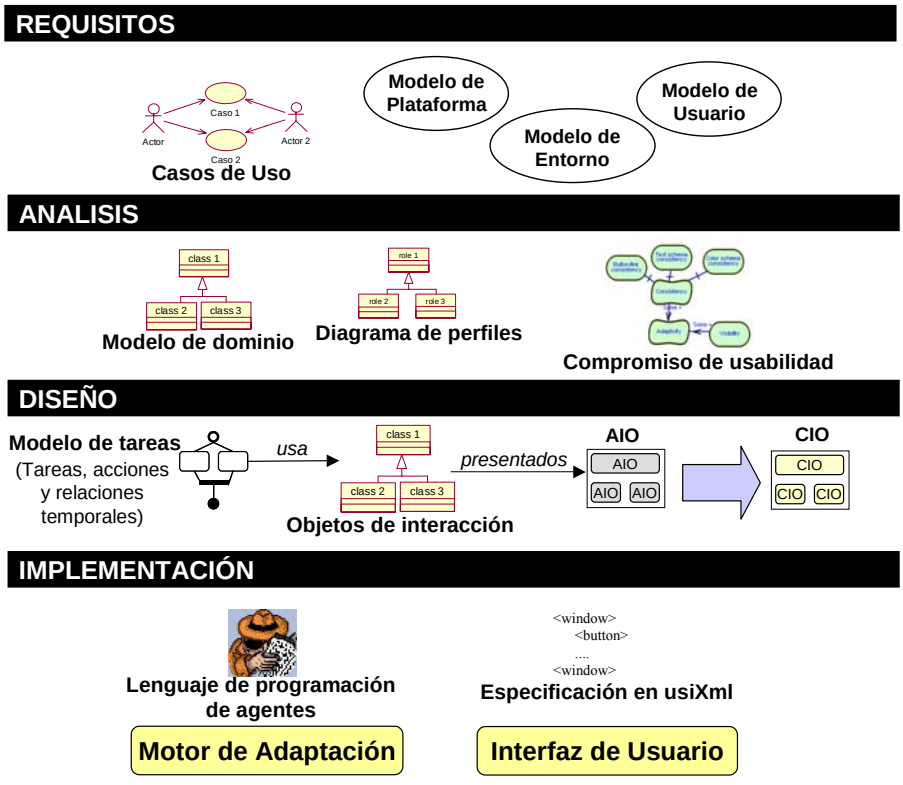


Figura 4.1 Fases en el ciclo de desarrollo propuesto en AB-UIDE.

A continuación se describe cada una de las etapas del ciclo de desarrollo propuesto de forma detallada. Para su ejemplificación se usará un ejemplo simple donde un usuario debe registrarse (*login*) en un sistema. Para ello el usuario tendrá que suministrar su nombre de usuario y contraseña, e instar al sistema a que los valide.

4.2.1 Fase de adquisición de requisitos

La fase de adquisición de requisitos es esencial para un correcto diseño. Una adquisición deficiente de los requisitos conducirá al diseño de sistemas incorrectos. Aún más, una adquisición de requisitos incompleta tendrá un impacto proporcionalmente exponencial sobre el coste de la aplicación dependiendo de en qué fase del desarrollo se encuentre la proceso. Unos requisitos erróneos detectados en las últimas fases del desarrollo introducirán un aumento importante en los costes y los plazos de entrega.

La importancia de la etapa de adquisición de requisitos ha dado lugar al surgimiento de la Ingeniería de Requisitos como una rama de la Ingeniería del Software. La Ingeniería de Requisitos es definida por Zave [Zav97] como: "*es la rama de la Ingeniería del Software que trata los objetivos del mundo real, las funcionalidades y las restricciones de los sistemas software. Trata también las relaciones entre estos factores para precisar las especificaciones del comportamiento del sistema, y su evolución a lo largo del tiempo y en distintas familias de software*".

4.2.1.1 Adquisición de requisitos en interacción persona-ordenador

La adquisición de los requisitos necesarios para el diseño de una interfaz de usuario necesita técnicas específicas capaces de capturar los requisitos no funcionales de usabilidad que introducen el factor humano dentro del diseño de la interfaz. Por ello, se ha propuesto una gran variedad de técnicas que van desde métodos textuales a métodos de prototipado rápido.

Dentro de AB-UIDE los objetivos del usuario cuando se utiliza la aplicación son descritos mediante la utilización de los *Casos de Uso* [Jac99]. Los casos de uso muestran de forma clara e intuitiva las metas del usuario, en un lenguaje que puede ser entendido tanto por el usuario como el desarrollador. Hay que tener en cuenta que los diagramas de casos de uso no son por sí mismos suficientes para la captura de todos los detalles, ya que tan sólo capturan qué debe hacer el sistema, pero no capturan ningún tipo de información sobre el flujo de control entre los distintos casos de

uso. Será por lo tanto necesario completar el diagrama de casos de uso con diagramas de secuencia que describan el flujo de control entre los distintos casos de uso. De igual forma será necesario acompañar cada caso de estudio con una descripción detallada de su propósito en lenguaje natural, y restricciones, así como de otros posibles métodos de documentación. Será especialmente útil documentar cada caso de uso mediante una imagen gráfica que el usuario identifique con ese objetivo a realizar, que puede ser capturada de una aplicación conocida para el usuario. A menudo la asociación de patrones de interacción como los propuestos en [Wel03][Tid02][Mon03] a los casos de uso identificados facilitará en gran medida el proceso de desarrollo. En la figura 4.2 se muestra un ejemplo de especificación para el caso de uso *Login* (figura 4.2a). Obsérvese cómo el caso de uso se ha documentado a través de una imagen gráfica (figura 4.2b) que ilustra la tarea a realizar, y que facilita en gran medida la discusión de los objetivos del sistema con el usuario.

De igual manera, los casos de uso pueden ser documentados y validados a través de las técnicas de prototipado. En [Gra04] se puede encontrar una detallada descripción de las técnicas de prototipado rápido más extendidas dentro del diseño de la interacción persona-ordenador.

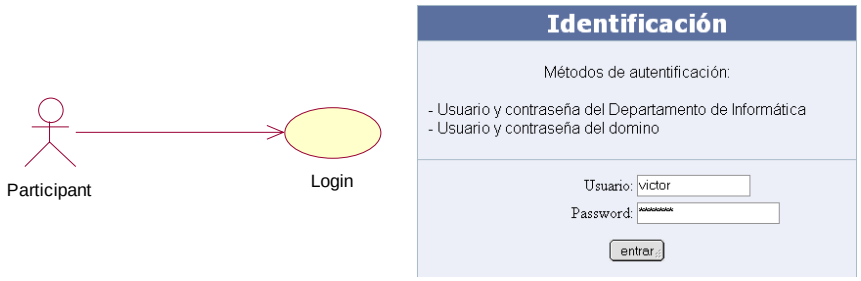


Figura 4.2 (a) Diagrama de casos de uso para la entrada en un sistema. (b) Imagen asociada al caso de uso *Login*.

4.2.1.2 Contextualización de la interacción

Uno de los factores determinantes para la creación de una interfaz de usuario de calidad es su adecuación al contexto de uso donde será usada. Ello implica que dentro de la captura de requisitos es necesario describir el contexto de uso de la aplicación. El contexto de uso estático viene definido por la descripción del usuario que interactuará con la aplicación, la plataforma tanto software como hardware donde tendrá lugar la

interacción y finalmente el entorno físico donde se sitúa la interacción (véase la sección 3.4.1.2).

Modelado de la plataforma en AB-UIDE

La descripción de la plataforma tanto software como hardware donde se produce la interacción es el pilar fundamental de las aplicaciones multiplataforma. La plataforma puede ser descrita usando el estándar CC/PP[20].

Sin embargo, la extensibilidad del modelo de plataforma, así como del resto de componentes de la descripción del contexto es una pieza clave en la flexibilidad necesaria para la especificación de sistemas adaptativos de forma.

Será necesaria una descripción de tantas plataformas como sea necesario. En general, la descripción por familias de plataformas será suficientemente detallada para una adaptación apropiada a la plataforma. En la figura 4.3 se muestra una descripción de dos posibles plataformas, caracterizadas mediante el número de color capaz de mostrar en pantalla simultáneamente (*colorDepth*), la resolución de la pantalla (*screenSize*) y la capacidad que tiene la plataforma para mostrar imágenes (*isImageCapable*).

Platform	**1**	**Platform**	**2**
colorDepth	256	colorDepth	2
screenSize (x,y)	800x600	screenSize (x,y)	320x200
isImageCapable	true	isImageCapable	false

Figura 4.3 Ejemplo de modelo de plataforma en AB-UIDE.

Modelado del entorno en AB-UIDE

El modelado del entorno será a menudo reutilizable de una aplicación a otra, ya que no existe una gran variedad de entornos habituales de interacción posibles, excepto para aplicaciones muy específicas.

[20] http://www.w3.org/Mobile/CCPP/

Las propiedades que describen un entorno son muchas; sin embargo, no existen demasiadas que incidan en la interacción. En el ejemplo de la figura 4.4 sólo se han considerado dos entornos de interacción posibles: en casa y en la calle. Como características que incidan en la interacción se han considerado el grado de luminosidad del ambiente, así como el nivel de ruido del entorno.

Home		**Street**	
lighting	high	lighting	medium
noisy	low	noisy	high

Figura 4.4 Ejemplo de modelo de entorno en AB-UIDE.

Modelado del usuario en AB-UIDE

La descripción del modelo de usuario debe ser dividida en características dependientes de la aplicación concreta y características independientes.

Por otra parte, el modelo de usuario en AB-UIDE también está dividido en características del usuario (*user features*) y relaciones (*relationships*) (véase la figura 4.5). Las características del usuario describen propiedades del usuario que son independientes del resto de las componentes del contexto. Las relaciones, sin embargo, describen características del usuario relacionadas con otras componentes de la descripción del contexto. Por ejemplo, en la figura 4.5 se describe cómo es la atención que el usuario puede prestar a la aplicación (*cognitiveFocus*) en dos entornos distintos (*Home* – en casa, y *Street* – en la calle).

La figura 4.5 describe dos características del usuario, por una parte describe si prefiere que los datos se le presenten de forma textual o no (*Preferentes.textualRepresentation*), y por otra parte describe posibles grados de ceguera de un usuario (*Physical.blindness*). Dicha figura describe igualmente cómo son las propiedades del usuario en relación con los modelos de plataforma y entorno. El usuario es inexperto en la utilización de la plataforma software 2, pero es usuario medio en la plataforma software 1 (*softwarePlatformExperince.Platform2/ softwarePlatformExperince.Platform1*). De forma similar, el usuario puede mantener una alta concentración en la

interacción con el sistema cuando está en casa, pero sólo podrá mantener una baja atención cuando esté en la calle (*cognitiveFocus.Home* / *cognitiveFocus.Street*).

User features	
Preferences.textualRepresentation	true
Physical.blindness	null

Relationships	
softwarePlatformExperience.Platform1	average
softwarePlatformExperience.Platform2	rookie
hardwarePlatformExperience.Platform1	expert
hardwarePlatformExperience.Platform2	rookie
cognitiveFocus.Home	high
cognitiveFocus.Street	low

Figura 4.5 Ejemplo de modelo de usuario en AB-UIDE.

Es importante resaltar que el modelo de usuario será habitualmente replicado para cada uno de los perfiles de usuarios identificados en la aplicación. Ante la imposibilidad de identificar los perfiles de usuarios futuros de la aplicación, el diseñador podrá optar por un solo perfil, y su especialización a lo largo del tiempo de acuerdo a los parámetros de la interacción captados.

4.2.2 Fase de análisis

A lo largo de la fase de análisis del ciclo de desarrollo de la interfaz de usuario en AB-UIDE se trata de modelar los requisitos capturados, tanto funcionales como no funcionales, en un lenguaje que no presente ambigüedades y pueda permitir un diseño detallado del sistema.

4.2.2.1 Modelado de los objetos del dominio

El diagrama de dominio consiste en la descripción de aquellos objetos con los que interactúan las tareas que el usuario realiza a través de la interfaz de usuario. Las dos notaciones más extendidas para la especificación de este tipo de modelos son los diagramas entidad/relación y los diagramas de clases de UML.

Dentro de AB-UIDE se propone la utilización de diagramas de clases como medio para la representación del modelo de dominio, ya que hoy en

día los métodos de diseño basados en el paradigma de orientación a objetos son más extendidos que los métodos basados en el modelo entidad/relación. Aunque cabe destacar que ambos modelos presentan una capacidades de expresidad equivalentes.

Para maximizar la utilidad del modelo de dominio se hace necesaria la descripción detallada de los tipos de datos involucrados tanto en los atributos de las clases como en los métodos. Los tipos de datos de los atributos y métodos junto con las tareas de interacción a las que están asociados permiten elegir los objetos concretos de interacción más apropiados a cada caso [Bod94]. En este sentido, la descripción de los tipos enumerados utilizados facilita en gran medida la selección de los objetos concretos de interacción apropiados.

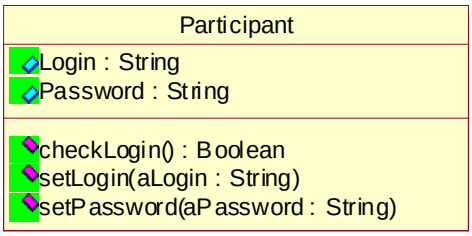


Figura 4.6 Ejemplo de modelo de dominio en AB-UIDE.

Siguiendo con el ejemplo del usuario intentado registrarse en el sistema, la figura 4.6 describe el modelo de dominio para la realización del caso de uso *Login* identificado en la fase de requisitos. El usuario (*Participant*) es descrito mediante un nombre de usuario (*Login*) y una contraseña (*Password*). Se pueden realizar tres acciones sobre ese tipo de objetos. Validad que el usuario y la contraseña son válidos (*checkLogin*), asignar el valor del nombre de usuario (*setLogin*) y asignar la contraseña al usuario (*setPassword*).

4.2.2.2 Los perfiles de usuario en el sistema

A menudo, existen distintos tipos de usuarios en el sistema que gozan de distintos privilegios o que poseen características que pueden ser especificadas estáticamente en tiempo de diseño. Si tomamos como ejemplo una aplicación de venta habitual, podemos encontrar distintos tipos de perfiles (*roles*) en los usuarios del sistema. Por supuesto, contamos con los usuarios/compradores, pero también se cuenta con las personas encargados de mantener el stock, o el gestor de la tienda. No sería lógico que todos los perfiles de usuario tuvieran acceso a todo el sistema, por

ejemplo, el comprador no debe ser capaz de acceder a las zonas de gestión de la tienda. Por otra parte obsérvese que el gestor del sistema debería ser capaz de supervisar el stock disponible así como el aspecto actual de la tienda, es decir, es capaz de asumir otros perfiles, introduciéndose por lo tanto el concepto de herencia entre los perfiles.

Para afrontar el modelado de los perfiles del sistema y sus relaciones se ha optado por un diagrama de clases UML por su versatilidad a la hora de representar relaciones entre entidades (perfiles) y especialmente su capacidad de expresión de relaciones de herencia. La figura 4.7 describe un posible modelo de perfiles para el ejemplo descrito anteriormente.

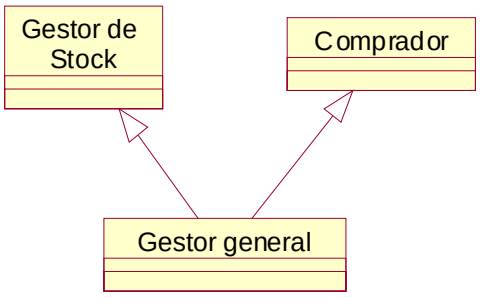


Figura 4.7 Ejemplo de modelo de perfiles en AB-UIDE.

Esta relación entre los perfiles establece el primer nivel de personalización del diseño. Los distintos perfiles identificados tendrán que ser descritos en forma de modelo de usuario tal y como se describe en la sección 4.2.1.2.

4.2.2.3 Un compromiso entre adaptación y usabilidad

Cuando se desarrolla la interfaz de usuario para una nueva aplicación, se aplican una serie de técnicas que persiguen la creación de interfaces de usuario con un alto grado de usabilidad. Entre ellas podemos destacar:

- La aplicación de métodos centrados en el usuario [Nor86], donde el usuario es incluido dentro del proceso de diseño permitiendo validaciones de los distintos prototipos a distintos niveles de detalle por parte del usuario.

- La aplicación de directivas que permitan la reutilización de la experiencia acumulada por los desarrolladores durante el desarrollo de aplicaciones a lo largo de los años. Esta experiencia puede ser expresada en forma de guías de estilo

> [Shn92][IBM92][Smi86], patrones [Duy02][Mon03][Tid99][Tid99] [Wel03] u otros métodos [Bas95], como por ejemplo en forma de transformaciones que se aplican para convertir un modelo en otros a distintos o iguales niveles de abstracción. Estas transformaciones pueden venir representadas siguiendo distintos formalismos, como pueden ser la reescritura de términos, las hojas de transformaciones XSLT, o las transformaciones de grafos como en [Lim04].

Dentro de una interfaz de usuario adaptativa, la interfaz de usuario evoluciona, y dicha evolución implica indudablemente una posible modificación en los criterios de usabilidad aplicados durante el diseño de la interfaz de usuario. Sin embargo, la evolución de los criterios de usabilidad de la interfaz de usuario no puede dejarse al azar, sino que debe ser posible definir cómo dichos criterios pueden evolucionar a lo largo del proceso de adaptación de una interfaz de usuario, pudiendo restringir qué tipo de variaciones en los criterios de usabilidad modificados son válidos y cuáles no. Por ejemplo, cuando una interfaz de usuario sufre un cambio de tamaño del espacio donde está siendo visualizada, es necesario evaluar las posibles adaptaciones de la interfaz de usuario al nuevo tamaño. Es importante destacar que dependiendo del contexto actual de uso la interfaz de usuario evolucionará de distintas formas. Las distintas formas en las que puede evolucionar la interfaz de usuario deben ser controladas en términos de los criterios de usabilidad que deben ser maximizados para cada plataforma. Es necesaria la especificación de qué criterios deben ser priorizados, ya que aunque idealmente todos los criterios de usabilidad son importantes, a menudo, el aumento de uno de ellos conduce a la reducción de otro. Por ejemplo, si queremos maximizar el criterio de visibilidad, puede ser contradictorio con la maximización del criterio de accesibilidad. Si se desea mantener unos tamaños de fuente suficientemente grandes para mejorar la interfaz de usuario para personas con dificultades de visión, no siempre será compatible con mantener la visibilidad de todos los elementos necesarios para realizar la tarea actual.

Por lo tanto se hace necesaria la especificación a priori de los criterios de usabilidad que deben ser maximizados cuando se produce una adaptación como respuesta a un cambio en el contexto actual de uso (denominado en AB-UIDE compromiso de usabilidad – *usability trade-off*).

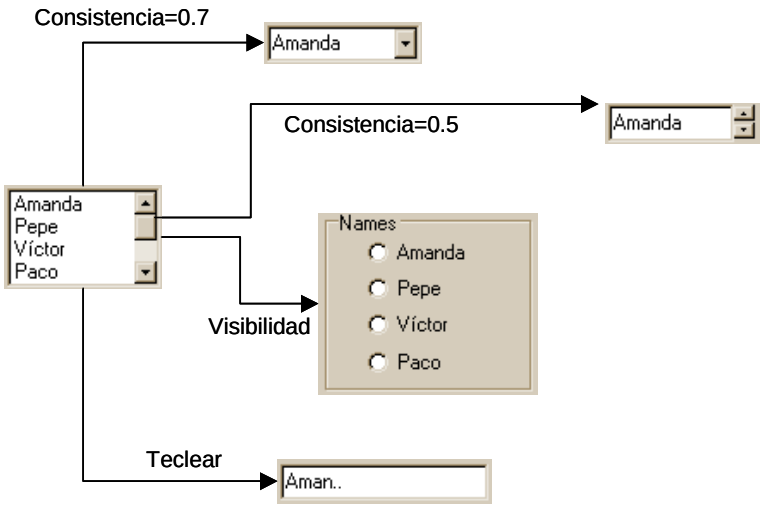


Figura 4.8 Ejemplo de distintas opciones de adaptación dependiendo de distintos criterios de usabilidad.

La figura 4.8 describe las distintas posibilidades de adaptación de una lista de selección y los distintos criterios a los que afecta. Obsérvese como distintas adaptaciones afectan distintos parámetros, o los mismos pero en distinto grado. Por ejemplo, si se convierte la lista de selección en una lista desplegable, se mantiene un alto grado de consistencia, ya que ambos elementos son manejados usando técnicas de interacción muy similares. Sin embargo, si se convierte la lista de selección en un *spin*, la consistencia sigue siendo aceptable (0.5), aunque menor que en el caso anterio, ya que ahora la técnica de interacción es algo distinta, en vez de elegir una opción entre todas las opciones visualizadas simultáneamente, el usuario debe ir pasando de una a otro con los botones de arriba y abajo del *spin*. Si lo que desea mantener es la visibilidad, la conversión a un grupo de botones de radio sería una opción viable, ya que todas las opciones posibles son visibles simultáneamente como en el *widget* original. Finalmente obsérvese como si eligiéramos seleccionar la opción mediante la escritura en una caja de texto, se estaría eligiendo una técnicas de interacción distinta de la original, y además la visibilidad sería drásticamente disminuida. La especificación del compromiso de usabilidad posibilita el modelado de los criterios que el sistema debe seguir para elegir entre las distintas posibilidad de adaptación que se presentan.

Especificación del compromiso de usabilidad

La especificación del compromiso de usabilidad *(usability trade-off)* implica la descripción de los requisitos de usabilidad para cada plataforma (o tipo de plataforma) donde la aplicación puede ser ejecutada.

Como ya hemos visto, dentro de la Ingeniería de Requisitos [Lam00] se persigue la adecuada captura de los requisitos de un sistema así como su transformación en una entrada válida y utilizable para la fase de análisis del sistema. La Ingeniería de Requisitos se encarga de la identificación de los objetivos que deben ser alcanzados por el futuro sistema, la operacionalización de dichos objetivos como servicios y restricciones, y la asignación de las responsabilidades para los requisitos resultantes a agentes (humanos, dispositivos o software). Los requisitos de un sistema se suelen clasificar en requisitos funcionales y requisitos no funcionales. Los requisitos funcionales engloban las funcionales que se le suponen al futuro sistema, mientras que los requisitos no funcionales incluyen características como: seguridad, fiabilidad, usabilidad, flexibilidad, robustez, interoperabilidad, coste, mantenimiento, etc.

En el caso que nos ocupa se persigue la especificación de los criterios de usabilidad que el motor de adaptatividad debe preservar durante la evolución de la interfaz de usuario durante su adaptación. Para ello se propone una variante de *Goal-oriented Requirement Language* (GRL) [Yu 04] (véase la sección 2.3.3.3), basado en la notación I* [Yu 97], y el entorno NFR [Chu97], que permite al diseñador capturar los requisitos del compromiso de usabilidad, así como su documentación.

Requisitos orientados a objetivos en interfaces de usuario adaptativas

La especificación del compromiso entre los distintos criterios que componen la usabilidad del futuro sistema es realizada usando una variante de la notación GRL.

Dentro de AB-UIDE se describe un compromiso de usabilidad para cada uno de los tipos de plataformas destino. El compromiso de usabilidad no tiene por qué ser igual para un ordenador de sobremesa que para una PDA, ya que las características de ambas plataformas son totalmente distintas.

Los criterios de usabilidad son representados mediante *objetivos* y *objetivos no funcionales*, que contribuyen al objetivo final (el criterio de adaptatividad) (véase la figura 4.9). De esta manera, la prioridad que un criterio de usabilidad debe tener dentro del futuro sistema a la hora de la aplicación de una adaptación es representada mediante relaciones de contribución (debe destacarse que las relaciones de contribución de los tipos *hurt*, *some-*, *unknown* y *equals* no son permitidos dentro de modelo) de cada uno de los criterios al objetivo general de adaptatividad. Los criterios pueden ser descompuestos en subcriterios, para conseguir una mayor precisión en la especificación del compromiso de usabilidad. Aquellos criterios de usabilidad que puedan ser evaluados directamente serán representados usando *objetivos*, mientras que aquellos que dependen para su evaluación de una serie de subcriterios serán representados usando *objetivos no funcionales*.

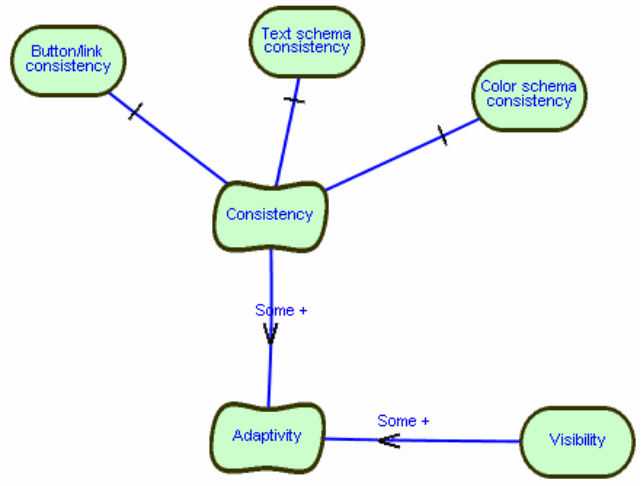


Figura 4.9 Los criterios de usabilidad contribuyen al objetivo principal: adaptatividad.

En la figura 4.9 se especifica un posible compromiso de usabilidad. En este caso dos objetivos contribuyen al objetivo (no funcional) de conseguir adaptatividad *(adaptivity)*: consistencia *(consistency)* y visibilidad *(visibility)*. Obsérvese como la visibilidad es especificada mediante un objetivo, ya que el diseñador considera que es directamente evaluable, sin embargo la consistencia es representada en forma de objetivo no funcional, ya que no es directamente evaluable. Para su evaluación se descompone en tres objetivos evaluables: (1) consistencia de los botones/enlaces *(Button/link consistency)*, (2) consistencia del esquema de texto *(Text schema consistency)*, y (3) consistencia del esquema de colores *(Color schema consistency)*. Tanto el criterio de consistencia como el de

visibilidad tendrán en este caso el mismo peso (positivo en este caso) durante la evaluación del compromiso de usabilidad *(some+)*.
El diseñador puede especificar de igual manera cómo unos criterios influyen en otros usando las relaciones de correlación disponibles en GRL (véase la figura 4.10). Las relaciones de correlación de los tipos *equals* y *unknown* no son contempladas dentro del modelo. Las relaciones de correlación no expresan dependencias explícitas entre los criterios de usabilidad, sino relaciones implícitas que aparecen entre los criterios, habitualmente no deseables. De esta manera el diseñador identifica cuáles son los conflictos entre los distintos criterios que conforman el compromiso de usabilidad. En el ejemplo de la figura 4.10 el diseñador ha especificado que el criterio de accesibilidad va en contra del criterio de visibilidad (*hurt)*. Por lo tanto, cuando se evalúe la visibilidad, el valor obtenido será influido negativamente por el valor de accesibilidad obtenido. Si por el contrario el diseñador hubiera especificado una relación de correlación del tipo *Help* (ayuda), el valor obtenido en la evaluación de la visibilidad hubiera sido influido positivamente por el valor obtenido al evaluar la accesibilidad.

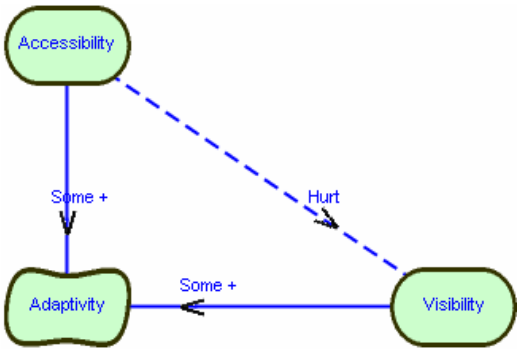


Figura 4.10 El criterio de accesibilidad tiene un impacto negativo sobre la visibilidad.

Para mejorar la comunicación entre los desarrolladores y la documentación del proyecto, el diseñador puede documentar las decisiones que ha tomado durante el diseño del compromiso de usabilidad añadiendo creencias (*beliefs*) dentro de la propia especificación del compromiso de usabilidad. En el ejemplo de la figura 4.11 se ha documentado la contribución de la visibilidad mediante una creencia que explica la razón por la que se ha diseñado así la contribución de ese criterio.

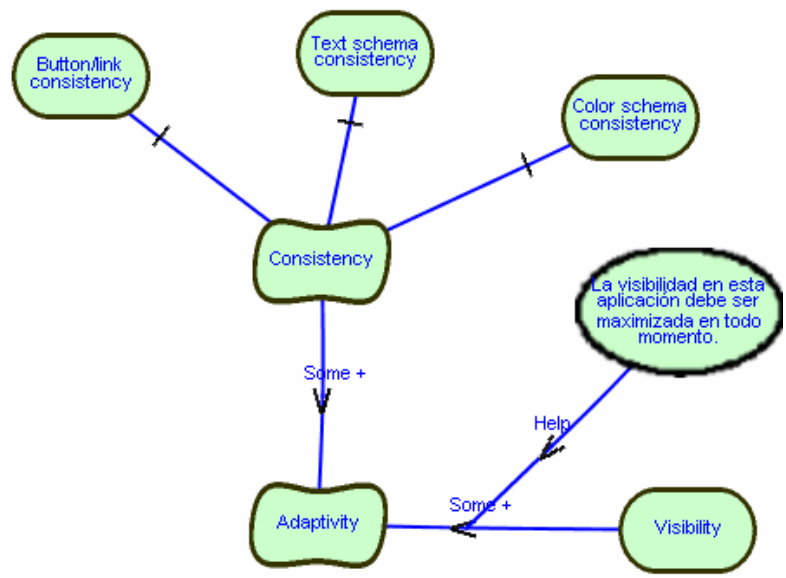


Figura 4.11 Documentación de la contribución del criterio de visibilidad con creencias.

Criterios de usabilidad

Para el modelado del compromiso se pueden usar cualquiera de los criterios que describen la usabilidad del sistema. Para facilitar la tarea del diseñador se ha realizado una recopilación de los criterios de usabilidad más ampliamente utilizados [Gra96][Gra00][Cal01a][Maj97].

Ya que el compromiso de usabilidad describe un modelo de calidad que debe ser evaluado en tiempo de ejecución, a continuación se propone el modelo de calidad usado por defecto por el sistema.

Los criterios de usabilidad recopilados han sido asociados a aquellos criterios ergonómicos [Bas95] sobre los que tienen un mayor impacto, para guiar al diseñador en la consecución de aquellos criterios que se desee maximizar. Dichos criterios de usabilidad han sido refinados con una serie de subcriterios que permiten la especificación de un modelo de calidad más preciso, y más fácilmente evaluable. La tabla 3 muestra el modelo de calidad propuesto, con los criterios ergonómicos, los criterios de usabilidad asociadas a cada criterio ergonómico, y los subcriterios que refinan los criterios de usabilidad.

Tabla 3. Criterios de usabilidad en el compromiso de usabilidad, y criterios ergonómicos.

Criterio ergonómico	Criterio de usabilidad en la adaptación	Subcriterio de usabilidad en la adaptación
1. Compatibilidad	1. Completitud de las tareas 2. Accesibilidad 3. Visibilidad 4. Multiplicidad de dispositivos 5. Multiplicidad de representación	
2. Consistencia	1. Consistencia en la distribución (*layout)*	1.1 Uniformidad en distribución 1.2 Densidad de contención 1.3 Márgenes 1.4 Relación de aspecto 1.5 Equilibrio de áreas
	2. Continuidad	
3. Carga de trabajo	1. Migrabilidad	
4. Control de Diálogo	1. Alcanzabilidad 2. Multitarea	
5. Adaptación	1. Adaptabilidad 2. Multiplicidad de perfiles 3. Preservación	
6. Representatividad	1. Consistencia del esquema de colores 2. Consistencia del esquema de texto 3. Consistencia del esquema de botones/enlaces	1.1 Color de fondo 1.2 Color en primer plano 2.1 Tipos de letra 2.2 Abreviaturas 3.1 Patrón de botones/enlaces 3.2 Esquema de colores de botones/enlaces 3.3 Esquema de texto de botones/enlaces
7. Orientación	1. Preentividad 2. Insistencia 3. Reutilización de entradas/salidas	
8. Manejo de errores	1. Tolerancia a las anomalías	

A continuación se describe cada uno de los criterios y subcriterios de usabilidad considerados:

1 **Compatibilidad**: el criterio de compatibilidad se refiere a la correspondencia entre las características del usuario (memoria, percepción, costumbres, habilidades, expectativas, etc) y las características de las tareas, y la organización de las entradas, salidas y el diálogo para una aplicación dada.

1.1 Completitud de las tareas: todas las tareas disponibles antes de realizar una adaptación deberían ser accesibles tras la aplicación de la adaptación. Si se desea preservar completamente esta propiedad, cualquier adaptación no debe ocultar ninguna tarea al usuario. En ocasiones, funcionalidades avanzadas o peligrosas pueden ser ocultadas a usuarios inexpertos para evitar resultados indeseados. Otro posible ejemplo es cuando una adaptación debe conmutar de una plataforma a otra. Podrá haber casos en que ciertas tareas no estarán disponibles tras la aplicación de las adaptaciones porque no puedan ser realizadas en la nueva plataforma.

1.2 Accesibilidad: accesibilidad supone que cualquier persona sea capaz de interactuar con la interfaz de usuario, incluso personas con algún tipo de deficiencia (por ejemplo, con dificultades de visión o motrices). Conforme el nivel de accesibilidad aumenta, las diferencias de facilidad de uso entre los distintos usuarios potenciales del sistema aumenta. Debe tenerse en cuenta que aunque una persona no tenga dificultades visuales, sí que puede tenerlas transitoriamente cuando se están realizando ciertas tareas. Por ejemplo, una persona conduciendo tendrá reducidas sus capacidades de visión de cara al manejo de una interfaz de usuario, ya que su atención estará centrada en la carretera, y sólo podrá mirar a la interfaz de usuario de forma interrumpida.

1.3 Visibilidad (observabilidad): es la capacidad del sistema de mostrar todos los objetos necesarios para realizar una tarea de interacción. Ello no significa que todos los datos necesarios para la realización de la tarea deban ser mostradas de una vez, sino que datos adicionales necesarios pueden ser inspeccionados por el usuario navegando en varias etapas. Por ejemplo, si deseamos seleccionar el tipo de habitación en la reserva a un hotel, una lista desplegable tendrá una visibilidad menor que un grupo de botones de radio, ya que para mostrar todas las opciones posibles el usuario tendrá que mostrar las opciones explícitamente, mientras que en el caso de los botones de radio se muestra todas las opciones posibles sin necesidad de que el usuario necesite intervenir explícitamente.

Una de las claves en la definición de la visibilidad de una interfaz de usuario es restringir la información mostrada en cada momento

a la información relevante para la tarea actual. Por lo tanto, cualquier adaptación aplicada debería intentar restringir la visibilidad a los datos relevantes. Para ello, las adaptaciones deben explotar las relaciones definidas durante el diseño del sistema entre el modelo de tareas y los datos que manipulan del modelo de dominio. Los datos relevantes deben ser como máximo el conjunto de datos que son accesibles en cada momento, es decir, el conjunto de datos del dominio que necesitan el conjunto de tareas activas (*enabled task set*) (véase la sección 2.2.2.5).

1.4 Multiplicidad de dispositivos: es la capacidad del sistema de ofrecer distintas posibilidades de dispositivos de entrada y salida. Por ejemplo, muchas órdenes en las interfaces de usuario *WIMP* pueden ser realizadas con el teclado o con el ratón.

1.5 Multiplicidad de representación: es la capacidad del sistema para proporcionar al usuario representaciones alternativas para las entradas y las salidas. Por ejemplo, la hora puede ser representada usando una imagen de un reloj, o como texto. Un ejemplo de entrada con multiplicidad de representación puede ser una aplicación de dibujo, donde el usuario puede dibujar una línea usando el ratón o escribiendo las coordenadas para el punto inicial y el final.

2 Consistencia: el criterio de consistencia se refiere a la manera en que las elecciones de diseño (código, nombrado, formatos, procedimientos, etc) se mantienen en contextos similares, y son distintos en contextos diferentes.

Existen distintos tipos de consistencia. Por ejemplo, para maximizar la consistencia en la sustitución de *widgets* es necesario sustituir los elementos de la interfaz de usuario con elementos con las características más parecidas. La consistencia puede ser vista desde dos puntos de vista. Por un lado, la consistencia de la presentación en un momento concreto puede ser considerada, pero por otro lado también se puede considerar la consistencia entre dos presentaciones en dos momentos distintos. En el caso que nos ocupa, donde tenemos una presentación antes de la adaptación y otra presentación distinta tras la adaptación, tendremos que centrarnos en este segundo tipo de consistencia descrita. Un ejemplo de adaptación en la que se preserve

la consistencia en un cierto grado sería sustituir una lista desplegable con un *spin*, ya que tienen similaridades tanto en la manera en la que muestran la información como en la manera en la que se interactúa con ellos.

2.1 Consistencia en la distribución (*layout*): se refiere a la manera en que los elementos de la interfaz de usuario son distribuidos para mejorar la consistencia de la interfaz de usuario.

2.1.1 Uniformidad en la distribución: representa la uniformidad en la distribución de los elementos de la interfaz de usuario. Se evalúa de acuerdo a las alturas, anchuras y alineaciones de los *widgets*. La uniformidad para un ventana concreta puede ser calculada utilizando la métrica de uniformidad propuesta por Constantine y Lockwood [Con99].

2.1.2 Densidad de contenencia: determina cuántos elementos están contenidos en cada contenedor. Un número alto para este criterio significa que hay muchos elementos dentro de cada contenedor, lo cual puede llegar a confundir al usuario debido a la falta de una estructura clara y de una sobrecarga de información.

2.1.3 Márgenes: es deseable mantener los mismo márgenes para los *widgets* en los diferentes diálogos y ventanas.

2.1.4 Relación de aspecto: las ventanas de diálogo para una misma tarea deben mantener la misma relación de aspecto para hacer fácil la diferenciación de unas tareas a otras.

2.1.5 Equilibrio de áreas: especifica cómo de equilibrada es la distribución de los componentes en las ventanas. Distribuciones equilibradas mejoran la estructura de la información y reducen la carga cognitiva.

2.2 Continuidad: la continuidad en la interacción no se da cuando un usuario se ve obligado a dividir su atención entre dos entidades. Por ejemplo, la continuidad se preserva en gran medida si una adaptación sustituye un *widget* por otro, que ocupa el mismo espacio en la pantalla que el original, ya que el usuario no necesita

desplazar su atención de la misma zona de la pantalla. Sin embargo, si un *widget*, por ejemplo una lista desplegable, es sustituida por un grupo de botones de radio con una cantidad moderada de elementos, el usuario necesita volver a centrar su atención en el lugar apropiado antes de continuar con el uso normal de la aplicación.

3 **Carga de trabajo**: el criterio de carga de trabajo tiene que ver con los elementos de la interfaz de usuario que influyen en la reducción de la carga cognitiva o perceptiva que el usuario debe emplear para utilizar la interfaz de usuario.

3.1 Migrabilidad: es la capacidad del sistema para permitir al usuario/sistema transferir la responsabilidad de una tarea. Esta es especialmente interesante en interfaces de usuario que incluyen agentes software, donde un agente, por ejemplo, el usuario, puede pasar el control de una tarea a otro agente para completar la tarea. Por ejemplo, en el nivel físico de abstracción, el usuario puede delegar la tarea de completar la escritura de las órdenes al sistema. En el nivel funcional de abstracción, guardar un fichero puede ser hecho explícitamente por el usuario o se puede permitir al sistema manejar la tarea de guardar el fichero en disco de vez en cuando conforme el sistema piense que es necesario.

4 **Control de diálogo:** el criterio de control de diálogo se refiere tanto al procesamiento por parte del sistema de las acciones explícitas del usuario, como al control que los usuarios tienen sobre cómo realizar sus acciones el sistema.

4.1 Alcanzabilidad: la capacidad del sistema de permitir a los usuarios alcanzar cualquier estado observable, independientemente del estado actual. El usuario puede desear alcanzabilidad inversa para poder volver a un estado anterior de la interacción después de hacer un error o de descubrir la necesidad de información disponible en un estado anterior. Ello requiere mantener un historial de la interacción. Este tipo de alcanzabilidad está presente en todos los navegadores de Internet mediante el uso del botón "Atrás" o "Ir a página anterior".

La alcanzabilidad hacia delante significa que el usuario puede acceder a cualquier estado de interacción, independientemente del desarrollo de los diálogos anteriores. Un sistema tiene una buena alcanzabilidad si el usuario puede navegar de un estado observable a otro con un esfuerzo que es aceptable para las expectativas del usuario. La alcanzabilidad del sistema está íntimamente relacionada con la preentividad (véase el criterio 7.1) y la visibilidad de las tareas. Tanto la preentividad como la visibilidad de las tareas pueden evitar que el usuario alcance cualquier estado observable posible.

4.2 Multitarea: es la capacidad del sistema de permitir que el usuario realice varias tareas a la misma vez, de forma que se superponen en el tiempo. Este factor debe ser considerado en el diseño de interfaces de usuario adaptativas, especialmente cuando se cambie la visibilidad de los objetos de interacción, ya que puede evitar que el usuario pueda realizar varias tareas a la vez.

5 Adaptación: la adaptación del sistema se refiere a la capacidad del sistema para comportarse de acuerdo al contexto de uso de la aplicación.

5.1 Adaptabilidad: la capacidad del sistema de permitir la personalización de la interacción por parte del usuario. Normalmente, la adaptabilidad es tratada en tiempo de diseño. Un sistema diseñado para ser adaptable también será más fácilmente adaptativo, ya que para permitir la adaptabilidad es necesario proporcionar un motor que puede ser usado de igual manera para producir adaptividad. En este sentido, la utilización de métodos de diseño adecuados, como por ejemplo el uso CSS en diseños Web facilita en gran medida las posibilidades de adaptación.

5.2 Multiplicidad de perfiles: es la capacidad del sistema de permitir al usuario asumir distintos perfiles en el sistema. Ello significa que el usuario puede pasar de un perfil a otro a lo largo del uso del sistema, o que distintos usuarios tendrán distintos perfiles en el sistema. El diseño para permitir la multiplicidad de perfiles puede ser afrontada en tiempo de diseño como en [Loz00][Cac03] o AB-

UIDE, donde cada usuario es relacionado con las tareas que puede realizar.

5.3 Preservación: preservación es la capacidad de una adaptación de preservar, tanto como sea posible, el estado de la interfaz de usuario antes de la adaptación. Este criterio permite evaluar cuán conservativa es una adaptación.

6 Representatividad: el criterio de representatividad describe la relación entre un término y/o símbolo y su referencia. Los códigos y nombres usados son significativos para los usuarios cuando tienen una fuerte relación semántica entre los códigos y los objetos o acciones que representan.

6.1 Consistencia del esquema de colores: la consistencia del esquema de colores está relacionada con el esquema de colores de los elementos de la interfaz de usuario. Es deseable usar el mismo esquema de colores para tareas similares.

6.1.1 Color de fondo: color de fondo de los elementos de la interfaz de usuario.

6.1.2 Color en primer plano: color en primer plano de los elementos de la interfaz de usuario.

6.2 Consistencia del esquema de texto: la consistencia del esquema de texto está relacionada con los distintos tipos de presentaciones usadas por los elementos de texto en una interfaz de usuario. Un número reducido de fuentes es deseable en la mayoría de los dominios de aplicación.

6.2.1 Tipos de letra: número de presentaciones distintas para el texto.

6.2.2 Abreviaturas: la consistencia en las abreviaciones necesita la utilización de las mismas reglas de abreviación para cualquier adaptación.

6.3 Consistencia del esquema de botones/enlaces: se refiere a la forma en que los botones/enlaces son presentadas en la interfaz de usuario. Los botones/enlaces deben ser presentados siempre de

la misma forma, especialmente para botones/enlaces que cumplan una misma tarea. De igual forma, este criterio está relacionado con los patrones de botones/enlaces recurrentes que habitualmente se usan en las interfaces de usuario, como por ejemplo, el habitual patrón "Aceptar", "Cancelar" y "Ayuda". La presentación del patrón tendrá que ser consistente en todas sus apariciones en la interfaz de usuario.

6.3.1 Patrón de botones/enlaces: describe la distribución para los patrones de botones/enlaces.

6.3.2 Esquema de colores de botones/enlaces: los esquemas de colores usados para presentar los botones/enlaces.

6.3.3 Esquema de texto de botones/enlaces: el esquema de texto usado para presentar los botones/enlaces (incluyendo la terminología – el mismo concepto debe ser representado con la misma palabra (*Quitar*, *Salir*, *Cerrar*, ...) y usar la misma abreviatura.

7 Orientación: Orientar al usuario tiene que ver con los medios disponibles para aconsejar, orientar, informar, instruir y guiar a los usuarios durante su interacción con el ordenador (mensajes, alarmas, etiquetas, etc).

7.1 Preentividad: un sistema es preentivo cuando no permite al usuario realizar las tareas en cualquier orden, sin seguir ninguna secuencia determinada. Sin embargo, un nivel bajo de preentividad puede producir en el usuario una sensación de "perdido en el espacio", donde no exista ninguna guía clara sobre cómo volver a casa (realizar las tareas).

Tradicionalmente los sistemas adaptativos, y especialmente los sistemas de autorización inteligentes, han estado enfocados a guiar al usuario para evitar precisamente la sensación de "perdido en el espacio", pero dicha tarea debe ser compaginada con el objetivo de permitir al usuario aprender a su manera. Por lo tanto, la preentividad suele ser necesaria dentro del diseño de interfaces de usuario adaptativas.

7.2 Insistencia: sólo porque cierta información esté disponible en la interfaz de usuario no significa necesariamente que el usuario será capaz de percibirla. Por ejemplo, un problema persistente en los sistemas basados en ventanas es la situación en la que el usuario se equivoca al elegir en qué ventana escribir un determinado texto. Todos los sistemas de ventanas proporcionan algún tipo de mecanismo para mostrar cuál es la ventana activa, a menudo marcando el título de ventana activa o sus bordes. Sin embargo, si el usuario está mirando al contenido de la ventana, estas medidas pueden ser insuficientes. Es un tema interesante dentro de las interfaces de usuario adaptativas hacer que la información importante no sólo esté disponible, sino que sea también percibida. Para asegurarse de que esto ocurre es necesario que las adaptaciones sean aplicadas en el momento apropiado. Existen muchas maneras posibles de proporcionar insistencia: (1) destacar los elementos visuales (agrandar ventanas, cambiar los colores de los *widgets*, ...), (2) interrumpir al usuario con diálogos (el típico, "¿está seguro de que desea salir de la aplicación?"), (3) señales auditivas, (4) dejar indicadores persistentes de eventos (por ejemplo, el icono que advierte en los clientes de correo de que hay correos pendientes de ser leídos). Para maximizar la eficiencia de la insistencia, es aconsejable mantener un modelo cognitivo que tenga en cuenta dónde se encuentra la atención del usuario en cada momento.

7.3 Reutilización de entradas y salidas: es la capacidad del sistema de permitir la utilización de entradas y salidas como futuras entradas. Las órdenes de "Copiar & Pegar" son ejemplos típicos de este tipo de reutilización de entradas y salidas. El comando *Doskey* del sistema operativo DOS es otro ejemplo de este tipo de comportamiento. El sistema puede sugerir posibles entradas en un campo de texto basándose en entradas anteriores (como ocurre en el navegador Mozilla o Firefox). El motor de adaptación debe proporcionar automáticamente la reutilización de las entradas cuando este criterio sea seleccionado. Es aconsejable también mantener "enlaces" a recursos usados con anterioridad, por ejemplo, de la forma en que la mayoría de los sistemas operativos hacen con la carpeta "*Mis documentos*", donde aparecen los documentos usados recientemente.

8 **Manejo de errores**: este criterio se refiere a los medios disponibles para evitar o reducir los errores y recuperarse de ellos cuando estos ocurren.

8.1 Tolerancia a anomalías: no importa lo bien que un sistema esté diseñado, los usuarios cometerán errores de los cuales querrán recuperarse. Los sistemas tolerantes a anomalías de uso pueden: (1) detectar los estados erróneos o "peligrosos", (2) evitar que se entre en un estado erróneo, (3) corregir descuidos y errores.

Cuando los usuarios pueden confiar en que el sistema les advertirá de aquellas acciones que sean "peligrosas", y recibirán ayuda para recuperarse de pequeños errores, se sentirán más libres de explorar la interfaz de usuario sin miedo. En este caso la noción de esfuerzo proporcional de Thimbleby [Thi90] es interesante aquí: si un error es fácil de cometer, entonces su efecto también debe ser fácil de recuperar. La tolerancia a las anomalías de uso en las interfaces de usuario adaptativas puede ser mejorada de distintas formas: (1) corregir entradas de texto mal escritas (la capacidad de autocorrección del procesador de textos Microsoft Word es un ejemplo de este tipo de comportamiento), (2) inferir el objetivo adecuado de los clics del ratón (seguramente un clic en medio de la nada estaba dirigido al *widget* más cercano), (3) el usuario puede ser avisado cuando se realicen tareas "peligrosas", (4) detectar situaciones "arriesgadas" (como por ejemplo cerrar una aplicación sin guardar documentos que hayan sido modificados) y advertir al usuario sobre el posible resultado indeseable.

La figura 4.12 describe un ejemplo de compromiso de usabilidad donde se han considerado todos los criterios descritos a lo largo de esta sección, así como las interrelaciones que surgen entre ellos. Sin embargo, debe ser el diseñador el que establezca un compromiso de usabilidad apropiado para su aplicación en las familias de plataformas elegidas. El compromiso no sólo puede ser especificado para cada plataforma, sino que puede ser especificado para los distintos entornos físicos o tipos de usuarios. De esta manera, si una plataforma sólo va a ser ejecutada en un solo tipo de plataforma, sería aconsejable definir un compromiso de usabilidad para cada uno de los tipos de usuarios que potencialmente usarán la aplicación.

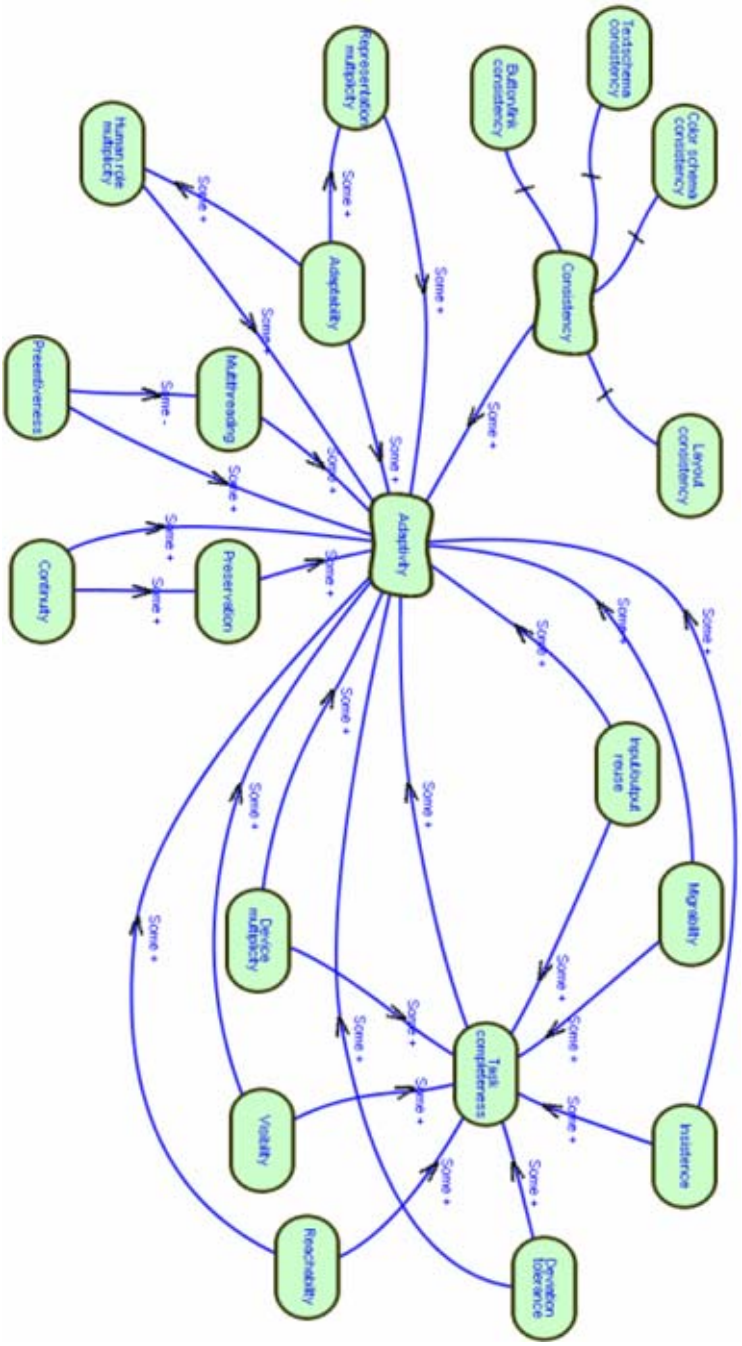


Figura 4.12 Ejemplo de compromiso de usabilidad donde se incluyen todos los criterios descritos y sus interrelaciones.

Evaluación de los criterios de usabilidad: métricas de usabilidad

Los criterios de usabilidad anteriormente descritos permiten la especificación del compromiso de usabilidad. Sin embargo, en tiempo de ejecución es necesario evaluar en qué grado se mantienen dichos criterios, de manera que se pueda decidir si se está cumpliendo el compromiso de usabilidad diseñado o no.

Para realizar la evaluación de los criterios es necesario proponer métricas que permitan conocer si un determinado criterio se da o no, y en qué medida. Para ello, las métricas de diseño propuestas en [Con99], pueden ser aplicadas para medir parte de los criterios. Por ejemplo, se puede especificar que para las aplicaciones de escritorio no se aplique ninguna adaptación que produzca un valor de visibilidad menor de 80. El método de evaluación de los criterios de usabilidad propuesto se encuentra descrito en la sección 5.2.7.1.

4.2.3 Fase de diseño

Durante la fase de diseño en AB-UIDE se crea una especificación de la interfaz de usuario suficientemente detallada para poder llegar a su implementación. Para ello se describen las tareas que el usuario podrá efectuar con el sistema y sus relaciones temporales de forma detallada, y qué objetos del dominio manipulan dichas tareas (objetos de interacción).

A partir de dicha descripción se crea una interfaz de usuario a un alto nivel de abstracción que será convertida en una representación concreta, que finalmente podrá ser visualizada a través de un visualizador (*renderer*).

4.2.3.1 Modelado de tareas

El modelo de tareas es el pilar principal de una aproximación basada en modelos. Dicho modelo proporciona una descripción de las tareas que el usuario podrá realizar a través de la interfaz de usuario, así como una especificación de las relaciones temporales que existen entre dichas tareas. Por ejemplo, dichas relaciones especifican si dos tareas pueden realizarse al mismo tiempo, o si por el contrario deben realizarse siguiendo una secuencia temporal predeterminada.

Dentro de AB-UIDE el modelo de tareas, a parte de describir las tareas que se pueden realizar a través de la interfaz de usuario y sus interrelaciones temporales, se describen también aquellas acciones que determinan un cambio de tarea, es decir, de forma paralela al modelo de

tareas se describe el modelo de diálogo de la aplicación. Para ello se conjugan dos notaciones: los diagramas de estado [Har87], y los ConcurTaskTrees [Pat99]. Finalmente, ambas notaciones son decoradas con las herramientas abstractas propuestas en [Con03].

El proceso de diseño del modelo de tareas es guiado lógicamente por el análisis de casos de uso realizado en la fase de requisitos. Cada caso de uso representa una tarea abstracta que el usuario debe llevar a cabo. Siguiendo con el pequeño ejemplo, anteriormente utilizado del usuario que desea registrarse en un sistema, se introduciría un nuevo estado que representa el caso de uso descrito en la figura 4.2. En esta ocasión, el caso de uso *Login* vendrá precedido del estado inicial, y seguido del estado final, ya que es el único (véase la figura 4.13).

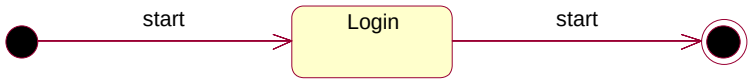


Figura 4.13 Modelo de tareas para el caso de uso *Login*.

Refinamiento del modelo de tareas

El modelado de los casos de usos y sus relaciones temporales produce un modelo de tareas demasiado genérico para servir más allá de la mera documentación del proyecto. Para poder generar de forma automática o semiautomática una interfaz de usuario es necesaria una especificación más detallada de las tareas que el usuario podrá realizar a través de la interfaz de usuario. Para ello, se realiza un proceso de refinamiento de las tareas identificadas inicialmente, describiendo detalladamente las acciones necesarias para llevar a cabo cada tarea, y sus interrelaciones (a través del uso de los operadores del lenguaje LOTOS [ISO88] propuestos en CTT).

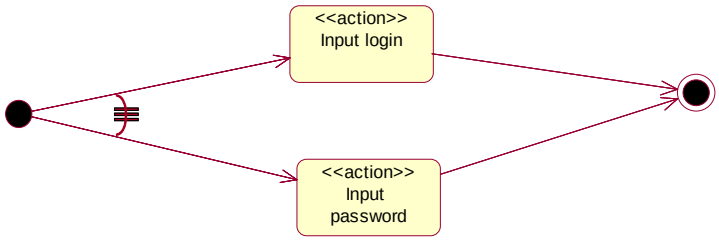


Figura 4.14 Refinamiento de la tarea *Login*.

La figura 4.14 ilustra el refinamiento de la tarea *Login* especificada en la figura 4.13. En ella, las tareas *Input login* e *Input password* estarán activas tan

pronto como se entre en la tarea de alto nivel (abstracta) *Login*, ya que ambas pueden ser ejecutadas de forma concurrente (especificado mediante el operador ||| de LOTOS). Ello significa que el usuario puede realizar cualquiera de las dos acciones (tareas de bajo nivel) en cualquier orden.

Propiedades de una tarea/acción

El modelado de las tareas, acciones y sus interrelaciones no es suficiente, sino que es también necesario describir en detalle cada una de las tareas/acciones, especificando sus propiedades más relevantes. Dentro de AB-UIDE las propiedades propuestas están reflejadas en la tabla 4.

Tabla 4. Propiedades de una tarea/acción en AB-UIDE.

Propiedad	Descripción
Nombre	Nombre suficientemente descriptivo de la tareas/acción.
Descripción	Descripción del propósito de la tareas/acción. Es usado principalmente como medio de documentación.
Tipo	Tipo de tarea (*abstract, input, output, operation, start, stop, select, create, delete, modify, move, duplicate, perform, toggle, view*).
Frecuencia	Frecuencia con que se espera se ejecute la tarea/acción en relación al resto (baja, media, alta).
Precondición	Condición que se debe dar para que la tarea/acción pueda ser ejecutada.
Poscondición	Condición que se debe dar tras la ejecución de la tareas/acción.
Presentación (sólo las tareas)	Tipo de presentación abstracta asociada a la tarea (*FreeContainer*, *Container*) (véase la sección 2.2.2.11).

Las tareas se describen mediante un nombre único, una descripción para la documentación del modelo, la frecuencia con que se espera se ejecutará la tarea/acción, y una precondición y una poscondición expresadas en OCL[21] (*Object Contraint Language*). OCL es un lenguaje desarrollado para expresar las restricciones existentes entre objetos, y habitualmente usado en conjunción con los diagramas UML.

Por lo tanto la definición de la tarea *Login*, y las acciones *Input login* e *Input password* se completaría proporcionando las propiedades descritas, tal y como se muestra en la figura 4.15.

[21] http://www.klasse.nl/ocl/

Task name	Login
Description	The user logs into the system providing a login and a password.
Type	Abstract
Frecuency	*High*
Precondition	NULL
Postcondition	*Participant.checkLogin()*
Presentation	FreeContainer

Action name	Input Login
Description	The user types in the login.
Type	*Input*
Frecuency	*High*
Precondition	NULL
Postcondition	*Participant.login!=“”*

Action name	Input Password
Description	The user types in the password for the login.
Type	*Input*
Frecuency	*High*
Precondition	*Participant.login!=“”*
Postcondition	*Participant.Password!=“”*

Figura 4.15 Propiedades de la tarea y las acciones asociadas al ejemplo del *Login*.

Especificación del diálogo con herramientas abstractas

Constantine [Con03] propone un conjunto de operaciones que pretenden ser un corpus de constructores para la especificación abstracta de las acciones básicas que lleva a cabo un usuario a través de la interfaz de usuario. La figura 4.16 muestra las herramientas abstractas propuestas por Constantine, y un ejemplo de cada una de las acciones asociadas.

SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	action/operation*	Print symbol table, Color selected shape
	start/go/to	Begin consistency check, Confirm purchase
	stop/end/complete	Finish inspection session, Interrupt test
	select	Group member picker, Object selector
	create	New customer, Blank slide
	delete, erase	Break connection line, Clear form
	modify	Change shipping address, Edit client details
	move	Put into address list, Move up/down
	duplicate	Copy address, Duplicate slide
	perform (& return)	Object formatting, Set print layout
	toggle	Bold on/off, Encrypted mode
	view	Show file details, Switch to summary

Figura 4.16 Herramientas abstractas canónicas [Con03].

El conjunto de herramientas abstractas permite especificar las acciones que producen un cambio de estado dentro de la interfaz de usuario a

través de la manipulación de los objetos de interacción, es decir, el diálogo que se establece entre la interfaz de usuario y el usuario. A las acciones abstractas propuestas por Constantine se han añadido "input" y "output", para representar acciones de entrada/salida a un nivel de abstracción menor, y "error" parar etiquetar aquellas transiciones que deben tomarse cuando se produzca una situación de error durante la comprobación de las precondiciones o poscondiciones. Si no se especifica una transición para un estado erróneo, se mostrará un mensaje de error por defecto y no podrá avanzar al siguiente estado que correspondería según el modelo de tareas.

Obsérvese que en la figura 4.13 la transición entre el estado inicial y la tarea de *Login*, que marca el comienzo de la interacción, está etiquetada con la acción "*start*", indicando que es un inicio de tarea. De igual manera, la transición desde la tarea *Login* al estado final está también etiquetado con la acción "*start*", lo cual implica que el usuario tendrá que ejecutar una acción de ese tipo para pasar al estado final, en este caso tendrá que lanzar el proceso de validación del usuario y contraseña introducidos dentro de la tarea *Login*.

La especificación del modelo de diálogo junto al modelo de tareas permite la derivación de una interfaz de usuario abstracta capaz de representar los componentes necesarios para realizar las tareas descritas.

Tareas y dominio: una relación obligada

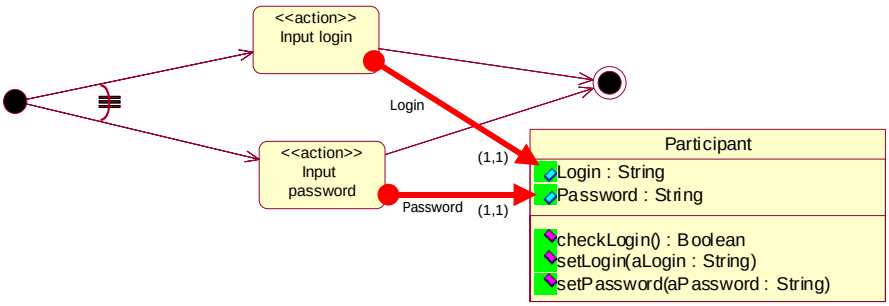
El modelo de tareas no es suficiente para derivar una interfaz de usuario abstracta, ya que no detalla qué tipo de datos se deben mostrar o leer, o cuál es la signatura de los métodos que deben invocar para realizar las funcionalidades demandadas a través de la interfaz de usuario.

Por lo tanto es necesario especificar qué objetos de interacción necesita usar cada una de las acciones para poder completarse.

Cada una de las relaciones entre una acción y los elementos del dominio con los que interactúa da lugar a un objeto de interacción. Los objetos de interacción representan la información que debe ser presentada o introducida en la interfaz de usuario.

La figura 4.17 muestra los objetos de interacción necesarios para realizar las dos acciones propuestas. Nótese que se especifican en forma de

relación entre una acción y un elemento del modelo del dominio. Las relaciones son unidireccionales y deben ser etiquetadas. De igual manera, se debe especificar una única cardinalidad (siguiendo el formato usado en los diagramas de clase UML) para determinar con cuántas instancias del elemento del modelo de dominio está relacionada la acción.



Name	Login	
Type	String	
	Initial value	<HISTORY_LOGIN>
	Password	false
	Max. Size	12

Name	Password	
Type	String	
	Initial value	""
	Password	true
	Max. Size	NULL

Figura 4.17 Objetos de interacción para el ejemplo del *Login*.

Las relaciones pueden tener atributos que describen propiedades que no pueden ser inferidas automáticamente a partir de la relación. En el ejemplo de la figura 4.17 es necesario especificar si es o no una tarea de entrada de contraseña, un tamaño máximo, y el valor por defecto inicialmente, ya que dichos atributos no pueden ser inferidos automáticamente a partir de la relación.

4.2.3.2 La interfaz de usuario abstracta

La interfaz de usuario abstracta permite describir la futura interfaz de usuario utilizando unos elementos de alto nivel. Idealmente, la descripción de la interfaz de usuario abstracta debe ser independiente tanto de la plataforma destino como de la modalidad de interacción elegida.

La interfaz de usuario abstracta es descrita utilizando objetos abstractos de interacción (AIO – *Abtract Interaction Objects*). Distintos conjuntos de objetos abstractos de interacción han sido propuestos en la literatura [Van93][Pin02][Loz00] para afrontar el diseño abstracto de una interfaz de usuario. Dentro de AB-UIDE se propone la utilización de los objetos abstractos de interacción propuestos para UML*i* en [Pin02] (véase la

sección 2.2.2.11). Sin embargo, los objetos de interacción propuestos en UML*i* han sido enriquecidos con nuevo tipo de objeto abstracto de interacción: *selector*. Los objetos de interacción de este tipo permiten especificar operaciones de selección sobre los elementos del modelo de dominio.

La figura 4.18 muestra la notación gráfica adoptada para la representación de los objetos abstractos de interacción en AB-UIDE.

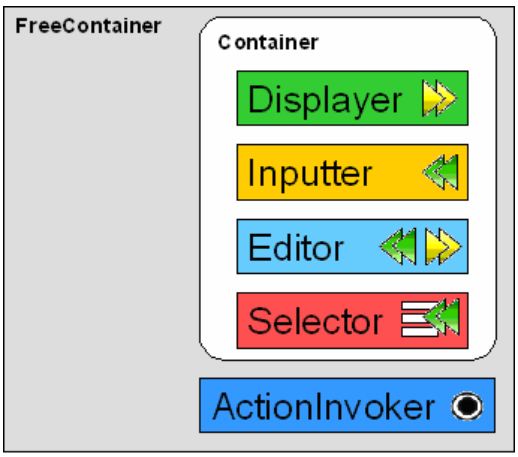


Figura 4.18 Notación gráfica de los objetos abstractos de interacción en AB-UIDE.

Todos los objetos abstractos de interacción son descritos especificando su nombre, tipo, tareas/acción a partir de la cual han sido derivados, y el tipo de acción a partir del cual han sido derivados. Los *ActionInvoker* además incluyen una precondición y una poscondición.

Derivación de la interfaz de usuario abstracta

La interfaz de usuario abstracta puede ser derivada automáticamente a partir de una descripción suficientemente detallada, tanto de las tareas de interacción que el usuario podrá hacer a través de la interfaz de usuario, como del modelo de dominio de la aplicación.

La estructura de contenedores

La estructura de contenedores de la interfaz abstracta refleja la separación de conceptos que el usuario debería ser capaz de discernir cuando se le presente la interfaz de usuario final. Una mala separación lógica de las tareas en la interfaz hará que el usuario tenga mayores problemas a la hora de manejar la aplicación, y tendrá un efecto negativo en el aprendizaje del

manejo de la interfaz de usuario (todo ello está relacionado con la componente facilidad de aprendizaje (*learnability*) de la usabilidad de un sistema) debido a su falta de estructura.

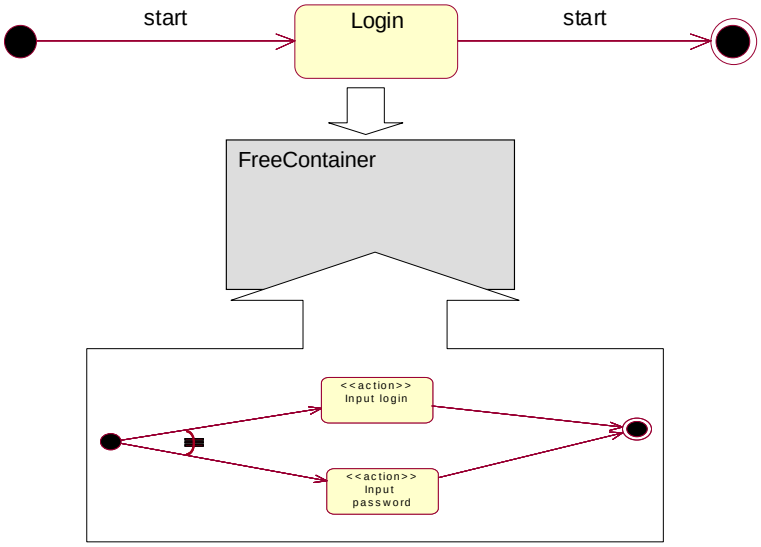


Figura 4.19 Estructura de contenedores para la tarea *Login*.

Cada tarea de alto nivel está asociada a un contenedor. Dicho contenedor puede ser un *FreeContainer* o un *Container*, dependiendo de si es un contenedor raíz o no. La elección de uno u otro es tomada por el diseñador durante el modelado de las tareas de interacción (propiedad "presentación"). Nótese que cuando la interfaz de usuario abstracta sea transformada en una interfaz de usuario concreta, el diseñador podrá decidir si se mantiene la estructura de contenedores creada, o si por el contrario, por ejemplo, todo se agrupa en un solo contenedor raíz. Habitualmente esta última solución no será apropiada, excepto para pantallas de gran tamaño en entornos donde la visibilidad simultánea de todas las tareas sea importante (por ejemplo, en un sistema de vigilancia hospitalaria de enfermos donde es importante poder monitorizar un enfermo sin perder de vista al resto de pacientes).

La figura 4.19 describe la estructura de contenedores derivada a partir del modelo de tareas descrito en la sección 4.2.3.1. La tarea *Login* será mostrada en un *FreeContainer* tal y como se especificó en la descripción de las propiedades de la tarea. Los objetos abstractos de interacción derivados del refinamiento de la especificación de la tarea estarán incluidos en el *FreeContainer* de la tarea *Login*.

Los objetos abstractos de interacción de las acciones

Los objetos de interacción asociados a las acciones son derivados a partir del tipo de acción especificado durante la creación del modelo de tareas. La tabla 5 muestra la correspondencia entre el tipo de acción especificada en el modelo de tareas y el objeto abstracto de interacción seleccionado para representarlo.

Tabla 5. Propiedades de una tarea/acción en AB-UIDE.

Tipo de acción	Objeto abstracto de interacción seleccionado
input, create, delete, move, duplicate	*Inputter*
output, view	*Displayer*
modify	*Editor*
action, perform	*Control*
select, toggle	*Selector*

La figura 4.20 muestra los objetos de interacción derivados a partir de las acciones especificadas para la tarea *Login*.

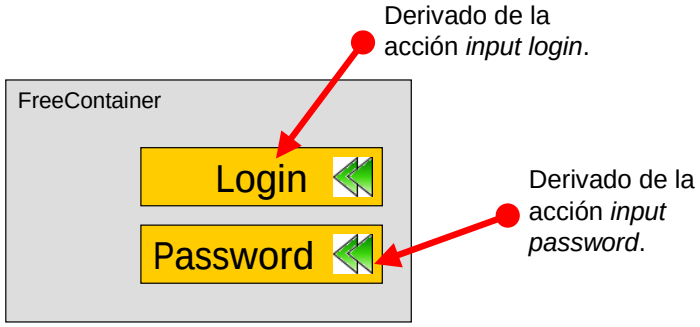


Figura 4.20 Elementos abstractos de interacción

Transiciones entre estados: diálogo entre el usuario y el sistema

Como se vio en la sección 4.2.3.1, el modelo de tareas está decorado mediante el etiquetado de las transiciones entre las tareas con herramientas abstractas [Con03]. Ello permite la derivación automática de los elementos necesarios para producir la transición entre los distintos estados de la interfaz de usuario.

Las herramientas abstractas aplicables a la transición entre las distintas tareas abstractas de la aplicación son: *start* y *stop*, donde ambas pueden

combinarse en una misma transición. *Start* significa que se deben derivar los elementos abstractos necesarios para navegar de una tarea abstracta a otra, mientras que *stop* significa que para seguir esa transición se debe realizar una operación de cancelación, y por lo tanto será necesario derivar los objetos abstractos de interacción en la tarea abstracta anterior para permitir realizar la operación de cancelación, y volver el foco de la interacción a la tarea abstracta anterior. Una transición puede estar etiquetada con ambas herramientas abstractas a la vez, para denotar que esa transición puede tomarse, tanto para navegar hacia una nueva tarea abstracta como para volver a la anterior.

En la figura 4.21 se muestran los elementos abstractos de interacción derivados a partir de la transición *start* para el ejemplo del *Login*. Obsérvese que se han añadido dos objetos de interacción abstracta. Por una parte se ha añadido un *ActionInvoker* para permitir la navegación a la siguiente tarea abstracta, y por otra parte se ha añadido un *ActionInvoker* que permite invocar la ayuda. Esta combinación puede ser configurada por el usuario, aunque debería ser consistente a lo largo de toda la interfaz. En esta ocasión no se ha añadido ningún *ActionInvoker* que permita cancelar la tarea abstracta, ya que no existe ninguna transición de tipo *stop*. Ello implica que la tarea abstracta *Login* es obligatoria para poder continuar con el uso de la interfaz de usuario creada.

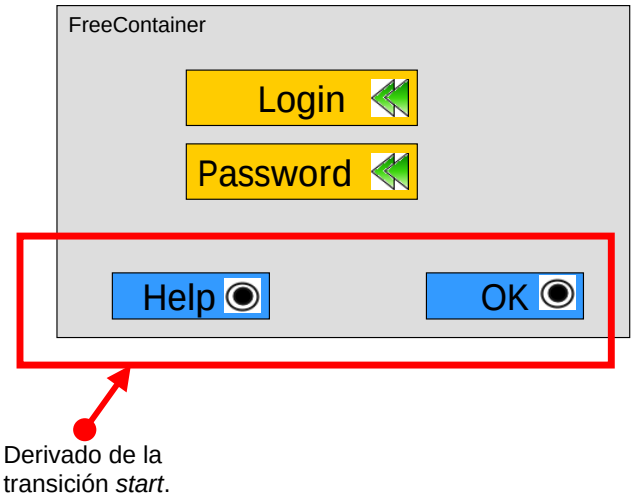


Figura 4.21 Elementos abstractos de interacción derivados de la transición *start* para el ejemplo del *Login*.

En el ejemplo de la figura 4.21, el *ActionInvoker* "OK" ha sido derivado a partir de la transición *start* para pasar de la tarea abstracta *Login* a la siguiente tarea, en este caso el final de la aplicación. Por lo tanto, para que dicha transición pueda realizarse debe darse la poscondición expresada para la tarea *Login*. Ello significa que la precondición del *ActionInvoker* "OK" tendrá que ser la poscondición asociada a la tarea *Login*. Adicionalmente, será necesario añadir como poscondición al *ActionInvoker* las sentencias de OCL necesarias para realizar la transición a la siguiente tarea.

En el caso en que tengamos acciones del tipo *action* o *perform*, la acción estará asociada a un método del dominio, cuya signatura estará disponible en el modelo de dominio. En el caso en que el método invocado tenga parámetros será necesario especificar los argumentos apropiados de entre los elementos del modelo de dominio y los elementos de la interfaz de usuario, usando OCL.

4.2.3.3 La interfaz de usuario concreta

La interfaz de usuario concreta representa la interfaz en un nivel de abstracción bastante cercano al código final, y debe ser capaz de especificar con detalle la presentación que el usuario verá finalmente.

La transformación de la interfaz de usuario abstracta a concreta puede realizarse de distintas maneras, tanto de forma automática como semiautomática. Aunque inicialmente se propusieron distintas aproximaciones que buscaban la transformación automática, como por ejemplo los árboles de selección [Van99b] [Bod94] o las reglas de selección [Van97][Jan93], la baja usabilidad de las interfaces de usuario generadas para aplicaciones medianamente complejas ha hecho que la mayoría de las aproximaciones propongan un proceso de traducción semiautomático, como es el caso de AB-UIDE.

El proceso de transformación debe ir orientado a la maximización de la usabilidad del sistema. Sin embargo, aunque existen criterios que ayudan a guiar el proceso de transformación, como guías de estilo, criterios ergonómicos o patrones son de difícil aplicación directa [Mon05b] por la falta de una estructuración clara, aunque sí sirvan de ayuda.

Dentro de AB-UIDE se ha elegido el modelo concreto de interfaz de usuario propuesta para usiXML, ya que recoge de una manera abstracta,

pero a su vez suficientemente concreta, todos los *widgets* disponibles en la mayoría de las plataformas destino. De igual manera, este modelo permite especificar el posicionamiento de los elementos dentro de los contenedores utilizando un modelo de cajas como el usado en el lenguaje XUL[22] o la distribución de componentes *BoxLayout* del lenguaje *Java*[23](véase la sección 5.3.1). La conversión a dicho modelo se realiza mediante la aplicación de reglas de transformación.

Selección de contenedores concretos

Los contenedores abstractos raíz (*FreeContainer*) son transformados en *windows* en usiXML, que representan la raíz de la especificación de una ventana en usiXML. Sin embargo, existirán distintas formas de traducir los *Containers*. Por defecto cada *Container* se transformará en una caja (*box*) de la ventana de la tarea abstracta de la cual dependen. Sin embargo, otras disposiciones son también posibles. Por ejemplo, cada subtarea de una tarea abstracta padre puede ser presentada en forma de una pestaña (*tab*), las cuales estarían contenidas dentro de un conjunto de pestañas (*tabbedDialogBox*) para producir una interfaz de usuario más compacta.

Debe destacarse que el posicionamiento de los elementos dentro de los contenedores define en gran medida el aspecto visual del sistema. La aplicación de reglas que produzcan resultados aceptables es posible. Sin embargo, hoy en día se hace imposible competir con la distribución manual de los componentes por parte del diseñador de la interfaz. Por ello, dentro de AB-UIDE la selección del modo en que los elementos de la interfaz se distribuyen dentro de los contenedores es refinada manualmente, para mejorar la usabilidad de la aplicación final creada, y a partir del cual partirá el proceso de adaptación.

Selección de *widgets*

Es necesario tener en cuenta que la selección de *widgets* para representar cada objeto de interacción abstracto no consiste en una correspondencia uno a uno. En general un mismo objeto abstracto con las mismas propiedades puede ser representado utilizando distintas combinaciones de objetos de interacción concretos. Es por lo tanto posible la transformación de una interfaz de usuario abstracta en distintas presentaciones concretas.

[22] http://www.xulplanet.com
[23] http://java.sun.com

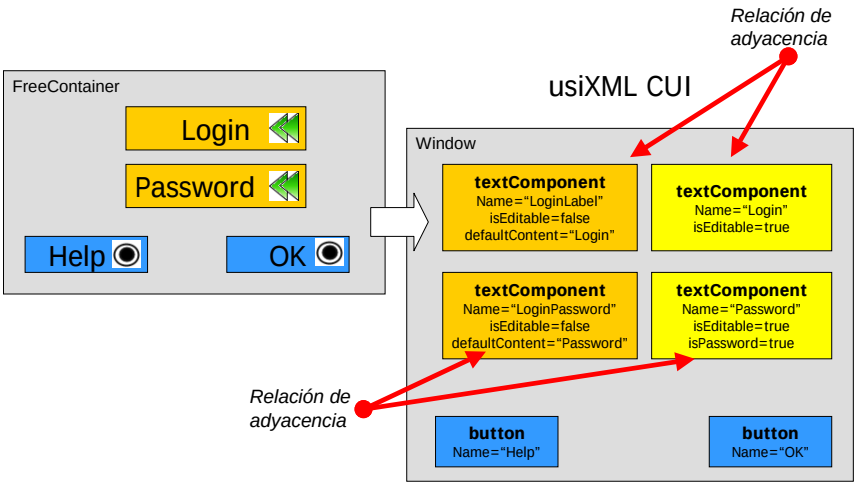


Figura 4.22 Transformación de interfaz de usuario abstracta a concreta para el ejemplo del *Login*.

Siguiendo este razonamiento, aunque dentro del método de proponen reglas para la traducción de las combinaciones de objetos abstractos más comúnmente utilizadas se deja el camino abierto al diseñador para que añada sus propias reglas. Puesto que el modelo abstracto se almacena en un formato basado en XML (usiXML), el diseñador puede crear nuevas reglas de transformación usando cualquiera de los procedimientos habituales aplicables a la transformación de una especificación XML: transformaciones XSLT[24], transformaciones de gramáticas de grafos [Lim04], o transformación de especificaciones algebraicas [Bor05].

Como se puede observar en la figura 4.22 una de las reglas existentes determina, que para cada objeto abstracto de interacción de tipo *inputter*, que represente una tarea de tipo *input,* se generará una caja de texto (*textComponent*) cuyo contenido pueda ser modificado (*isEditable* = *true*) donde el usuario podrá introducir sus datos, y además se generará una etiqueta de texto (*textComponent*) no modificable (*isEditable* = *false*) que describa qué es lo que el usuario debe introducir en la caja de texto. Para reflejar que ambos componentes están relacionados y deberían aparecen juntos en la interfaz de usuario, se generará también una relación de adyacencia entre ambos elementos. Las relaciones de adyacencia indican que cuando sean presentados al usuario, el visualizador (*renderer*) tendrá

[24] http://www.w3.org/TR/xslt

que intentar que ambos componentes aparezcan juntos, para evitar la pérdida de semántica en la interfaz de usuario.

El comportamiento de la interfaz

El dinamismo en la interfaz es introducido a través de la transición entre las distintas ventanas identificadas, la invocación de métodos de los objetos del dominio y el acceso a los atributos de los objetos del dominio.

Los *actionInvokers* serán normalmente representados en forma de botones. Dichos botones tendrán asociados dos tipos de comportamientos. Por una parte deben reflejar la navegación entre las distintas tareas abstractas identificadas, y por otra parte deben desencadenar la ejecución de métodos del dominio.

Para la inclusión del comportamiento asociado a los *ActionInvoker* en la interfaz de usuario concreta, se ha extendido el modelo de interfaz concreta de usiXML, añadiendo a todos los objetos concretos de interacción la posibilidad de tener asociados eventos, los cuales a su vez pueden tener asociados o no una precondición y una poscondición.

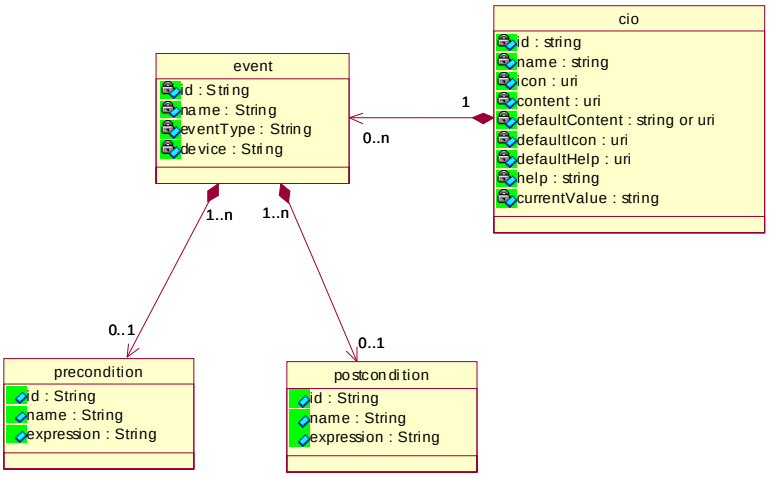


Figura 4.23 Modelo de comportamiento propuesto en AB-UIDE.

Tal y como puede apreciarse en la figura 4.23 el modelo de comportamiento propuesto se especifica utilizando los siguiente elementos:

- *Event*: expresa un evento asociado al objeto de interacción concreto. Viene expresado mediante el nombre del evento, el tipo de evento (los tipos de eventos permitidos son los habituales para cada tipo de dispositivo, por ejemplo, para un ratón: *onClick, onDoubleClick, onMouseUp, onMouseDown, onMouseMove, onMouseDrag, onMouseDrop, onEnter, onExit*). El campo *device* indica el dispositivo al cual está asociado el evento, por ejemplo, *mouse*.

- *Precondition*: almacena la precondición expresada en OCL del objeto concreto de interacción.

- *Postcondition*: almacena la poscondición expresada en OCL del objeto concreto de interacción.

Tanto el campo *precondition* como el campo *postcondition* son directamente obtenidos de los campos correspondientes de los objetos abstractos de interacción a partir de los cuales han sido derivados. Por defecto las precondiciones y poscondiciones son asociadas al evento *onClick* del ratón, y al evento *onKeyboardEnterPressed* del teclado para los *widgets*. De forma que se permita interaccionar con la aplicación tanto con el teclado como con el ratón. La precondición será ejecutada cada vez que el evento sea activado, mientras que la poscondición se producirá como respuesta al evento. En las ventanas las precondiciones están asociadas por defecto al evento *OnLoad*, que ocurre cuando se carga una ventana, y las poscondiciones al evento *OnBlur*, que sucede cuando se cierra una ventana. Durante la generación del código por parte del *renderer* se transforman los eventos especificados en eventos válidos para el lenguaje destino.

Aspecto visual de los objetos de interacción concretos

A lo largo de la etapa de diseño no se ha especificado cómo se describe el aspecto visual de los objetos concretos de interacción. Dentro de AB-UIDE se propone la aplicación de hojas de estilo en cascada (*CSS*[25] – *Cascade Style Sheets*) que aplicadas mediante transformaciones XSLT, permitan definir el aspecto de los componentes, y modificarlo de forma estructurada y sencilla. La principal ventaja de la utilización de CSS para describir el aspecto de la interfaz es que es ampliamente conocido por los diseñadores de interfaces de usuario actualmente, y además el aspecto de

[25] http://www.w3.org/Style/CSS/

los elementos puede ser diseñado visualmente para un buen conjunto de plataformas utilizando potentes herramientas visuales de programación.

4.2.3.4 Trazabilidad en tiempo de ejecución

AB-UIDE es un método cuyo objetivo es el diseño de interfaces de usuario adaptativas. Ello supone que el sistema debe ser capaz de adaptar la interfaz de usuario en tiempo de ejecución a los cambios en el contexto de uso detectados por el sistema. Una vez detectado un cambio, el sistema debe evaluar las posibilidades de adaptación posibles de acuerdo a los datos que posee sobre el contexto de uso, pero también sobre la interfaz de usuario en sí. Los datos que el sistema posee de la interfaz de usuario con los recopilados durante el diseño de la misma, los cuales serán actualizados debido, tanto a la interacción con el usuario, como al proceso de adaptación.

Sin embargo, la interfaz de usuario que es renderizada para ser presentada al usuario no es suficiente para la toma de decisiones, ya que es importante conocer explícitamente las relaciones de los distintos componentes de la interfaz de usuario concreta, tanto con el modelo de tareas como con el modelo de dominio. Por ejemplo, si se reduce el área de presentación en pantalla, y la interfaz intenta acomodarse a esta nueva situación será necesario conocer qué tipos de datos se está manejando. Así si en la aplicación un valor booleano se estaba presentado en forma de dos botones de radio (*radioButton*), podría ser presentado usando una casilla de verificación para ahorra espacio (véase la figura 4.24), pero para ello es necesario saber que es un valor booleano lo que se está presentando, y que por lo tanto sólo se necesita un control capaz de alternar entre dos valores.

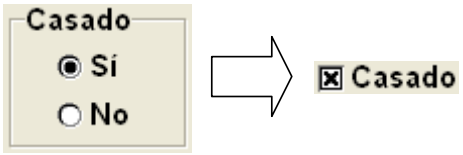


Figura 4.24 Ejemplo de adaptación para la presentación de un valor booleano.

Por lo tanto, se hace necesario para el motor de adaptación tener disponible en tiempo de ejecución las relaciones existentes entre los modelos de la interfaz de usuario, que han dado lugar finalmente a la interfaz de usuario concreta que se muestra al usuario. Las relaciones existentes entre los modelos describen la estructura interna y relaciones

entre sus componentes, es decir, definen la arquitectura interna de la interfaz de usuario. Ello ha motivado que dentro de AB-UIDE se proponga la utilización de los conectores [Lop03a][Lop03b] utilizados en la descripción de arquitecturas software como método para representar las relaciones entre los distintos modelos de la interfaz de usuario.

Un conector [All97] es un conjunto de roles y una especificación de un pegamento que los mantiene unidos. Los roles modelan el comportamiento de cada parte involucrada en la interacción, mientras que el pegamento proporciona la coordinación entre las instancias de cada rol [Wer00].

Los conectores fueron originalmente propuestos dentro del mundo de las arquitecturas software para proporcionar un mecanismo para la interconexión de distintos componentes software, y para permitir la reconfiguración dinámica de las arquitecturas software. Para usar los conectores en el proceso de construcción de un sistema concreto, los roles serán instanciados. Sin embargo, un componente no será capaz de instanciar un rol si no cumple con las especificaciones del servicio que ese rol debe desempeñar.

Un conector es especificado describiendo:

- Las variables de entrada que serán usadas como puerto de entrada.
- Las variables de salida que serán usadas como puertos de salida.
- El conjunto de acciones, que serán disparadas de acuerdo a una condición de guarda.

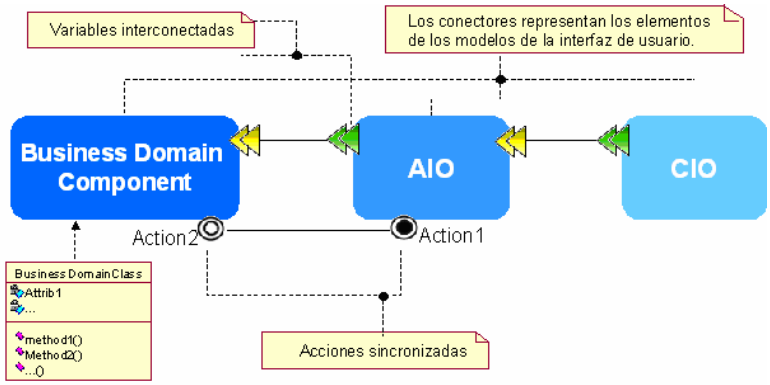


Figura 4.25 Esquema general del funcionamiento de los conectores.

La comunicación entre los componentes se realiza de dos formas distintas. Por un lado, las variables de entrada y salida de los distintos conectores están interconectadas, por lo que una actualización en una variable de un puerto de salida producirá automáticamente una transmisión de dicha actualización a través de su puerto de salida (si está conectado), y por otro lado los métodos de distintos conectores pueden ser sincronizados. Esto significa que cuando se ejecuta un método en un conector se disparará automáticamente el método con el que estén sincronizados, por ejemplo, en la figura 4.25 , la ejecución de *Action1* llevaría consigo la ejecución de *Action2*.

La utilización del concepto de conector dentro de AB-UIDE persigue dos objetivos. Por una parte permite mantener las relaciones entre los modelos en tiempo de ejecución para facilitar la toma de decisiones durante el proceso de adaptación, y por otra parte permite que el diseñador sea capaz de visualizar y modificar visualmente las relaciones existentes entre los distintos modelos de la interfaz de usuario, facilitando al diseñador información adicional sobre los aspectos de trazabilidad del sistema.

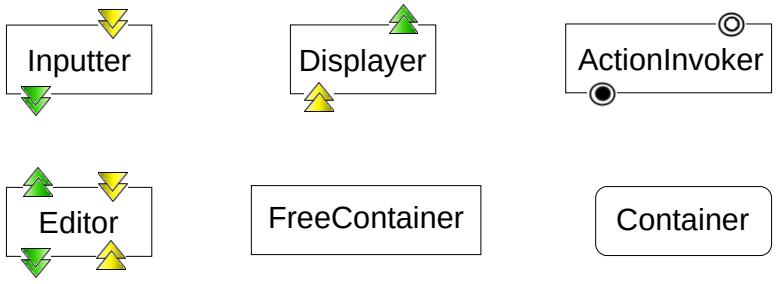


Figura 4.26 Representación gráfica para los conectores de los OAI.

En la figura 4.26 se muestra la notación gráfica adoptada para los conectores que representan los objetos abstractos de interacción (OAI).

La notación gráfica para los componentes concretos de interacción está formada por un rectángulo con el tipo y nombre del objeto (aunque el resto de propiedades estén también disponibles), y los puertos de entrada, salida, y sincronización de métodos apropiados. De está manera, todos aquellos objetos concretos de interacción que tengan capacidades de entrada (como por ejemplo, un campo de texto, o una lista desplegable) tendrán un puerto de salida para comunicar dicho valor al componente abstracto a partir del cual han sido derivados. Los componentes con capacidades de salida que muestran algún elemento del modelo de

dominio tendrán un puerto de entrada para recibir el valor a mostrar. Los componentes que permitan tanto entrada como salida, podrán tener un puerto de salida y uno de entrada. Los objetos concretos de interacción que tengan capacidades de invocación de métodos del modelo de dominio tendrán tanto puertos para la sincronización de métodos como métodos pueden invocar (ya que por ejemplo un clic del ratón puede desatar la ejecución de varios métodos). Nótese que dentro del diagrama se representan también las relaciones gráficas existentes entre los objetos concretos de interacción, para facilitarle al diseñador una visión más clara de la estructura actual de la interfaz de usuario.

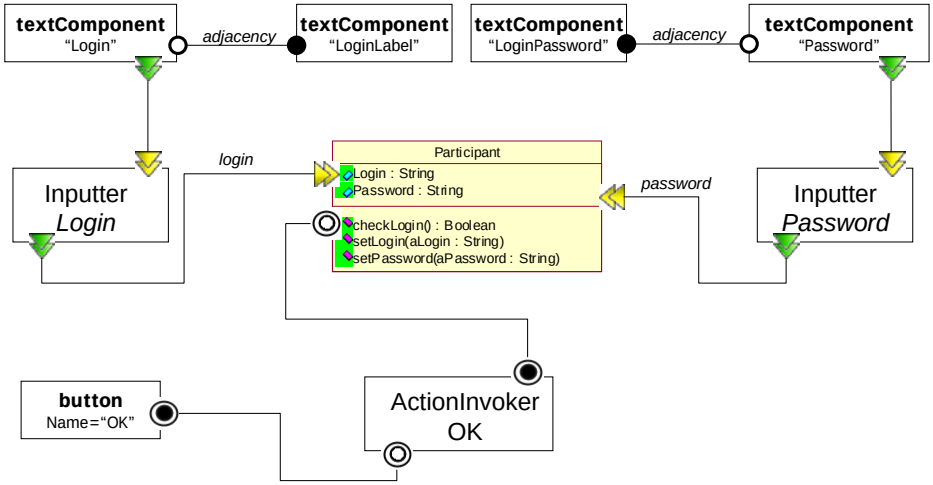


Figura 4.27 Diagrama de conectores para el ejemplo de la tarea *Login*.

En la figura 4.27 se puede observar la estructura de conectores correspondiente al ejemplo de la tarea de *Login,* usada hasta ahora como ejemplo. En dicha figura se observa que los *textComponent* no modificables (etiquetas) *LoginLabel* y *LoginPassword* no contienen ningún puerto, ya que son objetos que muestran una etiqueta fija, no tomada del modelo de dominio directamente. Los *textComponent* modificables (campos de texto) contiene un puerto para enviar el valor que sea introducido por el usuario. La toma de datos por parte de estos objetos no es representada en el diagrama, ya que pertenece a la interfaz de usuario final, es decir, al código generado de la interfaz. Los objetos abstractos de interacción de tipo *inputter* tiene dos puertos. Un puerto de entrada para recibir el valor del campo de texto asociado, y otro de salida para actualizar el modelo de dominio con el nuevo valor. Finalmente, el botón *OK* tiene una sincronización con el *ActionInvoker OK*, que a su vez provoca la ejecución

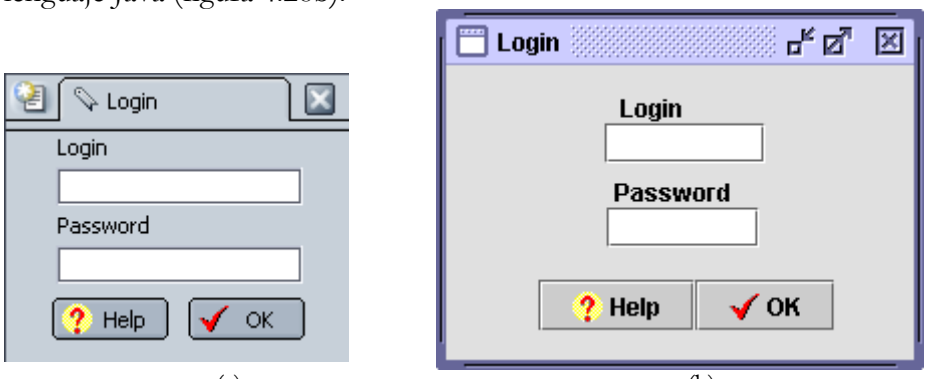
del método *checkLogin* del dominio. Debe destacarse, que el diseñador puede modificar cómo se produce la ejecución del método en el botón, es decir, puede especificar el evento que lo produce, y las precondiciones y poscondiciones asociadas (véase la sección 4.2.3.3).

El diagrama de conectores permite modificar visualmente las relaciones existentes entre los distintos modelos de la interfaz de usuario, facilitando en gran medida la compresión que el diseñador mantiene de la estructura de la interfaz de usuario en cada momento.

4.2.4 Fase de implementación

Una vez terminada la etapa de diseño ya se ha creado un modelo completo de la interfaz de usuario que debe ser interpretado o compilado en código ejecutable. Dicho modelo puede ser compilado para distintos lenguajes y plataformas a través de un visualizador (*renderer)* que sea capaz de transformar, tanto el aspecto visual de la interfaz como el comportamiento dinámico de la misma, en código ejecutable o interpretable

Actualmente se han desarrollado para AB-UIDE dos *renderers* que permiten generar las presentaciones finales con las que el usuario interactuará. Por una parte se ha desarrollado uno para el lenguaje XUL [Oes02], y otra parte se ha desarrollado un segundo *renderer* para el lenguaje Java[26]. En la figura 4.28 se muestra la interfaz de usuario creada para la tarea *Login* tanto para el lenguaje XUL (figura 4.28a), como para el lenguaje Java (figura 4.28b).



(a) (b)

Figura 4.28 Interfaz de usuario para el ejemplo del *Login* renderizado en XUL y en Java.

[26] http://java.sun.com

La generación de la interfaz de usuario final, es sólo un paso en la ejecución de una aplicación adaptativa creada usando AB-UIDE. La ejecución de una aplicación adaptativa en AB-UIDE conlleva la integración de los modelos y la interfaz de usuario creada dentro del motor de adaptación propuesto, implementado en forma de sistema multi-agente, y que permite la aplicación del proceso de adaptación de una interfaz de usuario de una forma eficaz. El motor de adaptación propuesto se halla descrito con detalle en el capítulo 5.

4.2.5 Usabilidad en AB-UIDE

A lo largo del proceso de diseño de una interfaz de usuario en AB-UIDE se persigue maximizar la calidad del producto final. Para ello, se han integrado técnicas intentando garantizar que el resultado final tras la aplicación del método será una interfaz de usuario de una alta calidad, es por lo tanto más un método centrado en el uso [Con99]. Por ello se ha propuesto un método basado en la transformación de modelos, y guiado por las tareas que el usuario futuro desea realizar con el sistema, proponiendo un método robusto para la generación final de una interfaz de usuario de calidad.

Sin embargo, no se ha renunciado a los beneficios que las aproximaciones centradas en el usuario proporcionan al desarrollo de la interfaz de usuario, como pueden ser: el modelado del usuario, y su entorno de interacción, o un diseño iterativo basado en la evaluación mediante técnicas empíricas de la interfaz de usuario.

Por todo ello, AB-UIDE se presenta como un método de diseño de interfaces de usuario centrado en el uso, pero que complementa su proceso con algunas de las técnicas más prominentes del diseño de interfaces de usuario centrado en el usuario.

4.3 Conclusiones del capítulo

A lo largo de este capítulo se ha propuesto un método para el diseño de interfaces de usuario, incluyendo el modelado de las capacidades de adaptación de la interfaz de usuario. El método sigue una aproximación dirigida por modelos (MDA) que cubre todo el ciclo de desarrollo de una interfaz de usuario.

El método es guiado por la elicitación de las tareas que el usuario futuro podrá realizar con el sistema, las cuales son refinadas hasta conseguir una interfaz de usuario asociada a cada una de ellas.

AB-UIDE permite también mantener la trazabilidad a lo largo del proceso de desarrollo de la interfaz, y aporta una notación gráfica para su visualización y edición de una forma clara e intuitiva para el diseñador a través del concepto de conector. El modelo de conectores propuesto permite adicionalmente disponer de las relaciones establecidas entre los modelos durante la fase de diseño en tiempo de ejecución, de forma que puedan ser empleadas para realizar un proceso de adaptación más exacto.

CAPÍTULO 5
ARQUITECTURA BASADA EN UN SISTEMA MULTI-AGENTE PARA LA EJECUCIÓN DE INTERFACES DE USUARIO ADAPTATIVAS

"Nadie podrá decir que un nido calentito y dichoso dará de sí muy grandes personas. La inadaptación a lo imperfecto es lo que mejora al hombre."
(Antonio Gala)

5.1 Introducción

El requisito principal de esta arquitectura será la ejecución de las fases necesarias para llevar a cabo el proceso de adaptación (véase la sección 2.1.1) de forma eficaz y eficiente. Por lo tanto, la arquitectura propuesta debe ser capaz de permitir la detección de los cambios en el contexto de uso de la aplicación (usuario, plataforma, entorno físico, y tarea actual del usuario), reaccionar ante éstos proponiendo distintas alternativas de adaptación, seleccionar las mejores adaptaciones a aplicar, y finalmente, aplicarlas.

Para el diseño e implementación de la arquitectura propuesta se ha optado por la utilización de un sistema multi-agente. Uno de los requisitos principales de la arquitectura es la necesidad de tomar decisiones sobre qué alternativa de adaptación es más apropiada. En este sentido, una arquitectura basada en el concepto de agente beneficia en gran medida al sistema, ya que permite un modelado más natural del proceso humano de toma de decisiones [Bra87]. Los sistemas multi-agente además fomentan la distribución de las responsabilidades entre distintos agentes, facilitando la actual tendencia de descentralización del software.

La arquitectura propuesta es capaz de aprovechar el conocimiento de la interfaz de usuario recopilado durante el diseño de la interfaz, permitiendo la integración del diseño de la interfaz de usuario basado en modelos dentro del proceso de adaptación.

5.2 Arquitectura del sistema multi-agente

A continuación, se presenta una visión general del diseño de la arquitectura multi-agente propuesta. La notación escogida para ello es la propuesta dentro de la metodología de desarrollo de sistemas multi-agente *Prometheus* [Pad02] (véase la sección 2.3.3.3).

5.2.1 Objetivos del sistema multi-agente

Los objetivos o metas principales de la arquitectura propuesta incluyen principalmente la ejecución del proceso de adaptación propuesto. Por lo tanto, el diseño del sistema multi-agente propuesto ha partido de la especificación de las distintas etapas del proceso de adaptación como metas, las cuales han sido refinadas para conseguir el conjunto de objetivos finales que debe perseguir el sistema multi-agente.

La figura 5.1 muestra los objetivos especificados para el sistema multi-agente. El objetivo final es la adaptación de la interfaz de usuario *(Adapt user interface)*. Dicho objetivo ha sido descompuesto en cuatro objetivos que representan las cuatro etapas del proceso de adaptación. La etapa de iniciativa es modelada mediante el objetivo *Init adaptation*. Esta etapa se subdivide a su vez en la detección de la posible necesidad de la aplicación de una adaptación *(Detect need for adaptation)*, y la detección de los cambios en el contexto *(Sense context of use)*. La detección de los cambios en el contexto incluye la detección de los cambios tanto en la plataforma donde se realiza la interacción *(Sense platform context)*, como en el usuario actual *(Sense user context)* y el entorno físico donde el usuario se encuentra *(Sense environment context)*. Por otra parte dicho objetivo también incluye la inferencia de cuál puede ser la tarea actual que el usuario está llevando a cabo en este momento *(Detect current user task)*, así como el procesado de los cambios en el contexto simulados *(Simulate context of use changes)* para permitir la evaluación y depuración de las capacidades de adaptación (especialmente cuando no se dispone de los sensores físicos necesarios para captar determinados cambios en el contexto).

Figura 5.1 Objetivos del sistema multi-agente propuesto.

Las metas también incluyen la proposición de las adaptaciones factibles dado el contexto de uso actual *(Propose adaptations)*, la selección de las adaptaciones *(Select adaptations)*, y la ejecución finalmente de las adaptaciones seleccionadas *(Execute adaptations)*. La selección de adaptations incluye la evaluación de cada una de las adaptaciones propuestas, ejecutándolas virtualmente. La ejecución de las adaptaciones se lleva a cabo mediante la aplicación de las reglas seleccionadas *(Run adaptations)*, la evaluación del compromiso de usabilidad que garantice la preservación de la usabilidad del sistema *(Evaluate usability trade-off)*, y finalmente la visualización de la interfaz de usuario adaptada *(Render user interface)*.

5.2.2 Papeles en el sistema multi-agente

Los papeles *(roles)* permiten agrupar los objetivos del sistema de acuerdo a las distintas personalidades que debería asumir un agente en el sistema para llevar a cabo una serie de metas. En la figura 5.2 se muestran los papeles identificados para la arquitectura multi-agente propuesta. Se ha identificado un papel para cada una de las etapas del proceso de

adaptación *(Start adaptation process*, *Propose feasible adaptations*, *Select best adaptations*, *Adaptations execution)*. Adicionalmente se ha identificado un papel distinto para cada una de los objetivos del sistema encargados de tratar con la detección y generación de los eventos del contexto *(Sense platform context changes*, *Sense environment changes*, *Sense user changes* y *Simulate context of use changes)*. Finalmente, existe un papel encargado de mostrar la interfaz adaptada al usuario *(Visualize user interface)*.

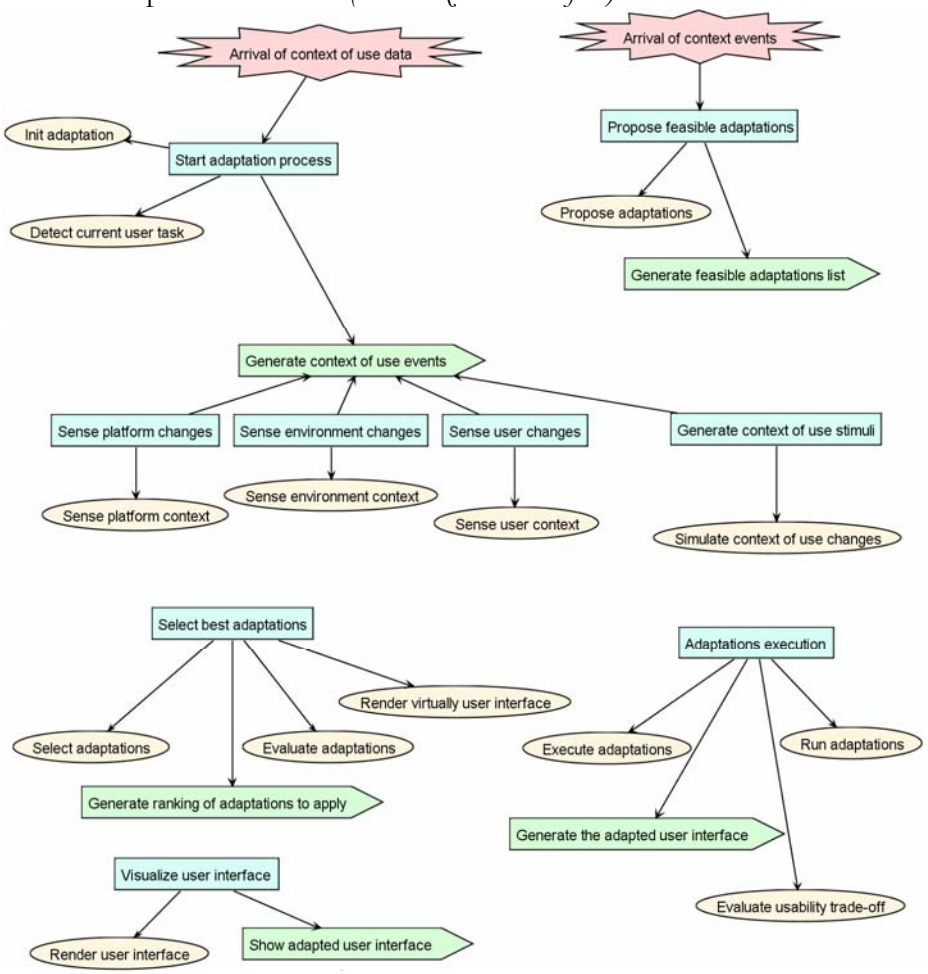


Figura 5.2 Papeles *(roles)* identificados en el sistema multi-agente.

5.2.2.1 Los Agentes y sus Papeles en el Sistema Multi-agente

La distribución de los distintos papeles entre los agentes que los desempeñarán da lugar a la definición de los agentes. Para ello se tienen en

cuenta factores como el grado de cohexión de los papeles que incluye cada agente. En la figura 5.3 se describe la distribución de los papeles que se ha realizado, así como los agentes resultantes de dicha distribución.

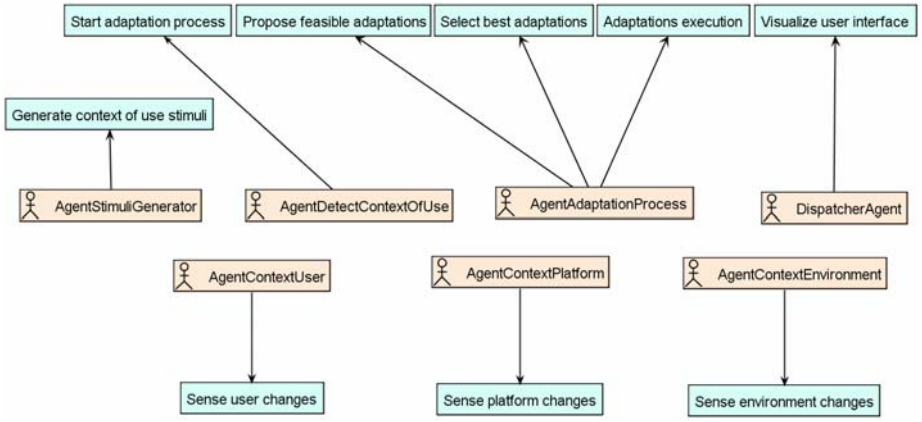


Figura 5.3 Distribución de los papeles entre los distintos agentes.

Se ha optado por mantener un agente para cada uno de los papeles encargados del tratamiento y generación de los eventos del contexto *(AgentContextUser*, *AgentContextPlatform*, *AgentContextEnvironment* y *AgentStimuliGenerator)*. El comienzo del proceso de adaptación ha sido encomendado al agente *AgentContextOfUse*. El agente *AgentAdaptationProcess* será el encargado de realizar la propuesta de las adaptaciones factibles, seleccionar las mejores y aplicarlas. Finalmente, el agente *DispatcherAgent* será el encargado de mostrar la interfaz de usuario.

5.2.3 Una visión general de la arquitectura

Una vez definidos los agentes implicados en la arquitectura para la adaptación de las interfaces de usuario propuesta, así como de los papeles que cada uno de dichos agentes desempeña, a continuación se describe una visión general del sistema multi-agente. En la figura 5.4 se puede observar dicha visión general. El agente *AgentAdaptationProcess* tendrá entre sus creencias, la información reunida durante el diseño de la interfaz de usuario en los modelos de interfaz de usuario concreta, interfaz de usuario abstracta, de dominio y de tareas (véase la sección 2.2.1). Dicho agente recibe los eventos que representan los cambios en el contexto desde los agentes *AgentContextPlatform*, *AgentContextUser* y *AgentContextEnvironment*, dependiendo de si el cambio en el contexto se ha producido en la plataforma, el usuario o el entorno respectivamente.

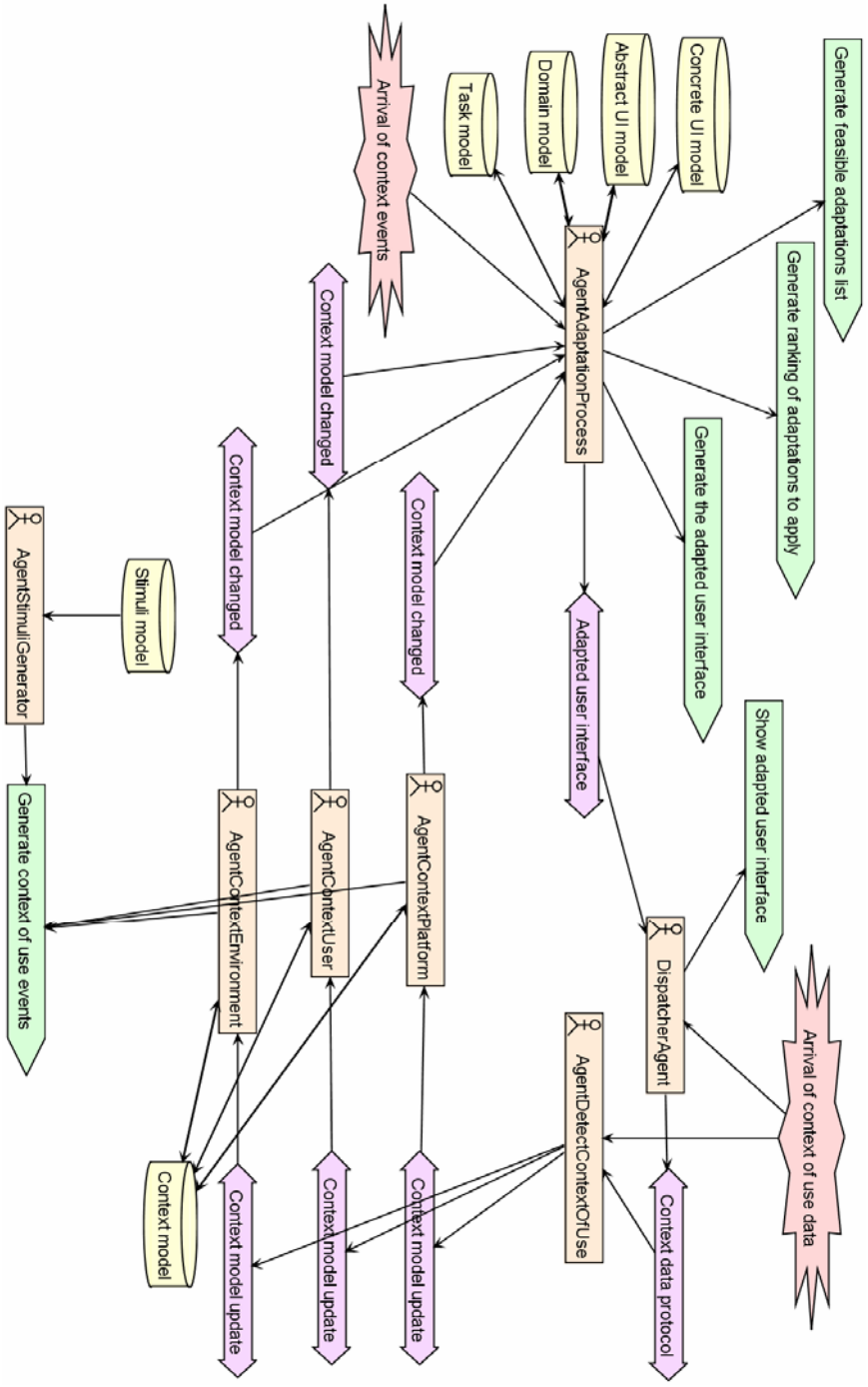


Figura 5.4 Vista general de la arquitectura multi-agente propuesta.

Los cambios en el contexto serán captados inicialmente por el agente *DispatchetAgent*, el cual los enviará al agente *AgentDetectContextOfUse*. Este agente decidirá sobre la relevancia o no de la información recibida de acuerdo a la especificación de los sensores, y le comunicará al agente correspondiente *(AgentContextPlatform*, *AgentContextUser* o *AgentContextEnvironment)* los cambios en el contexto. Dichos agentes serán los que finalmente produzcan los eventos del contexto que tratará el agente *AgentAdaptationProcess*, y a su vez actualizarán el modelo de contexto para reflejar los cambios. Tras la aplicación de las adaptaciones escogidas, el agente *AgentAdaptationProcess* enviará al agente *DispatcherAgent* la nueva interfaz de usuario adaptada para que sea mostrada al usuario.

Para facilitar la descripción del diseño de las facilidades creadas para producir el efecto de adaptación de la interfaz de usuario se describirá cada una de las etapas del proceso de adaptación por separado, describiéndose entonces cada uno de los agentes y modelos propuestos para cada fase. El proceso de adaptatión completo se encuentra descrito en la figura 5.5.

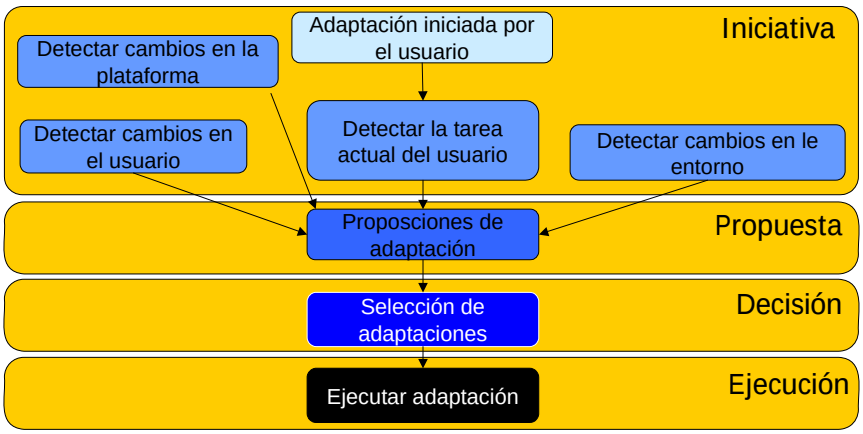


Figura 5.5 Etapas en el proceso de adaptación usado en la arquitectura propuesta.

5.2.4 Etapa de iniciativa

La etapa de iniciativa puede ser lanzada por dos causas:

- El sistema detecta a través de los sensores un cambio en el contexto de uso (usuario, plataforma, entorno físico, tarea actual), e intenta reaccionar.

- El usuario explícitamente expresa su deseo de adaptar el sistema.

En el segundo caso, si el usuario ha expresado su deseo de que comience el proceso de adaptación, el sistema necesita completar la información del contexto intentando inferir cuál es el objetivo actual del usuario, es decir, la tarea actual del usuario, de forma que pueda proponer adaptaciones que sean apropiadas.

5.2.4.1 Inferencia de la tarea actual del usuario

La inferencia de cuál es la tarea actual del usuario se basa en examinar el modelo de tareas creado en tiempo de diseño y el histórico de la interacción del usuario con la aplicación.

Figura 5.6 Ejemplo de conjunto de tareas activas.

A lo largo de la interacción del usuario con la aplicación sólo un conjunto reducido de tareas concreto puede ser realizado. Este conjunto de tareas posibles, dentro de la notación propuesta en CTT, es conocido como el "conjunto de tareas activas" (*enabled task set*), y puede ser calculado en cada momento a partir de un modelo de tareas basado en la notación CTT como el usado en AB-UIDE. Por supuesto, dicho conjunto cambia conforme el usuario va navegando a través del modelo de tareas durante el uso de la interfaz.

En la figura 5.6 se muestra el conjunto inicial de tareas activas para un ejemplo propuesto para la herramienta CTTE[27].

[27] http://giove.cnuce.cnr.it/ctte.html

Sin embargo, en un momento dado de la interacción el conjunto de tareas activas puede contener más de una tarea, y por lo tanto será necesario que el sistema infiera cuál es, entre las tareas que contiene el conjunto de tarea activas, la que realmente está realizando el usuario. Para ello, se han diseñado una serie de heurísticas que guían al sistema en la inferencia de la tarea actual que el usuario está llevando a cabo:

- Ya que a través del modelo de conectores (véase la sección 4.2.3.4) se mantiene en tiempo de ejecución la correspondencia entre las tareas y los componentes concretos con los cuales el usuario interactúa para realizarlas, el sistema estimará como más plausibles aquellas tareas en las que se haya manipulado recientemente alguno de los elementos de la interfaz de usuario incluidos en el contenedor donde se ejecuta esa tarea. Para ello, ha sido necesario añadir un nuevo atributo a los objetos concretos de interacción de usiXML, que almacena el instante de tiempo en que se produjo la última interacción con el objeto concreto de interacción.

- El proceso de inferencia de la tarea actual comienza explorando por las acciones del modelo de tareas, es decir, por las acciones concretas, ya que permiten una inferencia más precisa. En aquellos casos en que no se puede inferir qué acción es la actual, se irá ampliando el espectro de búsqueda a tareas más abstractas.

- Las propiedades que describen cada una de las tareas del conjunto de tareas activas ayudan a averiguar cuál es la tarea actual.

    - Frecuencia: las tareas con mayor frecuencia tienen mayores probabilidades de ser la tarea actual.
    - Tipo de tarea: sólo las tareas del usuario pueden ser tareas actuales.
    - Tareas finales: las tareas que finalizan la ejecución de la aplicación son menos probables.
    - Tipo de acción: el tipo de acción ayuda en la identificación de secuencias de tareas recurrentes.

- La identificación de secuencias de tareas recurrentes puede ayudar a predecir cuál es la tarea actual del usuario o la siguiente tarea por realizar.

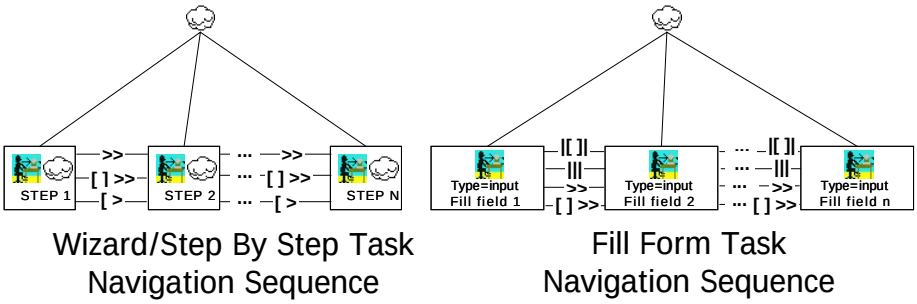


Figura 5.7 Ejemplo de secuencias de tareas recurrentes.

A través de la aplicación de las heurísticas presentadas el sistema completa la información sobre el contexto antes de proponer cuáles son las adaptaciones aplicables en la situación actual.

5.2.4.2 Detección de cambios en el contexto: sensores

Una interfaz de usuario adaptativa se basa en el principio de reacción ante los cambios detectados en el contexto de uso de la aplicación. Por otro lado, el principio básico de un agente es la reacción de acuerdo al conocimiento interno que posee dentro de sus creencias ante los cambios que suceden en su entorno, detectados mediante sus sensores, y actuar sobre su entorno a través de sus efectores. En el caso que nos ocupa, el entorno en el que los agentes se encuentran inmersos es la interfaz de usuario y su contexto de uso. Por lo tanto sus sensores deben ser capaces de detectar cualquier cambio significativo en la interfaz de usuario y su contexto de uso. Es necesario describir la información que es significativa para el sistema, así como la fuente a partir de la cual se obtiene dicha información. Es decir, es necesario describir los sensores a partir de los cuales se obtiene la información, y la información que producen.

La figura 5.8 muestra el metamodelo propuesto para la descripción de los sensores del sistema. En el metamodelo se distinguen dos tipos de sensores, sensores hardware y sensores software. Los sensores hardware representan sensores que proporcionan datos a través de algún tipo de circuitería, mientras que los sensores software son programas situados en la interfaz para recoger datos acerca del contexto que son relevantes para el sistema.

Tanto los sensores software como hardware pueden estar destinados a la detección de cambios en el usuario, la plataforma o el entorno. Los datos recibidos a través de los sensores se agrupan para producir eventos del

contexto (*contextEvents*). Los eventos del contexto serán los que finalmente producirán la ejecución de una serie de reglas de adaptación.

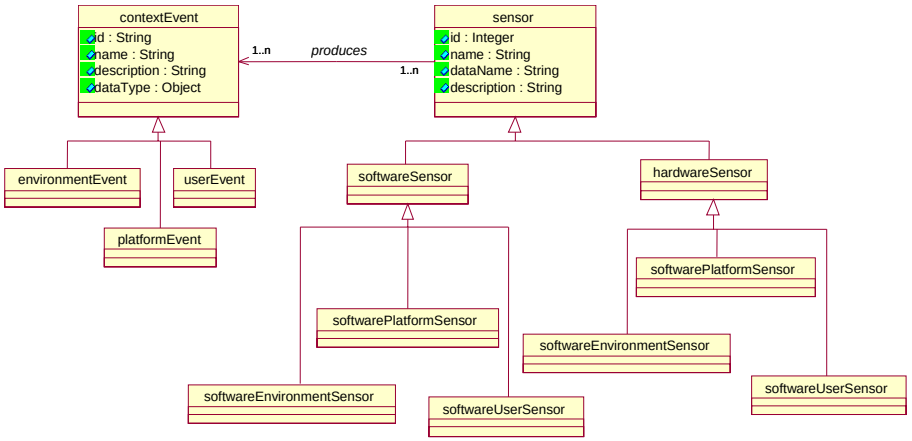


Figura 5.8 Metamodelo de sensores propuesto.

Los sensores constan de dos partes. Por una parte contienen una descripción de cómo se capturan los datos en el cliente, y qué tipos de datos adquieren, y por otra parte constan de una especificación sobre qué tratamiento deben recibir una vez que los datos adquiridos llegan al sistema multi-agente. Todos los datos que llegan de los sensores tienen un tratamiento por defecto, sin embargo, ciertos datos deben ser redireccionados a distintos agentes para recibir un procesamiento específico.

Todos los datos que los sensores son capaces de capturar describen la información del contexto (*context information*). La información del contexto es enviada periódicamente al sistema multi-agente para su procesado. Ciertas partes de dicha información permanecerán fijas siempre de una aplicación a otra, sin embargo, para cada dominio de aplicación a través de la especificación de los sensores se puede definir nueva información relevante para la interfaz de usuario que extienda el modelo de información de contexto inicial. La figura 5.9 muestra un ejemplo de información de contexto creado a partir de la definición de un conjunto de sensores. En dicha figura se observa como la información del contexto está descompuesta en información del usuario *(userInfo)*, información de la plataforma *(platformInfo)* e información del entorno *(environmentInfo)*. La información de la plataforma está a su vez dividida en información sobre la plataforma software donde se ejecuta la aplicación *(softwarePlatform)*,

información sobre la plataforma hardware donde se ejecuta *(hardwarePlatform)*, y características de la red de comunicaciones a la que está conectado el dispositivo donde se ejecuta la aplicación *(networkCharacteristics)*.

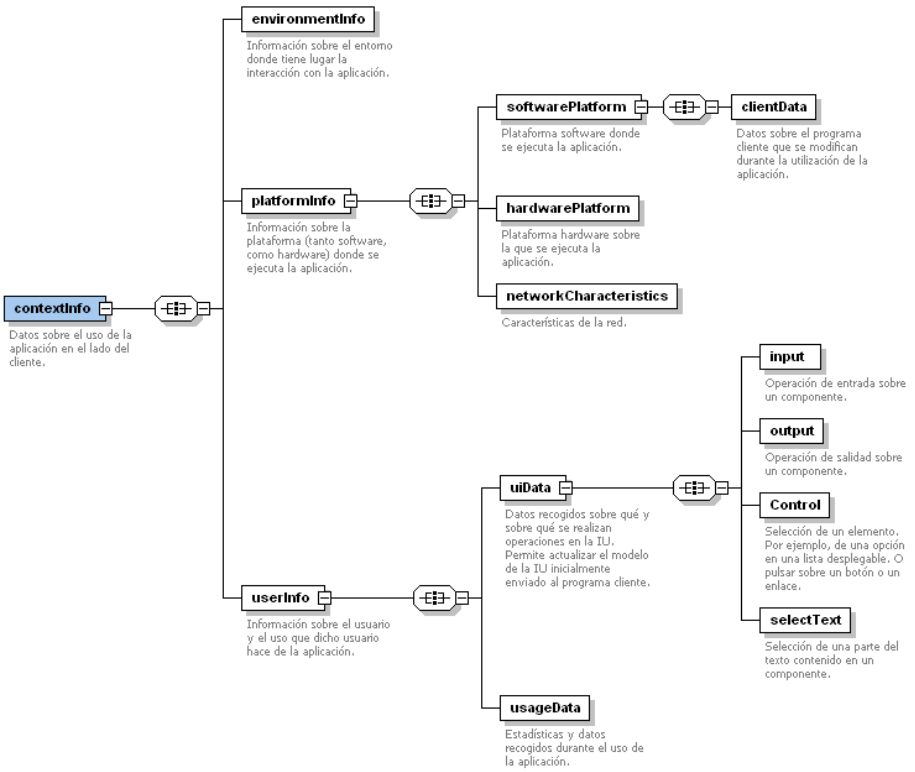


Figura 5.9 Ejemplo de modelo de información del contexto.

La información del contexto incluye dentro de la información del usuario dos tipos de datos. Por una parte contiene los datos sobre las operaciones que el usuario realiza sobre la interfaz *(uiData)*, como por ejemplo cuando el usuario introduce un valor en una caja de edición, y por otra parte contiene información sobre las estadísticas de uso de la aplicación, como puede ser el tiempo en que el usuario no está manipulado la aplicación *(idleTime)* o el último componente que el usuario ha manipulado.

Relevancia de la información del contexto

La mera recepción por parte del sistema multi-agente de un cambio en el contexto de uso de la aplicación no provoca automáticamente un cambio en la visión que el sistema multi-agente tiene del contexto, es decir, en las

creencias de los agentes. Dentro del sistema es posible especificar si un cambio en el contexto es relevante o no. Sin embargo, una nueva dificultad se presenta. Por ejemplo, un cambio en el tamaño de una ventana de 20 píxeles puede no ser relevante cuando se está trabajando en un PC, pero puede ser muy relevante si se está trabajando en una PDA.

Por ello se ha incorporado lógica difusa a la especificación de la relevancia de los datos de un sensor. La lógica difusa permite especificar condiciones lógicas utilizando conjuntos difusos.

Conjuntos difusos

Los conjunto difusos extienden el concepto de conjunto clásico. En la teoría clásica un conjunto es descrito especificando sus elementos o la condición que debe cumplir un elemento para formar parte del conjunto. Un conjunto difuso se describe mediante una función de pertenencia μ, que asocia a cada elemento un grado de pertenencia al conjunto dentro del intervalo [0,1]. La principal ventaja de esta aproximación en la definición de conjunto es que un elemento no tiene por qué pertenecer totalmente al conjunto, es decir, no todo es blanco o negro, sino que es también posible especificar tonalidades de gris.

La forma más habitual de especificar las funciones de pertenencia de los conjuntos difusos es mediante la descripción del conjunto a través de funciones predeterminadas, y de los parámetros que definen dichas funciones. Es habitual utilizar funciones que definen trapecios, triángulos, campanas de Gauss o formas sinusoidales. En la figura 5.10 se muestra la especificación de la función de pertenencia del cambio de tamaño de la pantalla en una PDA.

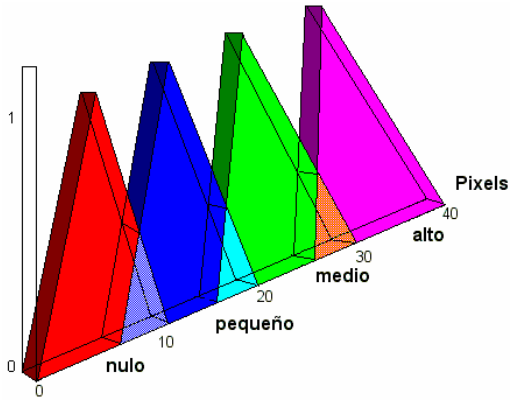


Figura 5.10 Conjunto difuso para el cambio de tamaño de la pantalla en una PDA.

Especificación de la relevancia de información

Sólo la información que produce cambios significativos en el sistema deber ser procesada, ello evita la pérdida de rendimiento del motor de adaptación debida a información que finalmente no conducirá a la aplicación de ninguna regla.

Los conjuntos difusos son utilizados en la definición de las reglas que definen si una información es relevante o no. Ello permite la especificación de la relevancia de los datos para distintas plataformas. El diseñador simplemente necesita especificar los conjuntos difusos para cada uno de los tipos de plataformas destino, de forma que la regla de relevancia seguirá siendo aplicable una vez ajustada la magnitud de los cambios a la plataforma actual a través de la definición de los conjuntos difusos para dicha plataforma.

La figura 5.11 muestra un ejemplo de especificación de una regla difusa de relevancia, donde sólo se considera un cambio en el tamaño de una ventana si es "grande". Donde "grande" pertenece uno de los conjuntos difusos definidos.

```
if window_size_increment is grande then
  platformEvent(old_Window_Size, new_Window_Size);
```

Figura 5.11 Ejemplo de regla de relevancia para el cambio de tamaño de la ventana.

5.2.4.3 Diseño de la etapa de iniciativa en el sistema multi-agente

La comunicación del sistema multi-agente con la plataforma cliente donde se presenta la interfaz de usuario finalmente es realizada a través del agente *DispatcherAgent*, que actúa como interfaz entre la plataforma cliente y el sistema multi-agente. La descripción de la interfaz concreta entre el sistema multi-agente y la plataforma cliente se encuentra descrita con mayor detalle en la sección 5.3.

El agente *DispatcherAgent* a través de su plan *PlanContextEventGeneration* procesa la información del contexto recibida en formato XML siguiendo el esquema descrito en la sección 5.2.4.2. Cada uno de los cambios recibidos es enviado al agente *AgentDetectContextOfUse*, el cual será el encargado de aplicar las reglas de relevancia (véase la sección 5.2.4.2), para decidir si el cambio es significativo para el sistema, o si por el contrario es

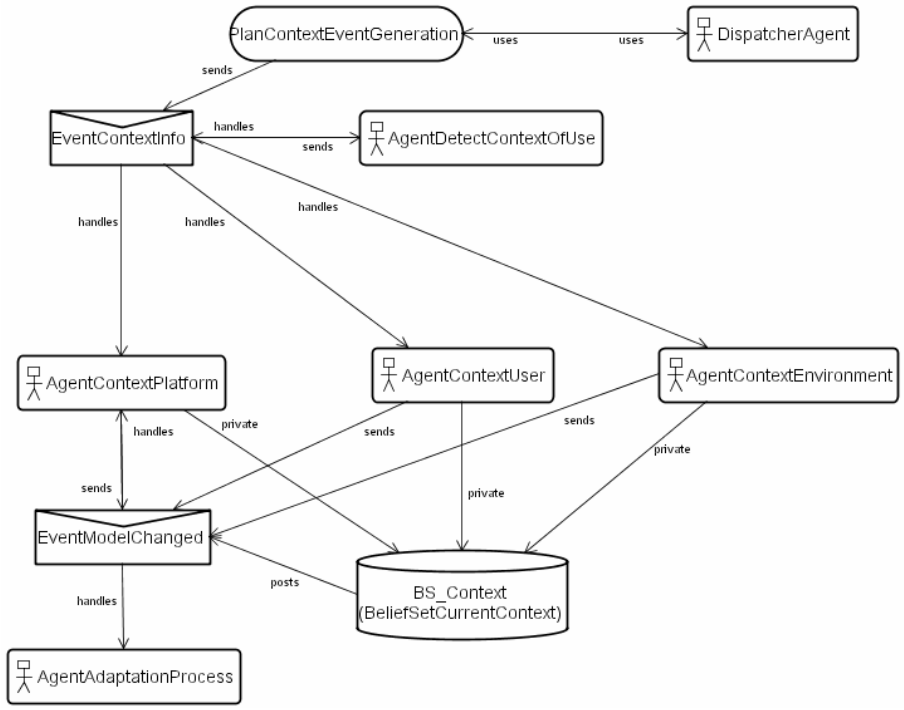


Figura 5.12 Diseño del sistema multi-agente para la etapa de iniciativa de la adaptación.

descartado. En el caso en que se considere el cambio como significativo, dicho cambio será enviado al agente correspondiente, es decir, los cambios detectados en el usuario serán enviados al agente *AgentContextUser*, los cambios en la plataforma serán enviados al agente *AgentContextPlatform*, y los cambios detectados en el entorno físico serán enviados al agente *AgentContextEnvironment*. Cada uno de estos tres agentes se encargará de realizar cualquier tipo de tratamiento concreto para el cambio recibido especificado en la definición del sensor diseñado para captar dicho cambio, y actualizará el modelo de contexto (*BeliefSetCurrentContext*), para mantener una visión actualizada del conocimiento que el agente tiene del entorno. Una vez almacenado el cambio en el modelo de contexto actual (actualización de las creencias de los agentes), se comunicará el cambio al agente *AgentAdaptationProcess*, el cual será el encargado de proponer posibles adaptaciones que sean adecuadas a los cambios detectados.

5.2.5 Etapa de proposición de adaptaciones

En la etapa de proposición se deben sugerir aquellas adaptaciones que sean apropiadas dado el contexto de uso actual. Para un agente basado en

el paradigma BDI, las posibles alternativas que puede tomar para afrontar la realización de una meta son sus planes. Dentro del sistema multi-agente propuesto, las posibles adaptaciones o soluciones a un cambio de contexto están reflejadas en los planes del agente *AgentAdaptationProcess*. Cada uno de dichos planes representa una regla de adaptación, la cual es diseñada siguiendo el metamodelo mostrado en la figura 5.13.

En el metamodelo mostrado en la figura 5.13 se describen las reglas de adaptación. Una regla de adaptación (*adaptivityRule*) es descrita en función de los eventos del contexto que la producen (véase la sección 5.2.4.2), una precondición de aplicación para el contexto, los datos que la regla necesita leer o que produce durante su ejecución, y la transformación en sí que produce la regla.

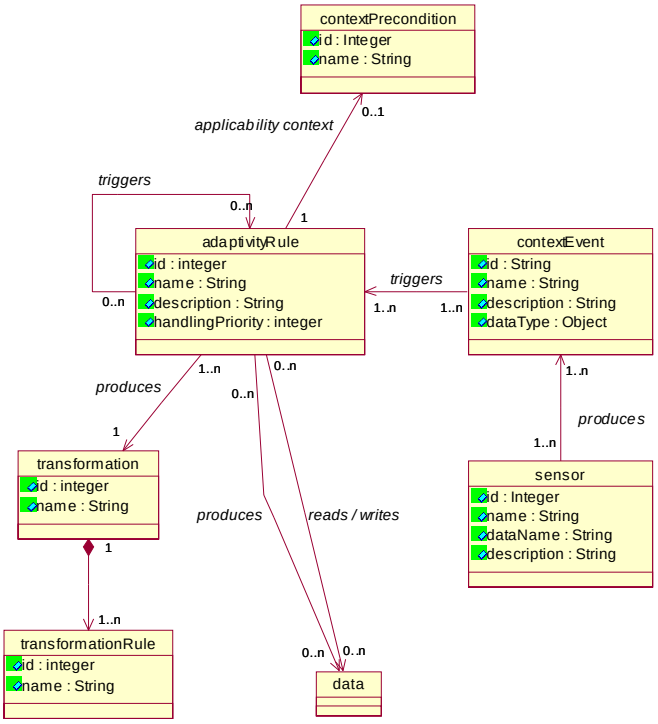


Figura 5.13 Metamodelo para la especificación de reglas de adaptación.

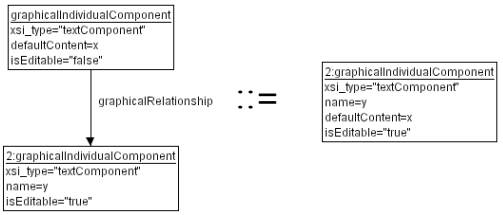
Es necesario destacar que un mismo evento del contexto podrá producir la activación de varias reglas distintas, y por lo tanto será necesario un mecanismo capaz de discernir qué regla, de entre las activadas por un

evento del contexto, es mejor en cada momento. De igual manera, se pueden especificar secuencias de reglas que se ejecuten de forma secuencial. Es decir, una regla podrá activar otras reglas.

La precondición del contexto actúa como un filtro adicional a la activación de reglas. Por ejemplo, si se produce un cambio en el tamaño de la ventana, y se ha estimado que dicho cambio es significativo mediante las reglas de relevancia, puede ser que aún así ese cambio no sea significativo para aquellas reglas de adaptación que sean aplicables cuando se aumente el tamaño, y no cuando se reduzca. Ese tipo de precondiciones pueden ser especificadas en la precondición del contexto de la regla (*contextPrecondition*).

Finalmente, la transformación permite especificar la adaptación real de la interfaz de usuario que se desea realizar. Dicha transformación debe ser expresada en función de los modelos definidos en tiempo de diseño, y actualizados en tiempo de ejecución, utilizando la especificación de reglas de transformación de gramáticas de grafos atribuidos descritas en la sección 3.4.1.4. Un ejemplo de regla de adaptación se encuentra en la figura 5.14, donde se ejemplifican los términos introducidos.

```
<?xml version="1.0" encoding="UTF-8"?>
<adaptivityRule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="adaptivityRule-1.1.xsd" id="AR001"
        name="shrinkEditBoxes"
        description="Regla de adaptación que substituye una caja de texto y su etiqueta asociada por una
                caja de texto con el rótulo de la etiqueta dentro.">
        <contextEvent id="CE001" name="contextEventShrinkEditBox">
                <softwarePlatformSensor id="SPS001" name="defaultSoftwarePlatformSensor"
                        dataName="/contextInfo/platformInfo/softwarePlatform/clientData/screenSize"
                        dataType="_2x2_Dimension_"
                        description="Senses whenever screen size changes."/>
        </contextEvent>
        <contextPrecondition id="CP001" name="contextPreconditionShrinkEditBox">
                <logicalSentence sentence="(SPS001.newValue <=320x200 )"/>
        </contextPrecondition>
        <transformation>
```




```
        <transformation/>
        <data/>
</adaptivityRule>
```

Figura 5.14 Ejemplo de especificación de regla una de adaptación.

Por lo tanto, el conjunto de posibles adaptaciones propuestas por el sistema serán aquellos planes activados por el evento del contexto producido por los cambios actuales, pero cuya precondición del contexto sea evaluada positivamente. A continuación el proceso de selección de adaptaciones determinará cuál es la adaptación más apropiada al caso actual.

5.2.5.1 Diseño de la etapa de proposición en el sistema multi-agente

En el diseño la etapa de proposición en el sistema multi-agente propuesto, cada una de las reglas de adaptación diseñadas se transforma en un plan del agente *AgentAdaptationProcess* de forma automática. Por lo tanto, las adaptaciones posibles dado un cambio de contexto concreto serán aquellos planes de *AgentAdaptationProcess* que se activen ante la llegada de ese evento del contexto (*EventContextChanged*). La figura muestra la estructura descrita, donde tres reglas de adaptación han sido definidas (*PlanUnShrinkEditBox*, *PlanShrinkEditBox* y *PlanShrinkEditBox2*).

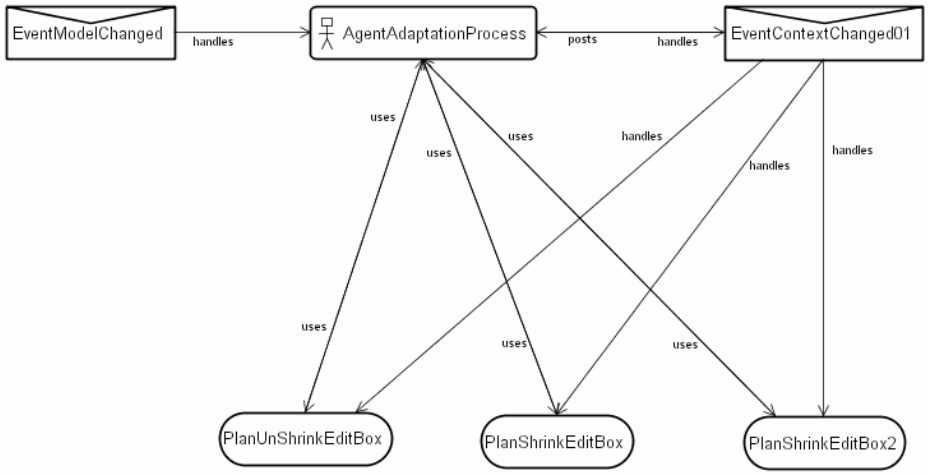


Figura 5.15 Diseño del sistema multi-agente para la etapa de proposición de adaptación.

5.2.6 Etapa de decisión: selección de adaptaciones

La etapa de decisión puede ser llevada a cabo siguiendo dos cauces. Por una parte, el usuario puede elegir la adaptación que prefiera, siguiendo una aproximación de iniciativa mixta, y por otra parte el sistema puede elegir cuál es la mejor adaptación a aplicar. En el segundo caso el sistema necesita ser capaz de evaluar cuán beneficiosa sería la aplicación de cada

una adaptaciones propuestas en la fase anterior, es decir, es necesario un método de evaluación de la bondad de las adaptaciones propuestas.

5.2.6.1 Evaluación de la bondad de una adaptación

La evaluación de las adaptaciones se basa en dos conceptos:

- Coste de migración: representa el esfuerzo físico, cognitivo y conativo que el usuario debe aplicar para migrar de un contexto a otro [Cal01a].

- Beneficio de adaptación: representa cuán buena será una adaptación para el usuario en el nuevo contexto.

En el modelo propuesto actualmente no se ha tenido en cuenta el coste monetario que puede estar asociado a una migración. Hay que tener en cuenta que en un futuro cercano, no será extraño el hecho de tener que pagar por la utilización de un servicio Web, y que por lo tanto adaptaciones que hagan uso de distintos servicios podrán tener también distintos costes monetarios asociados.

Coste de migración

Para la evaluación del coste de migración se ha asociado a los componentes de la definición de coste de migración (esfuerzo cognitivo, esfuerzo conativo) criterios que permitan su evaluación. En la versión actual se ha dejado fuera de la evaluación el coste físico, ya que excepto para entornos de interacción muy concretos, se hace muy difícil, o imposible de evaluar.

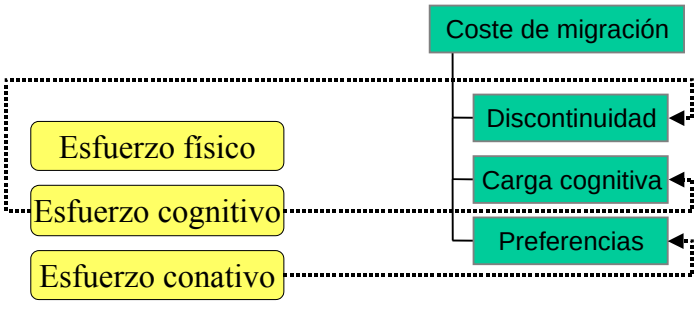


Figura 5.16 Relación entre coste de migración y criterios de evaluación.

El esfuerzo cognitivo es evaluado a través de dos criterios: discontinuidad y carga cognitiva, mientras que el esfuerzo conativo es evaluado de acuerdo a las preferencias del usuario.

La discontinuidad puede ocurrir cuando el usuario se ve forzado a dividir su atención entre dos entidades. Esto puede ocurrir por ejemplo cuando una adaptación cambia la disposición de los elementos de la interfaz de usuario de una forma drástica, haciendo que el usuario tenga que reposicionar su foco de atención.

La carga cognitiva que representa una adaptación de la interfaz de usuario se manifiesta por el aumento o disminución de la cantidad de información que el usuario debe procesar para realizar las tareas actuales. Por lo tanto, aquellas adaptaciones que impliquen un aumento o disminución de la información presentada (como por ejemplo es la aplicación de las técnicas como *stretch text* o *acordion* [Bru03]) afectan directamente a la carga cognitiva.

Finalmente, las preferencias del usuario serán aplicadas como medida correctora de las dos métricas aplicadas al coste de migración anteriormente expuestas. De esta forma, si el usuario, por ejemplo, tiene problemas de visión y prefiere la información en modo texto, aunque suponga un aumento de la carga cognitiva frente a la información presentada de forma gráfica el coste de migración será ponderado para reflejar dicha preferencia del usuario, es decir, para ponderar la disminución en la carga conativa que dicha preferencia supone.

Por lo tanto, durante el proceso de evaluación de las posibles adaptaciones a aplicar se realiza una primera evaluación de la usabilidad de la interfaz de usuario que el proceso de adaptación va a producir.

La discontinuidad se evalúa de acuerdo al esfuerzo mental necesario para retomar la actividad interrumpida por la adaptación. Para realizar la evaluación del esfuerzo mental necesario se realiza una estimación basada en el modelado cognitivo de la interacción, y especialmente en los resultados experimentales y teóricos que se han obtenido del estudio de la interacción hombre-máquina, y más concretamente de los obtenidos a partir del análisis de modelos especificados usando técnicas basadas en GOMS (*Goals*, *Operators*, *Methods*, *Selection Rules*) [Car83].

GOMS

GOMS se basa en la teoría del modelo humano de procesador. En GOMS se describen las metas (*Goals*) que el usuario quiere alcanzar (tareas), en función de unos métodos (*Methods*) especificados en función de acciones

elementales (*Operators*) perceptivas, motoras o cognitivas, y una serie de estrategias de selección (*Selection Rules*) entre los posibles métodos a aplicar para alcanza una meta. GOMS ha sido encontrado especialmente útil en [New85] :

1. La restricción del espacio de diseño inicial, evitando construir interfaces de usuario que necesiten una capacidad de memoria que supere la memoria a corto plazo de una persona.
2. Responder a cuestiones de diseño, escogiendo entre distintas alternativas de diseño de acuerdo a los valores de la evaluación.
3. Estimar el tiempo total en realizar una tarea.
4. Proporcionar una base a partir de la cual calcular el tiempo de aprendizaje de la interfaz de usuario.
5. Conocer qué etapas de la interacción llevan más tiempo o producen más errores.

En nuestro caso, los puntos 3 y 5, son los que nos han hecho decantarnos por esta aproximación. Basándose en la experimentación con especificaciones en GOMS y las aplicaciones finales asociadas a ellas, se estimó el tiempo necesario para la realización de cada una de las tareas básicas de interacción, usando para los experimentos aplicaciones de edición de texto, sistemas gráficos, y algunas funciones del sistema operativo. De igual manera, experimentos posteriores [You88] demostraron que los valores obtenidos para el cálculo del tiempo necesario para realizar una tarea de forma empírica eran similares a los obtenidos mediante el cálculo aplicando GOMS.

Los valores estimados por Card, Moran, Newell y otros [Car83][New85] [Ols90] empíricamente utilizados en la evaluación de la bondad de las reglas de adaptación se encuentran recogidos en la tabla 6.

Tabla 6. Estimación de tiempos para las acciones básicas de interacción en GOMS.

Parámetro	Tiempo estimado
Pulsar una tecla	230ms
Apuntar con un ratón	150ms
Mover la manos al ratón	360ms
Mover las manos sobre el teclado	360ms
Percibir un cambio	100ms
Hacer una *saccade*[28]	230ms
Entender una palabra de seis letras	340ms

[28] Tiempo en mover la vista y asimilar la información en cada golpe de vista.

Evaluación de la discontinuidad

La evaluación de la discontinuidad producida por una adaptación en la interfaz de usuario supone la evaluación de los distintos efectos que una adaptación puede causar en la interfaz de usuario que produzcan discontinuidad, teniendo siempre en cuenta que la aparición de un efecto de discontinuidad en una adaptación puede implicar la aparición de otros como efecto lateral. En la tabla 7 se muestran los efectos de las adaptaciones que producen discontinuidad considerados dentro de este trabajo.

Tabla 7. Efectos de adaptación considerados en la discontinuidad.

Efecto de adaptación
Agrandar/reducir un *widget*
Mover un *widget*
Eliminar a *widget*
Añadir un nuevo *widget*
Sustituir un *widget*
Añadir un nuevo contenedor
Cambiar la distribución de los *widgets*/contenedores de un contenedor
Agrandar/Reducir un contenedor
Eliminar un contenedor

A continuación se describe cada uno de los efectos de adaptación recopilados en la tabla 7, así como la fórmula propuesta para el cálculo de la discontinuidad producida por cada efecto:

- Agrandar/reducir un widget: en este caso la discontinuidad es producida tanto por el cambio del tamaño del *widget*, como por los cambios producidos en los elementos contenidos en el mismo contenedor que el que ha cambiado de tamaño.

$$(\Delta Size) / ScreenResolution + \sum_{i=1}^{n-1} Discontinuity(e_i)$$

  - ΔSize = incremento/decremento de tamaño del widget en píxeles.
  - n = número de widgets dentro del contenedor en el que está contenido el widget que está cambiando de tamaño.
  - *Discontinuity* = discontinuidad producida por el elemento i. e_i debe ser distinto del elemento al que se está cambiando el tamaño.
  - *ScreenResolution* = resolución actual del dispositivo donde se muestra la interfaz de usuario.
  - e_i = Elemento$_i$ de la interfaz de usuario evaluada.

- Mover un widget: la discontinuidad en este caso es el número de píxeles que el *widget* ha sido desplazado (verticalmente, horizontalmente o diagonalmente) dividido por el número mínimo de píxeles necesario para que el movimiento sea significativo. Puede aparece discontinuidad adicional si otros elementos son desplazados por el movimiento del *widget* actual.

$$\frac{\Delta Position}{ScreenResolution} + \sum_{i=1}^{n-1} Discontinuity(e_i)$$

  - ΔPosition = incremento/decremento en la posición del widget en píxeles.
  - *n* = número de widgets dentro del contenedor en el que está contenido el widget que se está moviendo.
  - *Discontinuity* = discontinuidad producida por el elemento *i*. e_i debe ser distinto del elemento al que se está moviendo.
  - *ScreenResolution* = resolución actual del dispositivo donde se muestra la interfaz de usuario.
  - e_i = Elemento$_i$ de la interfaz de usuario evaluada.

- Eliminar un widget: la eliminación de un *widget* produce discontinuidad si los *widgets* que lo rodean cambian para ocupar el espacio liberado.

$$\sum_{i=1}^{n-1} Discontinuity(e_i)$$

  - *n* = número de widgets dentro del contenedor en el que está añdiendo el widget.
  - e_i = Elemento$_i$ de la interfaz de usuario evaluada.
  - *Discontinuity* = discontinuidad producida por el elemento *i*. e_i debe ser distinto del elemento que se está añadiendo.

- Añadir un widget: cuando se añade un nuevo *widget*, puede aparecer discontinuidad si los elementos del contenedor donde está siendo añadido el widget tienen que ser modificados para poder añadir el nuevo *widget*.

$$\sum_{i=1}^{n} Discontinuity(e_i)$$

  - *n* = número de widgets dentro del contenedor en el que está añadiendo el widget.
  - e_i = Elemento$_i$ de la interfaz de usuario evaluada.
  - *Discontinuity* = discontinuidad producida por el elemento *i*.

- Cambiar la distribución de los widgets/contenedores de un contenedor: cambiar la distribución en un contenedor produce normalmente una gran discontinuidad. La discontinuidad depende de la discontinuidad individualmente producida por cada uno de los elementos del contenedor. Si el contenedor contiene otros contenedores, los cuales también cambian la distribución de sus componentes, la discontinuidad se irá calculando recursivamente, sino los contenedores serán tratados igual que si se tratara de *widgets* moviéndose o cambiando de tamaño.

$$\sum_{i=1}^{n} Discontinuity(e_i)$$

  - n = número de widgets dentro del contenedor en el que está ha cambiado la distribución.
  - e_i = Elemento$_i$ de la interfaz de usuario evaluada.
  - *Discontinuity* = discontinuidad producida por el elemento *i*.

- Sustitución de widgets: cuando unos *widgets* son sustituidos por uno o más *widgets*, la discontinuidad puede aparecer debido a dos motivos distintos. Por una parte, los nuevos *widget* pueden ocupar un área de pantalla distinta a la que ocupan los *widgets* originales (porque tienen distintos factores de visibilidad). Por otro lado, la interacción con los nuevos *widgets* puede ser distinta, teniendo el usuario que cambiar, por ejemplo, de escribir una opción en un cuadro de texto a seleccionar usando el ratón una valor en una lista desplegable.

$$\left| \frac{\sum_{i=1}^{n} \Delta Visibility()}{MaxWidgetsInLHS * Max(visibility)} \right| + \left| \frac{\Delta InteractionTechnique()}{Max(InteractionTechnique)} \right|$$

  - ΔVisibility = visibilidad de los nuevos widgets menos la visibilidad de los widgets sustituidos (véase la tabla siguiente).
  - ΔInteractionTechnique = representa el esfuerzo que el usuario debe hacer para cambiar de la técnica de interacción actual a la técnica de interacción necesario para interactuar con los nuevos widgets.
  - MaxWidgetsInLHS = número máximo de widgets en la especificación de parte izquierda de una regla de adaptación.

A continuación se describe la visibilidad (espacio relativo que ocupa cada widget en pantalla) de cada uno de los widgets. A

partir de estas visibilidades se puede establecer el incremento/decremento de visibilidad al añadir, quitar o reemplazar elementos de la interfaz de usuario. Grandes diferencias en las visibilidades producirán discontinuidad, ya que el usuario tendrá que mover su foco de atención para centrarlo en los nuevos elementos sobre los que debe seguir realizando sus tareas.

Tabla 8. Visibilidad de los *widgets* de una interfaz de usuario gráfica.

Widget (usiXML CUI elements)	Visibilidad
textComponent (labels, edit boxes, ...)	1 o el número de líneas
button	1
radioButton	1
toggleButton	1
checkBox	1
item	1
slider	1 (2 para el cálculo de la carga cognitiva)
imageComponent	imageHeight / altura de la línea para la fuente por defecto para la resolución actual.
comboBox	1 o el número de opciones visibles si es de tipo *listBox*.
menu	número de elementos del menú
spin	1 (número de *ítems* para el cálculo de la carga cognitiva)
menuPopUp	número de *ítems*
menuBar	número de menús
table	La suma de las visibilidades de los elementos contenidos en cada celda
tree	El número de ramas y hojas visibles.
box / dialogBox	La suma de los elementos que contienen.
tabbedDialogBox	La suma de los elementos que contiene la pestaña actualmente activa
window	La suma de la visibilidades de las cajas (*boxes*) que contiene.

Algunos componentes difieren en el valor de su visibilidad para el cálculo de la discontinuidad y de la carga cognitiva (slider y spin). Un slider ocupa un sitio de una línea. Sin embargo, la cantidad de información que el usuario debe procesar es 2, el valor máximo y el valor mínimo.

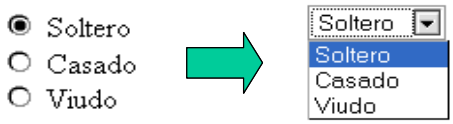
La siguiente tabla describe los dos cambios de técnica de interacción más habituales:

Tabla 9. Incremento de discontinuidad producido por la técnica de interacción.

Cambio de técnica de interacción	Incremento de discontinuidad
De teclado a ratón	MoveHandOverMouse (360ms) + PressMouseButton (230ms) * WidgetClicks + PointWithMouse (1500ms)
De ratón a teclado	MoveHandsOverKeyboard (360ms) + PressKeyboardStroke (230ms) * WidgetKeystrokes

Es necesario tener en cuenta que en la tabla 9 no se ha tenido en consideración el tiempo necesario para reposicionar el foco de atención. Se supone que el foco es situado automáticamente por el sistema. *WidgetClicks* es el número de clics necesarios para interactuar con un *widget*. Por ejemplo, interactuar con una lista para elegir una opción necesita más clics que interactuar con un grupo de botones de radio para elegir una opción. *WidgetKeyboardStrokes* es el número de pulsaciones de teclas necesario para interactuar con un *widget*. Será distinto de un *widget* a otro. *MoveHandOverMouse* expresa el tiempo necesario para situar la mano sobre el ratón, mientras que *MoveHandsOverKeyboard* representa el tiempo necesario para situar las manos sobre el teclado.

La figura siguiente ejemplifica el cálculo de la discontinuidad asociada a una regla que consiste en una sustitución de *widgets*.



Cálculo de la discontinuidad

$$\left|\frac{visibility(comboBox1) - visibility(radioButtonGroup3)}{Max(visibility)}\right| + \left|\frac{InteractionTechnique(comboBox1) - InteractionTechnique(radioButtonGroup3)}{Max(InterationTechnique)}\right| = \left|\frac{1-3}{3}\right| + \left|\frac{(Mouse + 2clicks) - (Mouse + 1click)}{2.32}\right| = 0.66 + 0.099 = 0.766$$

Figura 5.17 Ejemplo de cálculo de discontinuidad para una adaptación con sustitución de *widgets*.

Evaluación de la carga cognitiva

Cuando se aplica una adaptación se puede producir una variación en la carga cognitiva que representa la información mostrada en la interfaz al usuario. De tal forma que para calcular el beneficio o perjuicio en términos de carga cognitiva que una adaptación produce es necesario

calcular el diferencial de carga cognitiva que la adaptación a aplicar produciría. Dicho diferencial se calcula de acuerdo al aumento o decremento de la información que el usuario debe procesar.

Al igual que la evaluación de la discontinuidad, la evaluación de la carga cognitiva que lleva consigo una adaptación es dependiente de incremento/decremento que cada efecto de adaptación produce en la interfaz de usuario. La tabla 10 recoge los efectos de adaptación considerados en la evaluación del incremento de la carga cognitiva producida por una adaptación.

Tabla 10. Efectos de adaptación considerados en la carga cognitiva.

Efecto de adaptación
Añadir *widgets*
Eliminar *widgets*
Eliminar a *widget*
Añadir texto
Eliminar texto
Sustituir *widgets* por otros *widgets*

A continuación se describe cada uno de los efectos de adaptación recopilados en la tabla 10, así como la fórmula propuesta para el cálculo de la carga cognitiva producida por cada efecto:

- Añadir *widgets* : cuando se añaden nuevos *widgets*, el incremento de carga cognitiva que el usuario debe procesar es el esfuerzo que el usuario tiene que emplear en entender el nuevo *widget*.

$$\frac{\sum_{i=1}^{n}\left(Perceive * Visibility(W_i)\right) + UnderstandWidget}{MaxWidgets * (Max(visibility) * Perceive + UnderstandWidget}$$

  - n = número de widgets añadidos.
  - *Perceive* = Tiempo en percibir (100ms).
  - *Visibility* = Representa el número de elementos en un widget que deben ser percibidos por el usuario.
  - *UnderstandWidget* = Tiempo necesario para entender cómo funciona un widget (230ms).
  - *MaxWidget* es el número máximo de widgets tanto en la LHS como en la RHS de la especificación de una regla de adaptación.
  - W_i = Widget_i de la interfaz de usuario evaluada.

- Eliminar *widgets* : reduce la carga cognitiva que el usuario debe procesar, y por lo tanto será negativa.

$$-\frac{\sum_{i=1}^{n}\left(Perceive * Visibility(W_i)\right)}{MaxWidgets * Max(visibility)}$$

- *n* = número de widgets añadidos.
- Perceive = Tiempo en percibir (100ms).
- Visibility = Representa el número de elementos en un widget que deben ser percibidos por el usuario.
- MaxWidget es el número máximo de widgets tanto en la LHS como en la RHS de la especificación de una regla de adaptación.
- W_i = Widget$_i$ de la interfaz de usuario evaluada.

- Añadir texto : aumenta la carga cognitiva del usuario proporcionalmente al número de palabras añadidas y al tiempo necesario para comprenderlas.

$$\frac{NumberWordsAdded * UnderstandWord}{MaxWordsInTextComponentTextInRHS * UndertandWord}$$

- *NumberWordsAdded* = número de palabras que se han añadido.
- *UnderstandWord* = Tiempo necesario para percibir y entender una palabra de 6 letras (340ms) (asumiendo que seis es la longitud media de una palabra).

- Eliminar texto : reduce la carga cognitiva de acuerdo al esfuerzo necesario para percibir y entender las palabras eliminadas.

$$-\frac{\left(NumberWordsRemoved * UnderstandWord\right)}{MaxWordsInTextComponentTextInLHS * UnderstandWord}$$

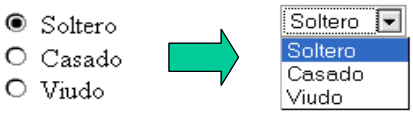
- *NumberWordsAdded* = número de palabras que se han añadido.
- *UnderstandWord* = Tiempo necesario para percibir y entender una palabra de 6 letras (340ms) (asumiendo que seis es la longitud media de una palabra).
- *MaxWordsIntextComponentInLHS* = número máximo de palabras en la especificación de un campo de texto en una regla de adaptación.

- Sustitución de unos *widgets* por otros : la carga cognitiva en la sustitución de *widgets* se calcula como la carga cognitiva producida por la eliminación de los *widgets* que son sustituidos menos la carga cognitiva producida por añadir los nuevos.

$$\frac{\sum_{i=1}^{n} perceive * Visibility(Wi) + UnderstandWidget - \sum_{j=1}^{m} perceive * Visibility(Wj)}{MaxWidgets * (Max(visibility) * perceive + UnderstandWidget)}$$

- *m* = número de widgets eliminados.
- *n* = número de widgets añadidos.
- *Visibility(Wj)* = visibilidad del widget *Wj*.
- *perceive* = tiempo necesario para percibir un widget (100ms).
- *UnderstandWidget* = tiempo necesario par comprender el funcionamiento de un widget (230ms).
- *MaxWidget* = máximo número de widgets eliminados/añadidos.

La figura siguiente ejemplifica el cálculo de la carga cognitiva asociada a una regla que consiste en una sustitución de *widgets*.



Cálculo de la carga cognitiva

$$\frac{\left[\left(Perceive * Visibility(comboBox)\right)\right] - \left[\left(Perceive * Visibility(radioButtonGroup)\right) + UnderstandWidget\right]}{MaxWidgets * (Max(visibility) * perceive + UnderstandWidget} =$$

$$= \frac{0.3 - 0.53}{0.53} = -0.434$$

Figura 5.18 Ejemplo del cálculo de la carga cognitiva producida por una regla de adaptación con sustitución de *widgets*.

Beneficio de adaptación

A la hora de ejecutar una adaptación habrá que tener en cuenta la impresión *(feedback)* que el usuario tuvo la última vez que se aplicó la adaptación. De esta forma, se tendrán en cuenta las veces que una adaptación ha sido aceptada por el usuario y las veces que una adaptación ha sido rechazada por este (la extracción de conclusiones puede ser individual o colaborativa. En el caso de que sea colaborativa las impresiones que otros usuarios han tenido de la adaptación actual también influirían en la toma de decisiones).

Los beneficios de las adaptaciones reducen el coste asociado al cambio de la interfaz original a la interfaz adaptada, ya que si una adaptación gusta al usuario (*user feedback*) debería ser más fácil que sea elegida para su

ejecución. Si la frecuencia del contexto que ha disparado la adaptación es alta significa que aunque el coste de la adaptación sea algo mayor, interesaría hacerla, ya que la situación para la que se ha determinado que es adecuada la adaptación se va a dar a menudo (véase la figura 5.19).

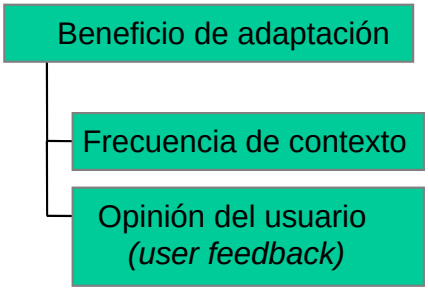


Figura 5.19 Componentes del beneficio de adaptación

Limitaciones del modelo

La aproximación para el cálculo del coste de migración propuesta anteriormente presenta una serie de limitaciones derivadas del propio modelo GOMS en el que se basa. Aunque el modelo GOMS ofrece unas estimaciones bastante exactas, no contempla una serie de factores que se dan en la interacción entre el usuario y la máquina dentro del modelo de procesador humano:

1. El modelo no contempla distintos tipos de usuarios. Es normal que los tiempos estimados no sean iguales para un usuario experto que para un principiante.

2. El modelo no tiene en cuenta la capacidad de aprendizaje de la interfaz de usuario (*learnability*). Es lógico que conforme el usuario adquiere una mayor familiaridad con la interfaz de usuario los tiempos se reduzcan.

3. No tiene en cuenta el tiempo empleado en errores.

4. No tiene en cuenta la naturaleza paralela del procesamiento humano, sino que asume un procesamiento secuencial de todas las acciones.

5. No tiene en cuenta la fatiga del usuario.

Por todo ello es necesario extender nuestra propuesta para ser capaces de mejorar las estimaciones a lo largo de la interacción con el usuario, es decir, es necesario que el propio sistema de selección de reglas de adaptación sea adaptativo.

Para afrontar este problema un sistema de aprendizaje Bayesiano es usado en tiempo de ejecución para hacer al sistema evolucionar y adaptarse en una mayor medida al usuario.

Aprendizaje en la selección de reglas

La idea en el aprendizaje Bayesiano es conseguir una fórmula que aplicada a cada una de las posibilidades a elegir (en nuestro caso cada posible adaptación) produzca una clasificación donde la elección más favorable sea la primera, y la menos favorable la última. El aprendizaje Bayesiano es utilizado ampliamente en los filtros anti-spam de los servidores de correo con bastante éxito.

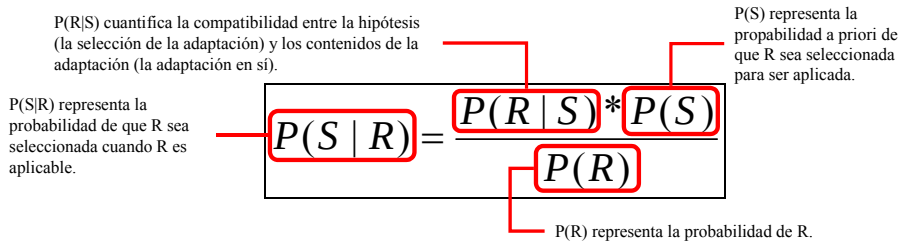


Figura 5.20 Fórmula para el aprendizaje Bayesiano propuesto.

La fórmula que es aplicada sobre cada una de las posibles elecciones (hipótesis) es mostrada en la figura 5.20.

A continuación se describe la manera en que cada una de las probabilidades necesarias para la aplicación del aprendizaje Bayesiano es calculada.

$$P(R) = \frac{1}{número\_de\_reglas\_aplicables}$$

Donde *número_de_reglas_aplicables* representa el número de reglas de adaptaciones entre las que el sistema debe elegir.

$$P(S) = \frac{número\_de\_planes\_positivos}{número\_de\_planes\_positivos + número\_de\_planes\_negativos}$$

P(S) refleja la información obtenida a lo largo del tiempo durante la aplicación de las reglas, es decir, la experiencia acumulada en la aplicación de las reglas. Se calcula usando el número de reglas de adaptación aplicadas de forma exitosa (*número_de_planes_positivos*) y el número de reglas donde el usuario no estuvo de acuerdo (*número_de_planes_negativos*) y deshizo la adaptación.

$$P(R \mid S) = (c\_migración - beneficio) * \frac{número\_veces\_R\_ha\_sido\_aplicado}{número\_planes\_aplicables}$$

P(R|S) estima cuán compatible es una adaptación con la selección actual, es decir, evalúa la bondad de una adaptación para la selección actual. En nuestro caso dicha bondad se evalúa en función del coste de migración, el beneficio de adaptación, la cantidad de veces que la adaptación (R) ha sido aplicada exitosamente (*número_veces_R_ha_sido_aplicado*), y finalmente, el número de reglas aplicables entre las que hay que elegir (*número_planes_aplicables*), tal y como se muestra en la fórmula anterior.

A partir de la clasificación resultante de la aplicación de la fórmula sobre cada una de adaptaciones posibles, el sistema intentará ejecutar la primera, en el caso en que su aplicación viole el compromiso de usabilidad, elegirá la siguiente en la clasificación, e intentará aplicarla siguiendo el mismo procedimiento. El proceso se repetirá hasta que no queden reglas elegibles en la clasificación o una regla sea aplicada y cumpla el compromiso de usabilidad.

5.2.6.2 Diseño de la etapa de decisión en el sistema multi-agente

La etapa de decisión es llevada a cabo íntegramente por el agente *AgentAdaptationProcess*. Cuando en evento indicando un cambio en el contexto es recibido, una serie de planes serán aplicables dado el contexto actual (véase la sección 5.2.5). Para seleccionar cuál es la adaptación más apropiada para la situación actual el agente *AgentAdaptationProcess* realiza un proceso de meta-razonamiento para decidir qué plan (regla de adaptación aplicar). Durante dicho razonamiento, se realiza una evaluación del coste de migración y el beneficio de adaptación de cada uno de los planes aplicables, y finalmente se crea una clasificación de acuerdo al sistema de selección de reglas Bayesiano descrito anteriormente.

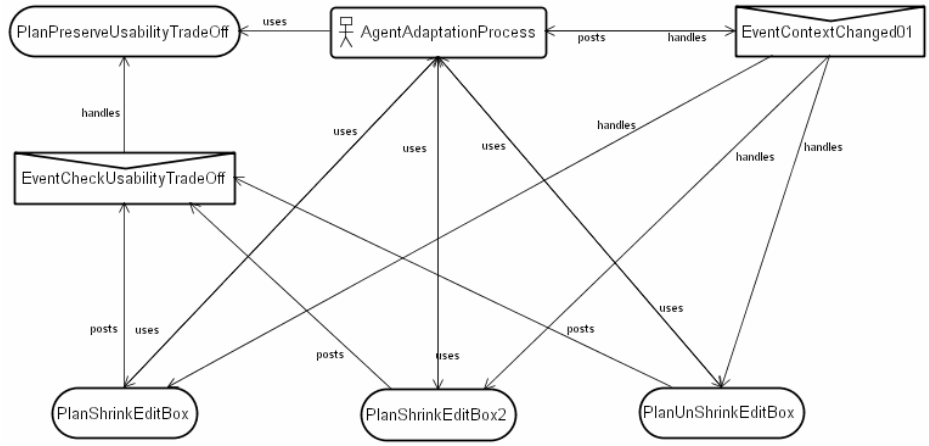


Figura 5.21 Diseño del sistema multi-agente para la etapa de decisión de adaptación.

5.2.7 Etapa de ejecución de adaptaciones

La etapa de ejecución es la última en la aplicación de una adaptación. Durante dicha etapa el sistema aplica la transformación asociada a la regla de adaptación seleccionada en la etapa anterior. Para ello, es necesario obtener una versión actualizada de la interfaz de usuario sobre la que aplicar la adaptación, aplicar la transformación, y facilitar al usuario la reanudación de sus tareas sobre la interfaz de usuario adaptada.

Para obtener una versión actualizada de la interfaz de usuario el sistema debe ser capaz de capturar el estado actual de la interfaz con la que el usuario está interactuando. En el modelo propuesto, el estado actual de la interfaz es actualizado a través de los sensores, por defecto incluidos en cualquier plataforma cliente que acceda la sistema, y que envían un historial de las acciones que el usuario realiza con el sistema (escribir texto, seleccionar elementos, pulsar botones, etc). Dicha información permite al sistema multi-agente ir actualizando constantemente el modelo de interfaz de usuario para reflejar las operaciones que sobre él realiza el usuario.

Una vez se ha obtenido una versión actualizada del modelo de interfaz de usuario concreta, se pueden aplicar las transformaciones asociadas a la regla seleccionada. Para ello, se pueden seguir distintas aproximaciones de transformación: transformación de grafos, transformación algebraica, XSLT, por citar algunos ejemplos. En el sistema diseñado se ha utilizado la transformación de grafos, tal y como se describe en [Lim04]. En la

sección 5.3 se describe la solución tecnológica adoptada para tal propósito.

Finalmente, es necesario mostrar al usuario la interfaz de usuario adaptada. Es por lo tanto necesario transformar el modelo de interfaz de usuario concreta resultante de la transformación en una interfaz de usuario ejecutable, siendo para ello necesaria la utilización de *renderers* (véase la sección 5.3). Un punto importante en este proceso de renderización es permitir al usuario retomar la interacción con la interfaz de usuario en el mismo estado donde fue interrumpida para la aplicación de la adaptación. Para ello, a través de los sensores de la plataforma cliente por defecto, siempre se mantiene la pista sobre cuál es el elemento que tiene el foco de entrada del usuario en cada momento. El elemento que tiene el foco en cada momento es almacenado en el modelo de interfaz de usuario concreto, de forma que el *renderer* añade el código a la interfaz de usuario adaptada para conseguir que el elemento que tenía el foco antes de la adaptación lo recupere. Para ello, ha sido necesario incorporar un nuevo atributo a todos los objetos concretos de interacción de usiXML que permita almacenar si tiene el foco o no. El atributo ha sido denominado *isFocused*. En aquellos casos en que el elemento que tenía el foco sea eliminado por una adaptación, se asignará el foco a uno de los elementos que sustituyan al eliminado, y en aquellos casos en que esto no sea posible se asignará al primer elemento de entrada de la interfaz de usuario, aunque esta última opción no es deseable debido al efecto de discontinuidad que produce.

5.2.7.1 Diseño de la etapa de ejecución en el sistema multi-agente

La etapa de ejecución de adaptación es realizada por el agente *AgentAdaptationProcess*. Una vez seleccionada la adaptación (plan) que se debe aplicar, el agente comenzará a ejecutar dicha adaptación. Inicialmente, se obtendrá una versión actualizada de la interfaz de usuario sobre la que aplicar la adaptación a través del plan *PlanGetUserInterface*. A continuación se aplicará la adaptación real sobre la interfaz de usuario siguiendo una aproximación basada en la transformación de grafos [Lim04].

Una vez aplicada la adaptación el motor debe comprobar que dicha adaptación cumple con el compromiso de usabilidad establecido para la plataforma actual, y para ello hará uso del plan *PlanPreserveUsabilityTradeOff*, el cual evaluará cada uno de los criterios establecidos en el diseño del compromiso de usabilidad, teniendo en cuenta la relaciones entre los

criterios establecidas por le diseñador (véase la sección 4.2.2.3). Si se cumple el compromiso de usabilidad establecido, el sistema procederá al siguiente paso, mientas que si no se preserva el compromiso de usabilidad establecido se descartará la regla aplicada y se intentará aplicar la siguiente regla en la clasificación creada en la etapa anterior.

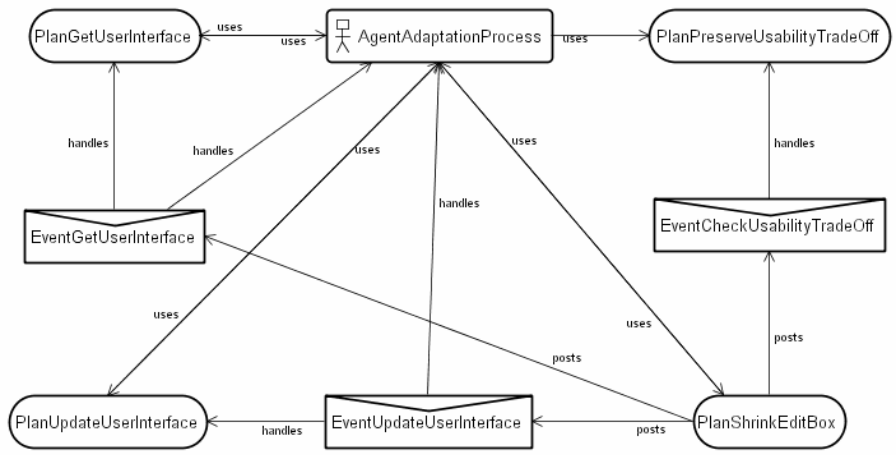


Figura 5.22 Diseño del sistema multi-agente para la etapa de ejecución de adaptación.

Si la adaptación ha preservado el compromiso de usabilidad, el sistema crea una nueva interfaz de usuario ejecutable a partir del modelo de interfaz de usuario concreta resultante de la aplicación de la adaptación, usando para ello el *renderer* apropiado para la plataforma actual.

Evaluación del compromiso de usabilidad

La evaluación del compromiso de usabilidad conlleva la valoración de los criterios de usabilidad y sus contribuciones a la calidad del proceso de adaptación, tal y como se haya definido en la fase de análisis.

Para la evaluación de los criterios y subcriterios implicados en el compromiso de usabilidad es necesario definir métricas que aporten una visión cuantitativa sobre en qué medida una adaptación mantiene cada uno de los criterios. Para ellos se usan métricas como las propuestas por Constantine en [Con99], modelos de evaluación, como el propuesto en el apartado 5.2.6.1 para la visibilidad y la continuidad, y restricciones atómicas definidas sobre el grafo usado para la representación de la interfaz de usuario.

Restricciones atómicas

Las restricciones atómicas son restricciones definidas sobre un grafo y de las cuales puede determinarse si se dan o no en un grafo determinado. Las restricciones atómicas constan de una parte izquierda donde se especifica el patrón que se debe emparejar con el grafo origen, y una o varias conclusiones que especifican una condición que debe darse cada vez que se encuentre un emparejamiento con el grafo parcial especificado en la parte izquierda. Adicionalmente se pueden añadir condiciones sobre las variables usadas en la especificación de la parte izquierda y las conclusiones de la restricción atómica.

Por ejemplo, en la figura 5.23 se muestra una restricción atómica donde se expresa que todas las imágenes (elementos de tipo *imageComponent)* que se encuentren en el grafo que representa la interfaz de usuario deben tener asociado un texto alternativo para la imagen, tal y como se expresa en el estándar de accesibilidad WAI[29].

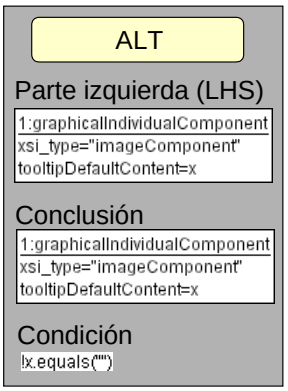


Figura 5.23 Ejemplo de restricción atómica.

Valoración de los criterios del compromiso de usabilidad

Los criterios de usabilidad implicados en el compromiso de usabilidad se valoran de acuerdo a la contribución que dichos criterios tienen en la calidad de la adaptación, especificado mediante las relaciones de contribución GRL (véase la sección 2.3.3.3) del modelo de compromiso de usabilidad descrito en la etapa de análisis propuesta (véase la sección 4.2.2.3). La evaluación de cada uno de los criterios se hará comparando la

[29] http://www.w3c.org/WAI/

interfaz de usuario original con la interfaz de usuario que produciría la adaptación que se desea evaluar.

Las relaciones de correlación pueden ser de distintos tipos haciendo que la evaluación del criterio de usabilidad involucrado sea distinta. En la tabla 11 se puede encontrar un resumen que muestra cómo se evalúan los criterios de acuerdo al tipo de relación de contribución especificado en el compromiso de usabilidad.

Si la evaluación de cada uno de los criterios de usabilidad involucrados en el compromiso de usabilidad es positiva, de acuerdo al modelo de evaluación descrito en la tabla 11, entonces se considerará que la adaptación aplicada preserva el compromiso de usabilidad especificado en tiempo de diseño, y por lo tanto la interfaz de usuario generada será mostrada al usuario.

Tabla 11. Tipos de relaciones de contribución en el compromiso de usabilidad.

Nombre	Contribución
AND	La contribución de cada uno de los criterios es positiva y necesaria.
OR	La contribución de cada uno de los criterios es positiva y suficiente.
MAKE	La contribución del criterio es positiva y suficiente. La evaluación del criterio para la interfaz de usuario adaptada debe ser al menos igual a la evaluación de dicho criterio en la interfaz de usuario original.
BREAK	La contribución del criterio es negativa y suficiente. Si la evaluación del criterio es mayor en la interfaz de usuario adaptada que en la original (es decir, es positiva), automáticamente se descarta la regla de adaptación evaluada.
HELP	La contribución del criterio es positiva pero insuficiente. La evaluación del criterio para la interfaz de usuario adaptada debe ser al menos igual al 25% de la evaluación del criterio para la interfaz de usuario original.
SOME+	La contribución del criterio es positiva y su aportación es media. La evaluación del criterio para la interfaz de usuario adaptada debe ser al menos igual al 50% de la evaluación del criterio para la interfaz de usuario original.

5.3 Implementación del sistema multi-agente

Hasta el momento se ha descrito la arquitectura multi-agente propuesta desde un punto de vista general. Sin embargo, su implementación requiere un diseño más detallado, donde la interfaz de comunicación entre el sistema multi-agente y la plataforma cliente que accede a las facilidades de adaptación suministradas por el sistema dependen del lenguaje destino para el cual se deba renderizar la interfaz de usuario.

La arquitectura propuesta ha sido planteada para maximizar su uso por distintas plataformas. Se plantea una arquitectura cliente/servidor donde cualquier plataforma cliente con capacidades de conexión vía TCP/IP, y con soporte para uno lenguajes destino actualmente soportados (XUL o Java) será capaz de utilizar una interfaz ejecutada en la arquitectura propuesta y aprovechar sus capacidades de adaptación.

El sistema multi-agente ha sido programado en el lenguaje de programación de agentes Jack [Bus99]. La plataforma permite la programación de sistemas multi-agente utilizando unos constructores equivalentes a los usados durante le diseño detallado de un sistema multi-agente siguiendo la metodología *Prometheus*. La plataforma de Jack incluye además un entorno que permite crear visualmente el esqueleto de los agentes, sus planes, creencias, eventos y capacidades. Jack proporciona además la posibilidad de encapsular el sistema multi-agente dentro de un Java *servlet*[30], permitiendo al sistema multi-agente creado ser accesible a través del protocolo HTTP.

5.3.1 Implementación del sistema multi-agente para *plataformas XUL*

A continuación se describe un caso concreto de la aplicación de la arquitectura para la adaptación de interfaces de usuario basadas en el lenguaje XUL.

5.3.1.1 XUL: Un lenguaje de especificación de interfaces de usuario

El lenguaje XUL es el lenguaje multi-plataforma que permite especificar la interfaz de usuario usando para ello una sintaxis basada en XML con un nivel de abstracción muy cercano al nivel concreto de interfaces de usuario propuesto para usiXML.

Una de las grandes ventajas de XUL es que puede ser ejecutado en cualquier navegador basado en el motor *Gecko* del proyecto *Mozilla*. Entre dichos navegadores se encuentran: *Netscape 6* en adelante, *Mozilla 6* en adelante, *Camino*, *Firefox*, *IBM Web Browser* o *Galeon*, entre otros. Ello permite que una interfaz de usuario diseñada en XUL pueda ser ejecutada en casi cualquier plataforma, desde un PC con el sistema operativo Microsoft Windows (*Netscape*, *Mozilla*, *Firefox*), con el sistema operativo Linux (*Netscape*, *Mozilla*, *Firefox*, *Galeón*), con el sistema operativo OS/2 de IBM (*IBM Web Browser*) a un Apple ejecutando Mac Os (*Camino*), una

[30] http://java.sun.com/products/servlet/

máquina Sun con el sistema operativo Solaris (*Netscape*), e incluso dispositivos móviles ejecutando la distribución de Linux Familiar y la versión para dicha distribución de Mozilla: *Minimo*.

XPToolKit: Desarrollo multi-plataforma en XUL

El lenguaje XUL se haya englobado dentro de un conjunto de facilidades que se agrupan dentro del *XPToolKit*[31]. Dentro de *XPToolKit* se proporcionan las herramientas, basadas en los estándares más extendidos para la creación de aplicaciones basadas en tecnología Web (JavaScript, CSS, SOAP, DOM, RDF, etc), necesarias para la creación de aplicaciones independientes, pero que se ejecutan utilizando el motor del navegador *Mozilla*.

La arquitectura de XPToolKit permite la especificación de la interfaz de usuario utilizando el lenguaje basado en XML XUL, definiendo el aspecto de los componentes a través de la utilización de estilos, normalmente aplicándolo mediante hojas de estilo en cascada (CSS[32]). El comportamiento de los elementos es creado usando JavaScript[33], el cual permite acceder a la estructura de la interfaz de usuario representada en forma de árbol DOM[34] (Document Object Model), tal y como ocurre habitualmente en el desarrollo de aplicaciones Web, y manipular la interfaz. Para aquellas ocasiones en que sea necesario invocar funcionalidades no escritas en JavaScript *XPToolkit* suministra una serie de objetos XPCOM. Estos objetos son una versión multiplataforma de los conocidos objetos COM de Microsoft y proporcionan independencia respecto del sistema operativo concreto. Ahora bien, para poder utilizar estos objetos desde JavaScript, es necesario un lenguaje intermedio de conexión: XPConnect (Cross-Platform Connect) y otro que permita definir API's adecuadas para los objetos XPCOM. Este último es el XPIDL (Cross-Platform Interface Definition Language), el cual permite describir la signatura de los objetos de una manera independiente de la plataforma (de una forma análoga a lo que sucede con IDL en el modelo de CORBA).

[31] http://www.mozilla.org/xpfe/xptoolkit/
[32] http://www.w3.org/Style/CSS/
[33] http://www.ecma-international.org/publications/standards/Ecma-262.htm
[34] http://www.w3.org/DOM/

Renderización de interfaces de usuario en XUL

Dentro de la arquitectura multi-agente propuesta, la interfaz de usuario se encuentra descrita utilizando el modelo de interfaz de usuario concreta propuesta en usiXML. Por lo tanto para la visualización de la interfaz es necesario traducir cada uno de los elementos del modelo de interfaz de usuario concreta a elementos de XUL. Sin embargo, internamente la interfaz de usuario no es representada en el formato XML propuesto en usiXML, sino en forma de un grafo dirigido y etiquetado equivalente, tal y como se describe en la sección 3.4.1.4. Dicha representación en forma de grafo permite la aplicación de las adaptaciones (transformaciones de la interfaz de usuario) utilizando reglas de transformación de grafos.

Para la manipulación de la especificación de la interfaz de usuario se ha creado la herramienta *usiXMLTransformer*, la cual es capaz de transformar una especificación de una interfaz de usuario en una representación en forma de grafo equivalente, aplicar cualquier adaptación de la interfaz de usuario descrita en forma de regla de transformación, y finalmente convertir la especificación de la interfaz de usuario resultante en una interfaz de usuario ejecutable directamente en el lenguaje XUL o el lenguaje Java. La figura 5.24 ilustra el funcionamiento de *usiXMLTransformer*, donde se observa el paso desde la especificación basada en XML de usiXML a una representación basada en grafos. Dicha representación en forma de grafo podrá o no ser transformada para reflejar un proceso de adaptación. En cualquier caso, dicha representación podrá ser traducida tanto al lenguaje XUL como a Java para su presentación al usuario.

La transformación del grafo que representa la interfaz de usuario se realiza a través de la API proporcionada por la herramienta AGG[35] (*Attributed Graph Grammars*), la cual permite la transformación de un grafo siguiendo la aproximación de *single push-out*.

usiXMLTransformer, proporciona también un API que permite su funcionamiento de forma totalmente autónoma, sin necesidad del resto de los componentes que componen la arquitectura multi-agente propuesta.

[35] http://tfs.cs.tu-berlin.de/agg/

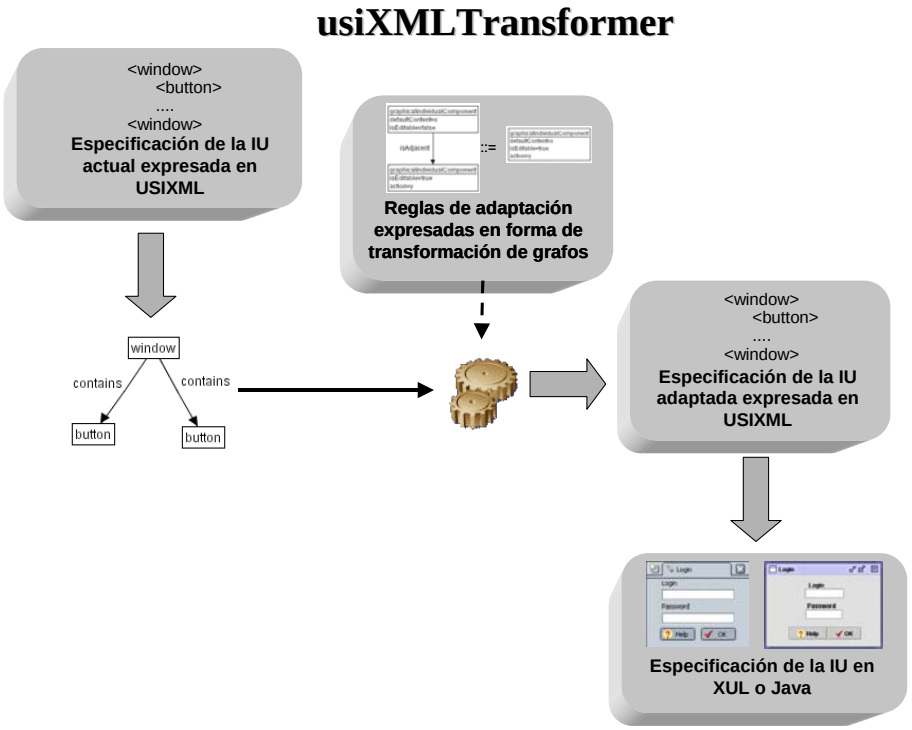


Figura 5.24 Vista general del funcionamiento de usiXMLTransformer.

5.3.1.2 Interfaz del sistema multi-agente con las *plataformas XUL*

La comunicación con una plataforma capaz de ejecutar una interfaz especificada utilizando el lenguaje XUL se realiza a través del protocolo HTTP, mediante el encapsulamiento del sistema multi-agente dentro de un *servlet*. Un *servlet* permite la ejecución de una aplicación Java dentro de un servidor Web, normalmente el servidor Tomcat. Las funcionalidades que la aplicación Java de servidor ofrece pueden ser accedidas mediante páginas Java de servidor (JSP – *Java Server Pages*). Dichas páginas permiten crear dinámicamente el contenido de las respuestas a peticiones HTTP, habitualmente para la generación dinámicas de páginas Web usando bases de datos, aunque en nuestro caso nos servirán como medio para recibir y enviar datos codificados utilizando un formato basado XML.

Cualquier plataforma XUL que desee aprovechar las capacidades de adaptación del sistema propuesto sólo necesitará conocer la interfaz proporcionada a través del *servlet* y la dirección IP o nombre del servidor que alberga el *servlet*. Dichas capacidades podrán ser accedidas tanto de

forma local como de forma remota a través de TCP/IP y el protocolo HTTP.

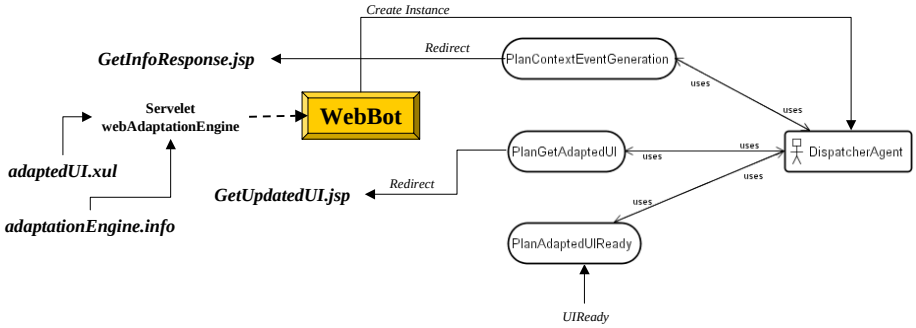


Figura 5.25 Interfaz del sistema multi-agente con las plataformas XUL.

La interfaz que el sistema multi-agente ofrece a las plataformas XUL es la siguiente:

- *adaptationEngine.info*: las plataformas XUL que deseen comunicar los datos recogidos por sus sensores podrán hacerlo a través de dicha dirección utilizando el método POST de HTTP. La información enviada por los sensores debe estar estructurada siguiendo el esquema descrito en la sección 5.2.4.2.

- *adaptedUI.xul* : cuando una plataforma XUL envía sus datos sobre el contexto de uso actual a través de *adaptationEngine.info*, el agente *DispatcherAgent* utiliza su plan *PlanContextEventGeneration* para tratar la información que llega del contexto. Dicho plan será también el encargado de responderle a la plataforma cliente, indicándole si existe una nueva interfaz de usuario adaptada o no. En caso afirmativo, la plataforma cliente la obtendrá a través de *adaptedUI.xul*. Una petición en esa dirección provocará la ejecución del plan *PlanGetAdaptedUI*, el cual elaborará la nueva interfaz de usuario adaptada haciendo uso de la página de servidor *GetUpdatedUI.jsp*.

A través de esta simple interfaz se acceden a las facilidades de adaptación, de forma tanto local como a través de Internet, lo cual posibilita la colaboración del sistema con otras tecnologías basadas en la Red como los servicios Web para proporcionar cualquier tipo de funcionalidad extra necesaria.

5.4 Conclusiones del capítulo

A lo largo de este capítulo se ha descrito una arquitectura que permite ofrecer al usuario una serie de capacidades de adaptación que son diseñadas siguiendo el método propuesto en el capítulo 4. La arquitectura propuesta se basa en el concepto de agente, lo cual facilita la implementación de los modelos de toma de decisiones necesarios para aplicar en cada momento las adaptaciones más apropiadas. De igual menera dentro de este capítulo se describe detalladamente cómo se lleva a cabo cada una de las etapas del proceso de adaptación dentro del sistema multi-agente.

CAPÍTULO 6
CASOS DE ESTUDIO

"Son vanas y están plagadas de errores las ciencias que no han nacido del experimento, madre de toda certidumbre."
(Leonardo Da Vinci)

6.1 Introducción

A lo largo de este capítulo se presentan ejemplos de aplicación de AB-UIDE en el desarrollo de aplicaciones. AB-UIDE es un método que extiende las habituales aproximaciones basadas en modelos para el desarrollo de interfaces de usuario, para permitir el modelado de las capacidades de adaptación de las interfaces de usuario. Para ello, se describen los modelos creados durante el proceso de desarrollo descrito en el capítulo 4 y distintos ejemplos de la ejecución de las aplicaciones adaptativas creadas dentro de la arquitectura multi-agente para la ejecución de interfaces de usuario adaptativas descrita en detalle en el capítulo 5.

Uno de los campos donde más acogida han tenido los sistemas adaptativos es en las aplicaciones que integran algún tipo de sistema sensible al contexto de asistencia o sugerencia al usuario. En este sentido, el primer ejemplo presentado describe el desarrollo de una posible interfaz de usuario para un coche, la cual asiste al usuario durante la conducción de un coche, basándose para ello en la información recogida del entorno en que el usuario/conductor conduce su vehículo. El segundo ejemplo presentado describe la interfaz de usuario para un cajero automático, y ejemplifica la especificación de una interfaz de usuario adaptativa capaz de acomodarse a distintas plataformas, usuario y entornos cambiando su aspecto, su navegación y sus contenidos.

6.2 Caso de estudio: Car UI

El primer caso de estudio presentado describe el diseño e implementación de una interfaz de usuario para vehículos, denominada Car UI, que asiste al conductor durante la conducción del vehículo evitando posibles

situaciones de peligro y haciendo más cómoda y segura la conducción. El sistema toma los datos sobre el estado del entorno de uso para proporcionar al usuario la asistencia apropiada en cada situación.

A continuación se describen cada uno de los modelos creados durante el desarrollo de la aplicación del método propuesto para el desarrollo de interfaces de usuario adaptativas: AB-UIDE.

6.2.1 Adquisición de requisitos

La etapa de adquisición de requisitos comienza con la descripción de las tareas que el usuario, conductor *(driver)* en este caso, debe poder realizar a través de la interfaz de usuario. La adquisición de requisitos es completada con la descripción de los modelos de usuario, plataforma y entorno.

6.2.1.1 Casos de uso

En la figura 6.1 se muestra el diagrama de casos de uso asociado a Car UI.

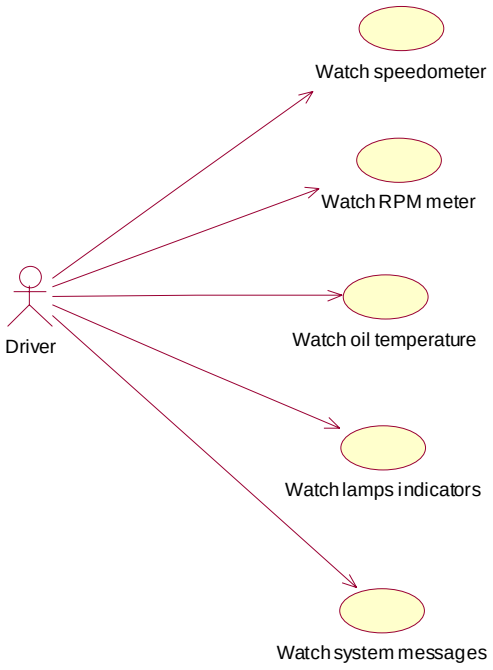


Figura 6.1 Diagrama de casos de uso para Car UI.

En él se muestra como el usuario puede observar el velocímetro *(watch speedometer)*, observar el contador de revoluciones *(watch RPM meter)*, observar la temperatura del aceite *(watch oil temperature)*, observar los distintos indicadores de las luces *(watch lamps indicators)* y observar los mensajes que el sistema le muestra al conductor con advertencias o sugerencias *(watch system messages)*.

El diagrama de casos de uso muestra de manera estática las tareas que el usuario puede realizar a través de la interfaz de usuario, pero no especifica las restricciones temporales entre dichas tareas. Las restricciones temporales son especificadas mediante diagramas de secuencia de UML. La figura 6.2 muestra el diagrama de secuencia para las tareas de la figura 6.1, donde se observa que en este caso todas tareas pueden realizarse de forma concurrente.

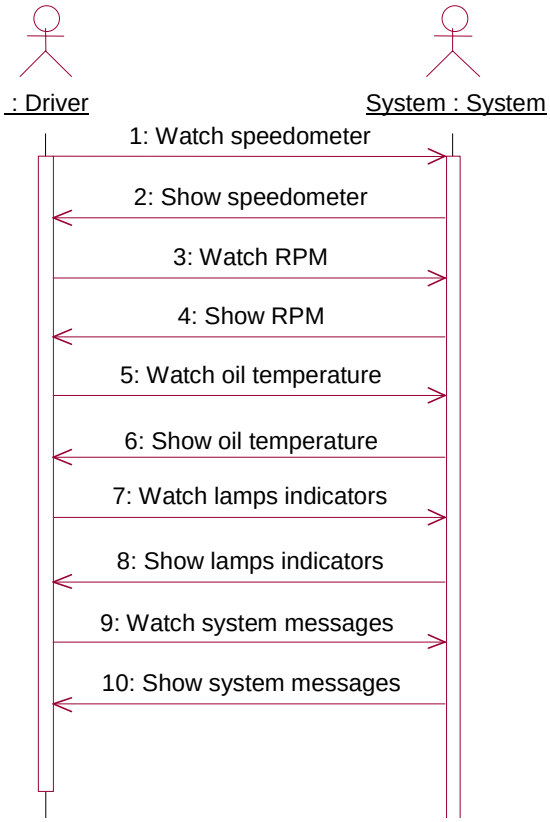


Figura 6.2 Diagrama de secuencia para los casos de uso de Car UI.

Los diagramas de casos de uso y secuencia serán en la etapa de análisis refinados para crear el modelo de tareas de la aplicación.

6.2.1.2 Modelos de usuario, plataforma y entorno

En un sistema adaptativo es imprescindible describir el entorno donde se llevará a cabo la interacción, y al que por lo tanto el sistema tendrá que ser capaz de adaptarse. El entorno en AB-UIDE se define como una cuádrupla formada por el modelo de usuario, el modelo de entorno, el modelo de usuario, y la tarea actual. El diseñador podrá especificar tanto modelos de plataforma, usuario y entorno como sean necesarios para describir los distintos tipos de entornos donde potencialmente se ejecutará la aplicación. La figura 6.3 ilustra la descripción del contexto propuesta para el ejemplo de Car UI. Nótese que la tarea actual no es descrita en este momento, ya que es una componente del contexto obtenida en tiempo de ejecución, tal y como se describe en la sección 5.2.4.1.

Platform	Car
platformId	*0021P*
platformName	*CarPlatform*
isColorCapable	*true*
isImageCapable	*true*
numberOfColors	*256*
screenSize	*542x128*
isSoundOutputCapable	*true*

Environment	Outdoor
environmentId	*0021E*
environmentName	*CarEnvironment*
lightingLevel	*average*
isNoisy	*true*
weatherConditions	*sunny*
roadType	*highway*
roadBending	*0*

User	Driver
userId	*0021U*
minutesDriving	*0*
drivingExpertise	*rookie*
Relationships	
cognitiveFocus.CarPlarform	*low*

Figura 6.3 Modelos de usuario, plataforma y entorno para Car UI..

En este caso la plataforma donde se ejecutará la interfaz de usuario será un coche *(car)*. El panel de conducción donde se mostrará la interfaz de usuario es en color y puede mostrar hasta 256 colores de forma simultánea. Su resolución es de 542 por 128, y es capaz de reproducir sonidos.

Los factores del entorno que afectarán a la interfaz son principalmente del exterior del coche. Por defecto, se ha considerado que el nivel de iluminación será medio, la interacción se realiza en un entorno ruidoso, tiempo soleado, en una autopista y sin curvas.

El usuario por defecto se ha considerado novato. También se consideran los minutos que el usuario lleva conduciendo (por el posible cansancio acumulado). Se incluye una relación que expresa cómo es una característica del usuario en relación con otro componente del contexto (en este caso a la plataforma *CarPlatform)*. Se especifica que el nivel de atención del usuario mientras utiliza la plataforma *CarPlatform* será bajo.

6.2.2 Análisis

La etapa de análisis transforma los requisitos, y los complementa, creando una descripción del sistema más cercano a la semántica del sistema. En AB-UIDE el análisis incluye la descripción de los objetos que manipula la interfaz de usuario mediante el modelo de dominio, la descripción de los posibles grupos de usuario que interactúan con el sistema (en Car UI sólo se ha considerado un tipo de usuario) y el compromiso de usabilidad que debe mantener cualquier adaptación aplicada.

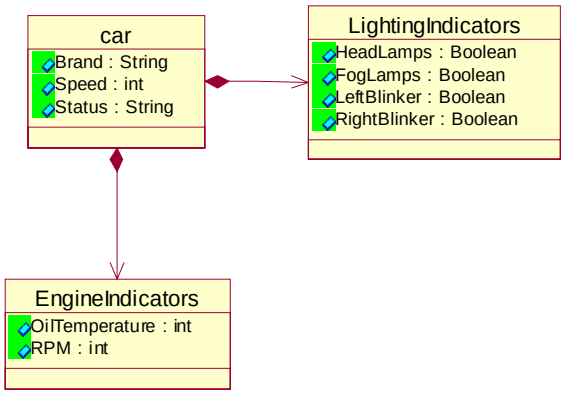


Figura 6.4 Modelo de dominio para Car UI.

6.2.2.1 Modelo de dominio

El modelo de dominio para Car UI se muestra en la figura. En el modelo de dominio sólo se muestra aquella información que es relevante para la

interfaz de usuario. En este caso se ha descrito el coche, el cual contiene tanto los indicadores de las luces *(Lighting indicators)* como indicadores sobre el estado del motor *(EngineIndicators)*.

6.2.2.2 Compromiso de usabilidad

El compromiso de usabilidad especifica un modelo de calidad, especificado por el diseñador, que debe respetar cualquier adaptación que se aplique sobre la interfaz de usuario, permitiendo de esta manera controlar cómo evolucionan ciertos criterios de usabilidad durante la adaptación de la interfaz de usuario (véase la sección 4.2.2.3).

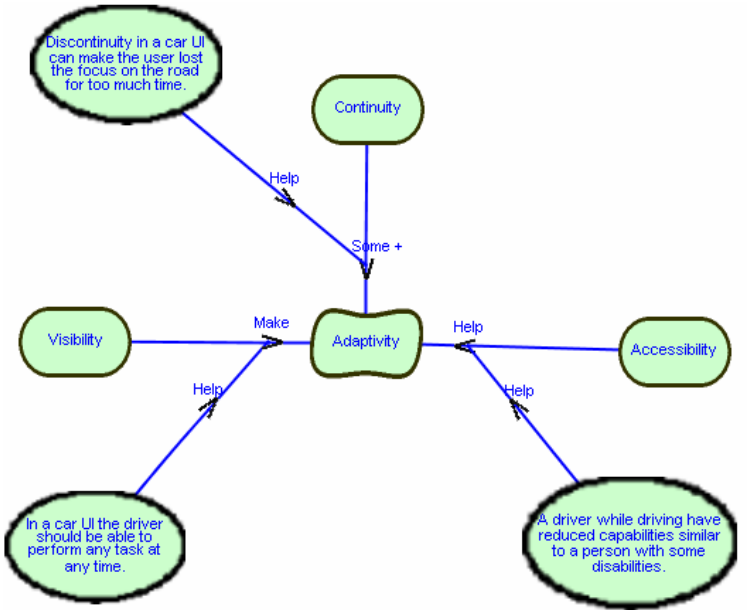


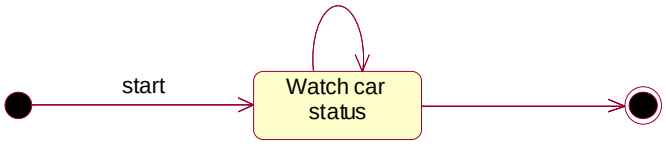
Figura 6.5 Compromiso de usabilidad para Car UI.

La figura 6.5 muestra el compromiso de usabilidad propuesto para la interfaz de Car UI. En dicha figura se observa como cualquier adaptación debe maximizar *(make)* la visibilidad, ya que el conductor debe ser capaz en todo momento de realizar cualquiera de las tareas que ofrece la interfaz de Car UI. En importante mantener la continuidad en las adaptaciones debido a que un conductor no mantiene su atención constantemente sobre la interfaz de usuario, sino que centra su atención en la carretera. Por lo tanto, aquellas adaptaciones que introduzcan un grado medio de discontinuidad harán que el conductor tenga que emplear un cierto tiempo en reconocer dónde están los elementos de la interfaz, tiempo que

debería ser minimizado para que el conductor pueda centrar su atención en la carretera. Finalmente, es aconsejable que cualquier adaptación mantenga la accesibilidad del sistema, ya que mientras un conductor centra su atención en la carretera, sus capacidades quedan mermadas de cara a la utilización de la interfaz de usuario, reduciendo por ejemplo su capacidad de visión al tener que mirar a menudo la interfaz de reojo. Ello hace que sea aconsejable que se ofrezca un alto grado de accesibilidad en el sistema, por ejemplo usando dos modalidades (gráfica o textual y vocal) para comunicar al usuario los mensajes importantes.

6.2.3 Diseño

Durante la etapa de diseño se debe crear una descripción de la interfaz de usuario con el suficiente detalle para permitir su implementación. En AB-UIDE la etapa de diseño consiste en la creación de un modelo de tareas que refleje de forma detallada las tareas que se van a ofrecer al usuario y sus relaciones temporales, la asociación de cada tarea con los elementos del modelo de dominio que manipula, la derivación a una interfaz abstracta, la transformación de dicha interfaz abstracta en una interfaz de usuario concreta, y finalmente, la especificación de las reglas de adaptación que permitirán ofrecer al usuario las capacidades de adaptación.



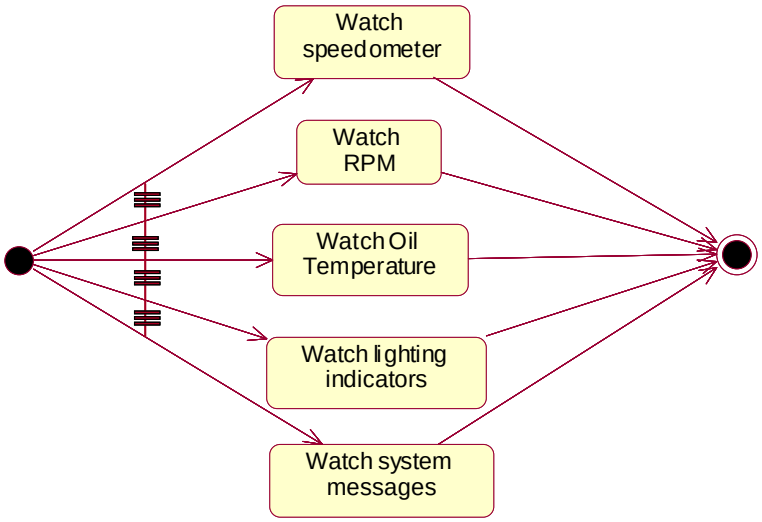
Task name	Watch car status
Description	The car driver watches the car status (speed, rpm, lighting, etc).
Type	Abstract
Frecuency	*High*
Precondition	NULL
Postcondition	NULL
Presentation	FreeContainer

Figura 6.6 Tarea raíz para Car UI.

6.2.3.1 Modelo de tareas

El modelo de tareas describe de forma detallada las tareas que el usuario será capaz de realizar a través de la futura interfaz de usuario.

La figura 6.6 muestra la tarea raíz de Car UI. El conductor observa el estado del coche. La tarea puede ser repetida tantas veces como sea necesario (lo cual es indicado mediante la transición del estado *Watch car status* a sí mismo. El comienzo de la interacción es marcado con la etiqueta *start* desde el estado inicial. La transición al estado final no está etiquetada, ya que no será el usuario el que haga que la tarea termine y se tome esa transición, sino el sistema cuando se quite la llave del contacto del coche. La tarea es de tipo abstracto, tendrá una frecuencia de realización alta y será presentada en un *FreeContainer*, es decir, será presentada en una ventana principal (raíz).

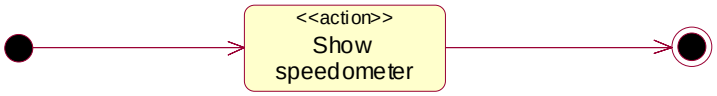


Task name	Watch speedometer
Description	The driver can watch the speed that the car is going at.
Type	Abstract
Frecuency	*High*
Precondition	NULL
Postcondition	NULL
Presentation	Container

Task name	Watch RPM
Description	The driver can watch the rpm that the car's engine is going at.
Type	Abstract
Frecuency	*Average*
Precondition	NULL
Postcondition	NULL
Presentation	Container

Figura 6.7 Refinamiento de la tarea de la figura 6.6.

La tarea abstracta descrita en la figura 6.6 se muestra refinada en la figura 6.7. Donde se observa como el conductor puede observar la velocidad actual *(Watch speedometer)*, observar las revoluciones del motor en cada momento *(Watch RPM)*, observar la temperatura del aceite *(Watch Oil Temperature)*, observar los indicadores de las luces *(Watch lighting indicators)* y observar los mensajes del sistema *(Watch system messages)*. Debe destacarse que las cinco tareas podrán realizarse de forma concurrente (como indica el operador |||). La figura también muestra las propiedades de dos de las cinco tareas mostradas (el resto se describirían de forma análoga), donde se puede observar como las tareas serán presentadas en *Containers*, es decir, no serán presentadas en una ventana a parte, sino dentro del contenedor en el que esté incluida la tarea a la que refinan (*Watch car status)*.



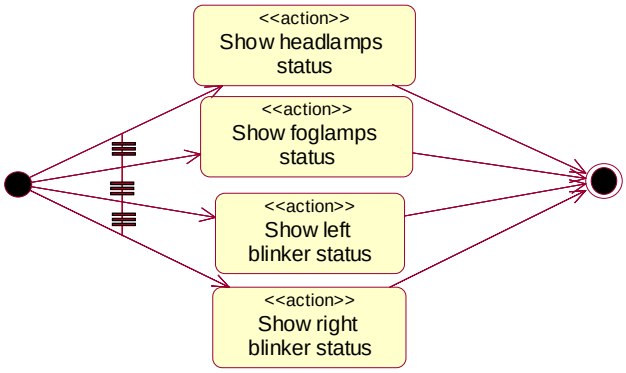
Action name	Show speedometer
Description	The system displays the current speed.
Type	*Output*
Frecuency	*High*

Figura 6.8 Refinamiento de la tarea *Watch speedometer* de la figura 6.7.

La tarea "Observar velocímetro" *(Watch speedometer)* implica que el usuario debe ver la velocidad actual, y por lo tanto, el sistema debe mostrar al usuario dicha velocidad. Por ello, la tarea *Watch speedometer* ha sido refinada, para reflejar ese requisito, con la especificación de la tarea "Mostar velocímetro" *(Show speedometer)* (véase la figura 6.8). En este caso *Show speedometer* ya es una acción, y por lo tanto habrá que describir su tipo, salida *(output)* ya que el sistema debe mostrar la velocidad actual, y frecuencia, alta *(high)* porque será una tarea ejecutada muy a menudo. Las tareas *Watch RPM*, *Watch Oil Temperature* y *Watch system messages* serían especificadas de forma totalmente análoga. La tarea *Watch lighting indicators* sin embargo, ha sido descompuesta en distintas acciones, tal y como se muestra en la figura 6.9.

La tarea *Watch lighting indicators* ha sido descompuesta en las acciones "Mostrar luces delanteras" *(Show headlamps)*, "Mostrar luces antiniebla" *(Show foglamps)*, "Mostrar intermitente izquierdo" *(Show left blinker)* y

"Mostrar intermitente derecho" *(Show right blinker)* (véase la figura 6.9). Las acciones pueden ser realizadas de forma concurrente, tal como indica el operador ||| que une las acciones.



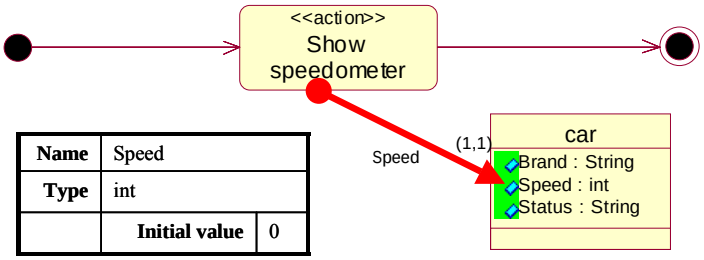
Action name	Show headlamps status
Description	The system displays current headlamps status (on/off).
Type	*Output*
Frecuency	*Low*

Action name	Show foglamps status
Description	The system displays current foglamps status (on/off).
Type	*Output*
Frecuency	*Low*

Figura 6.9 Refinamiento de la tarea *Watch lighting indicators*.

6.2.3.2 Definición de los objetos de interacción

Los objetos de interacción aparecen como resultado de la asociación de las acciones con los elementos del modelo de dominio que manipulan para realizar la acción.



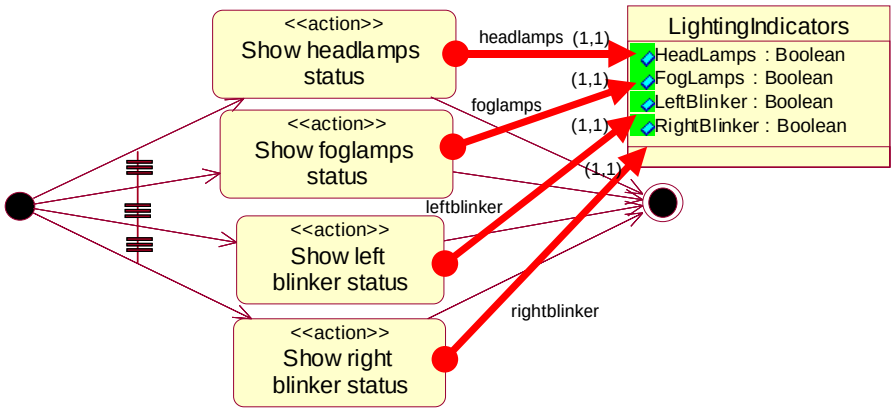
Name	Speed	
Type	int	
	Initial value	0

Figura 6.10 Objetos de interacción para la acción *Show speedometer*.

La figura 6.10 muestra la especificación del objeto de interacción *Speed*. Para mostrar al usuario la velocidad actual, la acción debe tomar el valor del modelo de dominio. La velocidad será mostrada como un objeto de tipo entero cuyo valor inicial será 0. La especificación de los objetos de

interacción para las acciones *Watch RPM*, *Watch Oil Temperature* y *Watch system messages* sería similar a la mostrada en la figura 6.10.

La especificación de los objetos de interacción asociados las acciones de la tarea *Watch lighting indicators* se describen en la figura 6.11. En este caso los elementos del dominio asociados a las acciones son atributos booleanos. Se ha especificado como valor inicial para los atributos el valor *false*, ya que inicialmente se supone que todas las luces estarán apagadas.



Name	headlamps	
Type	Boolean	
	Initial value	false

Name	foglamps	
Type	Boolean	
	Initial value	false

Name	leftblinker	
Type	Boolean	
	Initial value	false

Name	rightblinker	
Type	Boolean	
	Initial value	false

Figura 6.11 Objetos de interacción para las acciones de *Watch lighting indicators*.

6.2.3.3 La interfaz de usuario abstracta

A partir del modelo de tareas y los objetos de interacción asociadas a las acciones de dicho modelo se deriva una interfaz de usuario abstracta que representa, de una manera independiente de la modalidad y la plataforma, la futura interfaz de usuario.

Derivación de la estructura de contenedores

La estructura de contenedores es derivada a partir de la información del modelo de tareas, especialmente de la estructura jerárquica implícita en el modelo de tareas y la propiedad *Presentation* asociada a las tareas abstractas.

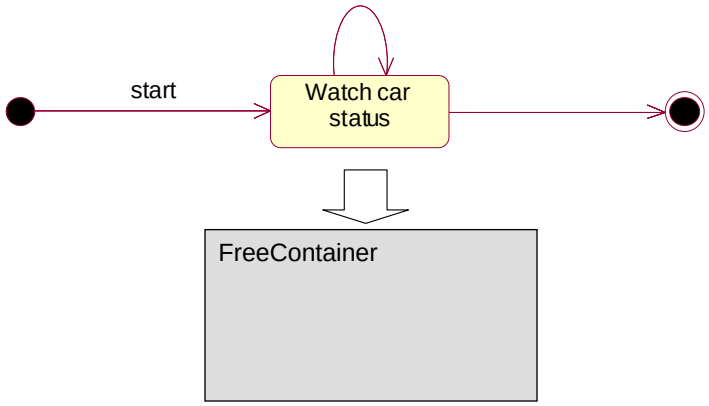


Figura 6.12 Interfaz de usuario abstracta para la tarea raíz de Car UI.

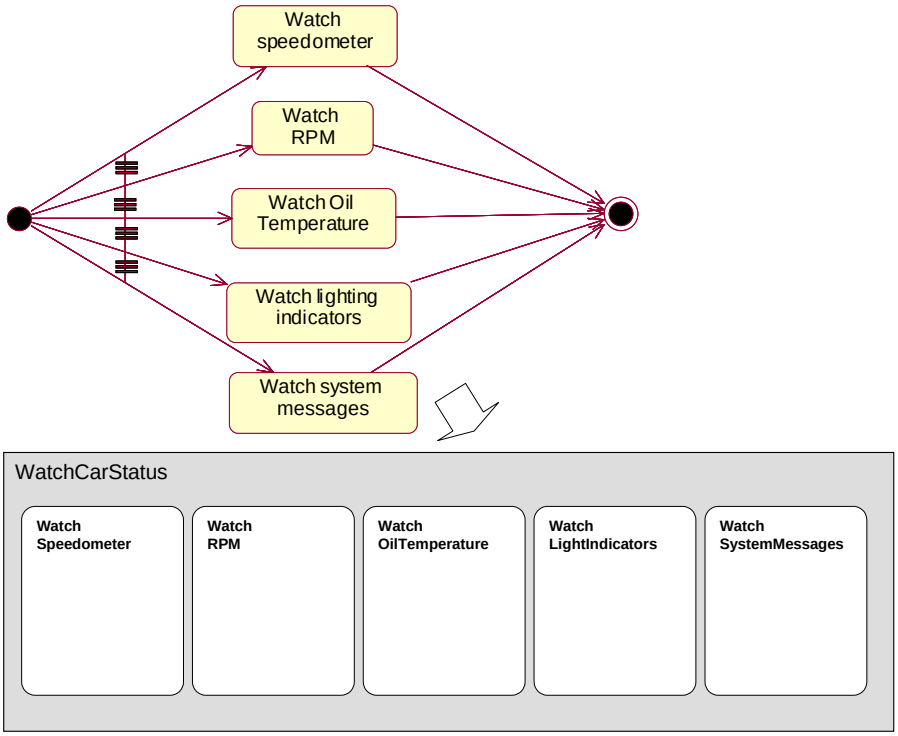


Figura 6.13 Estructura de contenedores abstractos derivada en Car UI.

La tarea abstracta *Watch car status* será representada en un *FreeContainer*, tal y como se muestra en la figura 6.12. Todas aquellas tareas que refinan esta tarea estarán contenidas dentro del *FreeContainer* de *Watch car status*, excepto para aquellos casos donde el diseñador explícitamente exprese su deseo de que una tarea concreta sea presentada en otro *FreeContainer*.

A partir de las tareas que refinan la tarea raíz *Watch car status* se derivan los contenedores abstractos asociados. En este caso, durante la especificación del modelo de tareas se ha establecido que las tareas abstractas que refinan *Watch car status* se presentarán en *Containers*. La figura 6.13 muestra la estructura de contenedores derivada para Car UI.

Derivación de los objetos abstractos de interacción

Los objetos abstractos de interacción que son necesarios para llevar a cabo una acción se derivan a partir de los objetos de interacción que maneja, y sus propias propiedades. En el ejemplo de la figura 6.14 se ha derivado un *Displayer* denominado *Speed* (como el objeto de interacción que debe mostrar), ya que la acción es de tipo *Output* (salida). Objetos de interacción adicionales pueden aparecer como consecuencia de la derivación del control de diálogo especificado con las etiquetas de las transiciones entre las tareas.

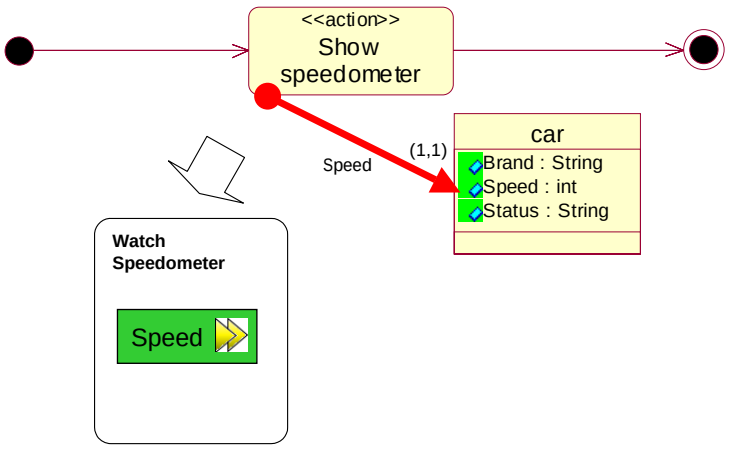


Figura 6.14 Objetos abstractos de interacción derivados de la acción *Show speedometer*.

Los objetos abstractos de interacción derivados a partir de las acciones *Show RPM*, *Show oil temperature*, y *Show system messages* se derivan de forma análoga a cómo se han derivado para la acción *Show speedometer*.

Obsérvese como en la figura se ha derivado un *Displayer* para mostrar cada uno de los cuatro tipos de luces que se deben presentar al usuario. Los cuatro *Displayers* han sido incluidos dentro del contenedor definido para la tarea a la que refinan *(Watch lighting indicators)*.

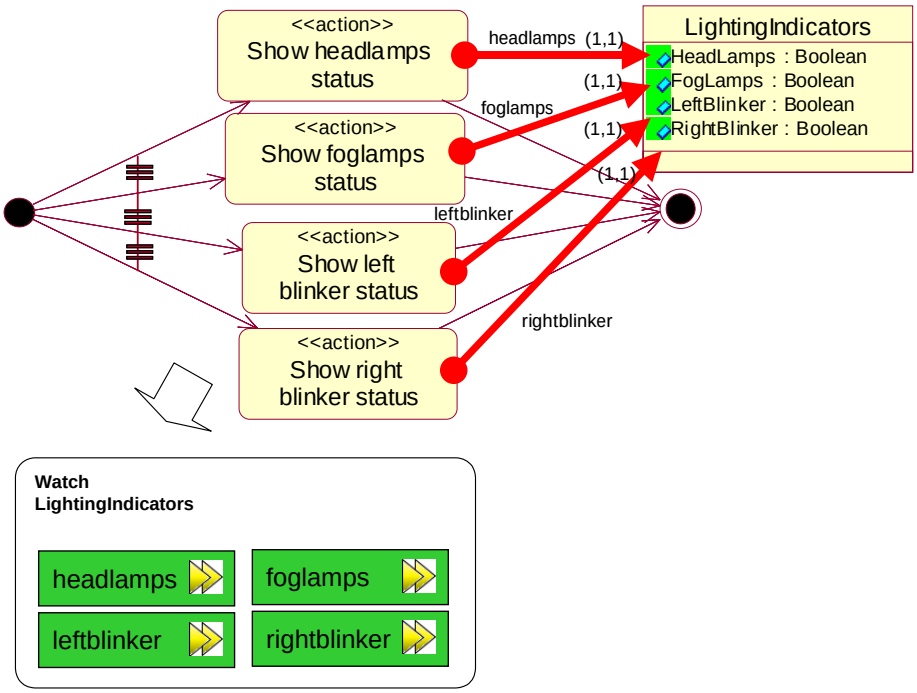


Figura 6.15 Objetos abstractos de interacción derivados para las acciones de *Watch lighting indicators*.

6.2.3.4 La interfaz de usuario concreta

La interfaz de usuario concreta representa la futura interfaz de usuario de una forma abstracta, aunque de una manera más cercana a la implementación final que la interfaz de usuario abstracta, y que será dependiente de la modalidad elegida para la presentación de la interfaz de usuario (aunque no será dependiente de la plataforma elegida). La derivación de la interfaz de usuario concreta se basa en los objetos de abstractos de interacción, el tipo de acción asociada a los objetos abstractos de interacción y las propiedades de los objetos de interacción que manipulan las acciones. A continuación se describe la derivación de la interfaz de usuario par Car UI para la modalidad de interacción gráfica.

Derivación de la estructura de contenedores concretos

Cada uno de los *FreeContainer* derivados en la interfaz de usuario abstracta será convertido en un contenedor concreto de tipo *Window*, los cuales representan las típicas ventanas en un entorno gráfico. Los *Containers* son convertidos en contenedores concretos de tipo *Box*, los cuales agrupan los elementos de la interfaz de usuario, y permiten la distribución de los componentes de la interfaz de usuario.

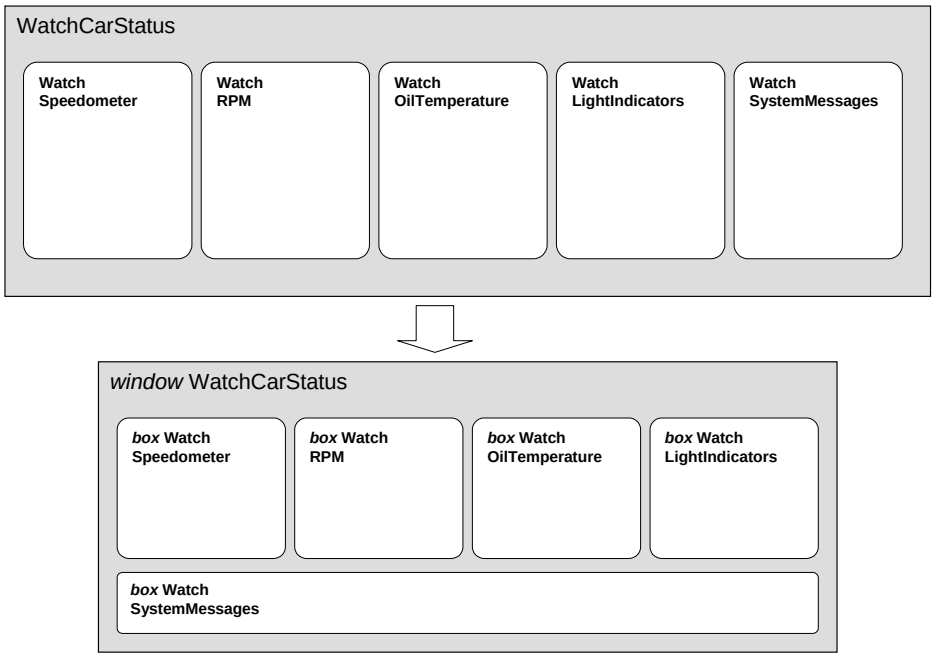


Figura 6.16 Estructura de contenedores concretos derivada para Car UI.

Debe destacarse que actualmente no existe ningún algoritmo que cree una distribución de los elementos concretos para cualquier interfaz de usuario creando interfaz de usuario tan usables como las interfaces en las que la distribución haya sido creada por un diseñador humano. Sin embargo, sí que existen dichos algoritmos para la distribución de los elementos de interfaces de usuario basadas en formularios. Por ello es necesario destacar que la distribución final de los elementos concretos de la interfaz de Car UI ha sido refinada manualmente.

Derivación de los objetos concretos de interacción

Los objetos concretos de interacción son derivados a partir de los objetos abstractos, los tipos de acciones, las transiciones entre las tareas, los

objetos de interacción que manipulan las tareas y las propiedades de las tareas.

En la figura 6.17 se describen los objetos concretos de interacción derivados para la acción *Show speedometer*. Para cada acción que deba mostrar un objeto de interacción al usuario, se ha derivado una etiqueta *(textComponent* con *isEditable =false)* que describe qué se está mostrando (*SpeedLabel)*, una etiqueta *(textComponent* con *isEditable =false)* que mostrará el valor tomado del elemento del dominio *(Speed)*, y una relación gráfica de adyacencia que denota que ambos componentes están relacionados y por lo tanto deben ser mostrados unidos. Sin embargo, otras representaciones podrían haber sido escogidas para la representación de un valor numérico, como por ejemplo, una representación gráfico del velocímetro o una barra de progreso.

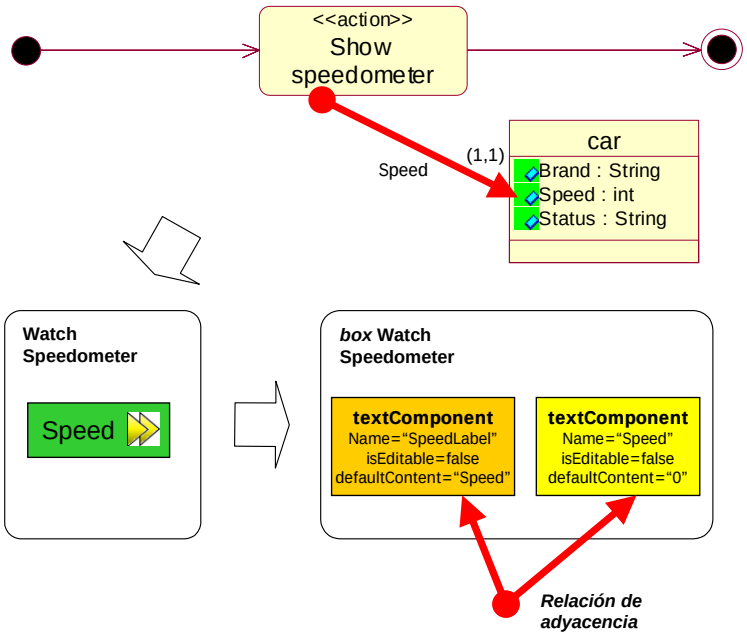


Figura 6.17 Objetos concretos de interacción para la acción *Show speedometer*.

Los objetos concretos de interacción derivados a partir de las acciones *Show RPM*, *Show oil temperature*, y *Show system messages* se derivan de forma análoga a cómo se han derivado para la acción *Show speedometer*.

Los objetos concretos de interacción derivados para las acciones asociadas a la tarea *Show lighting indicators* se muestran en la figura 6.18. Obsérvese

que en este caso los objetos de interacción que los objetos abstractos *Displayers* deben mostrar son de tipo booleano. Existen múltiples posibilidades de representación de un valor booleano en una interfaz de usuario gráfica. Las más habituales son una casilla de verificación *(checkBox)*, un botón de radio *(radioButton)* o un botón de conmutación *(toggleButton)*. En nuestro caso se ha optado por representar el valor booleano mediante dos imágenes: una imagen para representar el valor *false* y otra para representar el valor *true*.

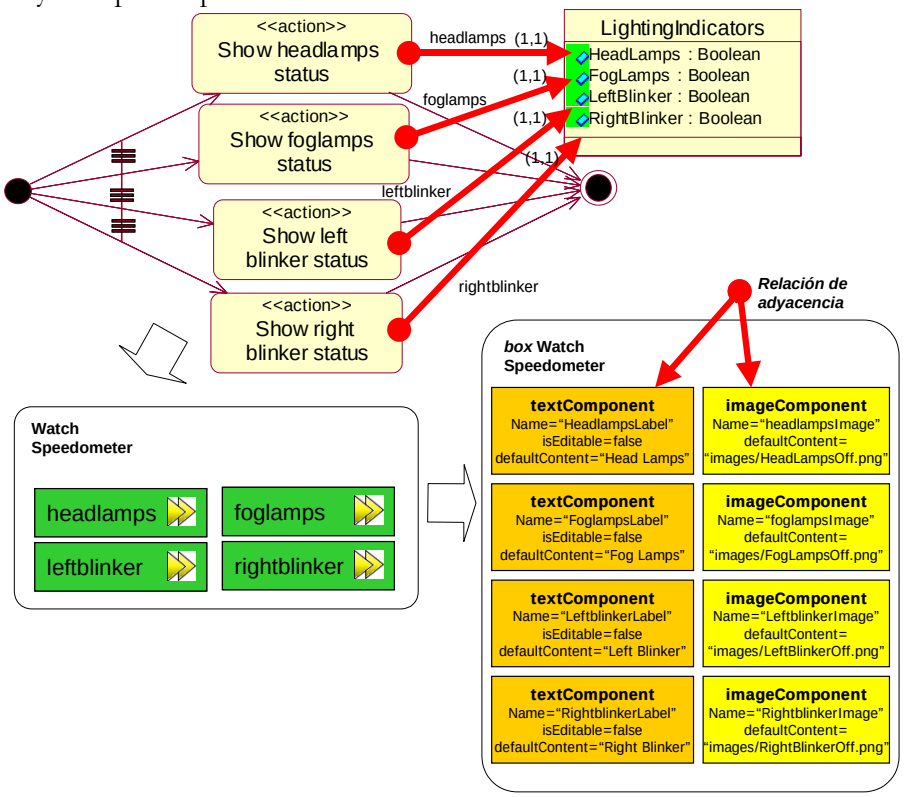


Figura 6.18 Objetos concretos de interacción para las acciones asociadas a la tarea *Watch lighting Indicators*.

Al igual que en el caso anterior, para cada *Displayer* se ha derivado una etiqueta *(HeadlampsLabel)* que describe lo que se muestra, y una imagen que representan el valor inicial especificado. En este caso *false (Off)*. De forma análoga a como sucediera en el caso anterior, una relación de adyacencia ha sido derivada para representar la relación existente entre cada etiqueta y la imagen que representa el valor.

Trazabilidad del proceso de derivación

Mediante el diagrama de conectores, creado automáticamente a partir de las derivaciones de las interfaces de usuario abstracta y concreta el diseñador puede observar cómo se ha llevado a cabo el proceso de derivación, es incluso modificarlo. La figura 6.19 muestra el diagrama de conectores para las acciones asociadas a la tarea *Watch lighting indicators*.

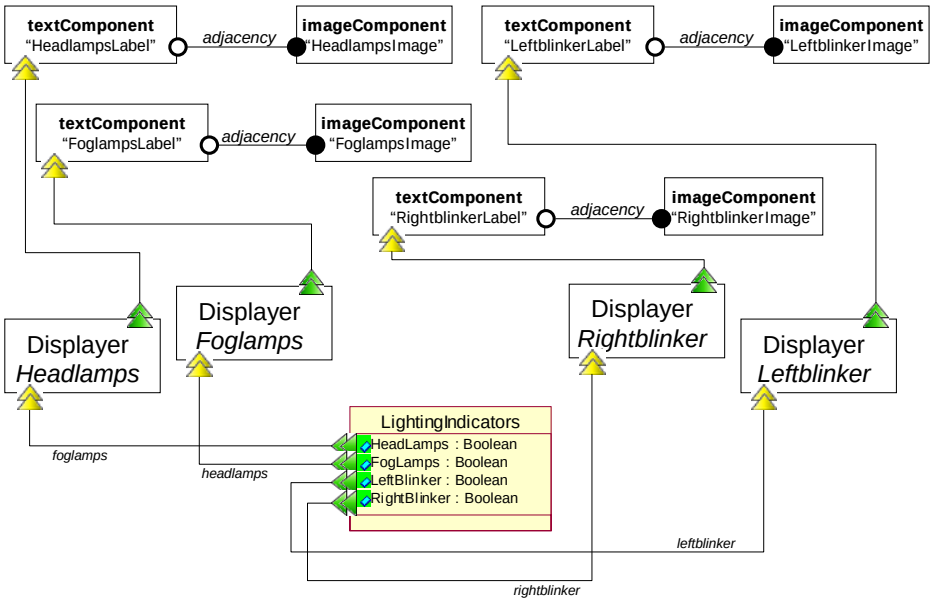


Figura 6.19 Diagrama de conectores para la tarea *Watch lighting indicators*.

6.2.3.5 Reglas de adaptación

Las reglas de adaptación representan cómo debe responder el sistema para adaptarse a los cambios en el contexto de uso. La especificación de las reglas de adaptación se basa (véase la sección 3.4.1.1)en la descripción de los sensores que captan y producen información a partir de los cambios en el contexto de uso, los eventos del contexto compuestos por la información que producen los sensores, las precondiciones que filtran cambios en el contexto irrelevantes, y la transformación que describe el efecto que la regla de adaptación tendrá sobre la interfaz.

A continuación se describen las reglas de adaptación definidas para el sistema de asistencia a la conducción Car UI. La especificación completa

de las reglas de adaptación propuestas pueden encontrarse en el Apéndice A al final de este documento.

Sensores

Los sensores deben describir qué información de capta del contexto de uso par obtener una visión actualizada del estado actual de contexto en cada momento.

```
<hardwareEnvironmentSensor id="HES001"
                name="lightingConditionsHardwareEnvironmentSensor"
                dataName="/contextInfo/environmentInfo/lightingConditions"
                dataType="LightingConditionsType"
                description="Senses changes in the lighting conditions"/>
```

Figura 6.20 Especificación de un sensor para Car UI.

La figura 6.20 muestra la especificación de un sensor que captura el grado actual de iluminación del lugar por donde va transitando el coche. Todos los sensores deben tener un identificador único, así como un nombre. El atributo *dataName* describe dónde se almacena el valor obtenido por el sensor siguiendo la sintaxis definida para XPath[36]. Los datos obtenidos por el sensor son descritos especificando su tipo de datos. Dicho tipo de datos puede ser tanto uno de lo predefinidos como uno definido por el diseñador. En el ejemplo de la figura 6.20 se ha especificado que el tipo de datos de los datos capturados sea *LightingConditionsType* (tipo enumerado que pueden tomar los valores: *poor*, *low*, *average*, *high)*. Finalmente, los sensores pueden ser documentados proporcionado una descripción (atributo *description)*.

Eventos del contexto

Los eventos del contexto están compuestos por los valores capturados por los sensores, y serán los que finalmente disparen la ejecución de la reglas.

```
<contextEvent id="CE004" name="lightingConditionsContextEvent"
        description="Produced when lighting conditions in the road change.">
            <contextEventSensors>
                <sensor id="HES001"/>
            </contextEventSensors>
</contextEvent>
```

Figura 6.21 Especificación de un evento del contexto para Car UI.

[36] http://www.w3.org/TR/xpath

En la figura 6.21 se describe uno de los eventos del contexto para CarUI. El evento mostrado es producido cuando cambian las condiciones de iluminación de la carretera. Los eventos del contexto son descritos mediante un identificado único, un nombre, una descripción, y los sensores que producen ese evento del contexto. El evento es producido por el sensor *HES001*, descrito en la figura 6.20, el cual detecta cuando cambian las condiciones de iluminación de la calzada. A continuación se describe una de las reglas que utilizan el evento del contexto descrito en la figura 6.21.

Reglas de adaptación

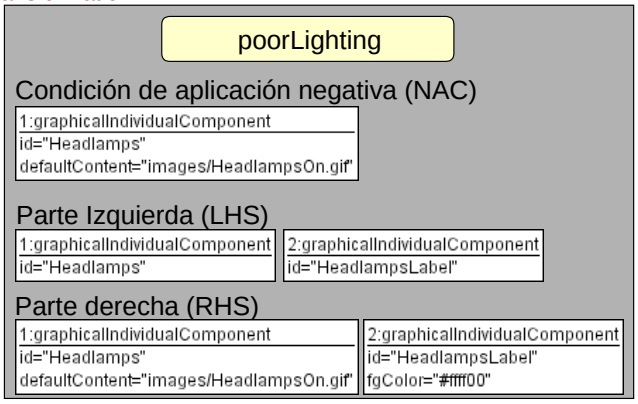
Las reglas de adaptación son producidas por los eventos del contexto. Cuando se produce el evento del contexto que dispara una regla, ésta no es disparada automáticamente, sino que previamente se comprobará que se da la precondición del contexto *(contextPrecondition)* especificada para esa regla. Obsérvese que la especificación del contexto se comporta como un programa lógico, es decir, si alguna de las sentencias lógicas *(logicalSentence)* de la precondición del contexto no se cumple, se considerará que la precondición del contexto no ha sido satisfecha.

Finalmente la regla de adaptación incluye la transformación que será aplicada a la interfaz de usuario y que llevará a cabo la adaptación realmente. Para facilitar al lector la comprensión de las transformaciones usadas en el ejemplo, se ha optado por mostrarlas utilizando una sintaxis gráfica donde se describe la condición de aplicación negativa, la parte izquierda de la regla y la parte derecha de la regla (véase la sección 3.4.1.4).

En la figura 6.22 se muestra la especificación de una regla de adaptación para Car UI que enciende automáticamente las luces cuando el nivel de iluminación de la calzada es muy bajo. La regla es disparada por el evento del contexto *CE004* (véase la figura 6.21). La precondición del contexto está formada por dos sentencias lógicas. En las sentencias lógicas pueden aparecer referencias a cualquier sensor definido y a cualquier atributo del modelo de contexto. Todos los sensores contienen dos atributos, uno donde se puede acceder al valor actual almacenado para la información que captura el sensor (atributo *value)*, y otro para el nuevo valor capturado y que reemplazará al valor nuevo (atributo *newValue)*. En la sentencia lógica *poorLighting* se comprueba que el nuevo valor capturado por el sensor *HES001* (véase la figura 6.20) es *poor*, es decir, que las condiciones de iluminación de la calzada son muy malas. La sentencia lógica

headLampsOff comprueba que las luces delanteras no estén ya encendidas, ya que si las luces ya están encendidas la regla no debería aplicarse.

```
<adaptationRule id="AR003" name="poorLighting"
                description="When the lighting conditions are too low Car UI turns on
                automatically the head lamps.">
        <contextEvents>
            <contextEvent id="CE004"/>
        </contextEvents>
        <contextPrecondition id="CP003" name="poorLightingContextPrecondition">
            <logicalSentence id="LS003" name="poorLighting"
                    sentence="HES001.newValue = 'poor' "/>
            <logicalSentence id="LS004" name="headLampsOff"
                    sentence="HPS004.Value = 'false' "/>
        </contextPrecondition>
        <transformation>
```




```
        </transformation>
    </adaptationRule>
```

Figura 6.22 Especificación de una regla de adaptación para Car UI.

La regla de transformación *poorLighting* se aplica sobre la imagen *Headlamps*, y sobre la etiqueta *HeadlampsLabel* asociada a dicho elemento imagen (especificado en la parte izquierda de la regla). A continuación hace que la imagen mostrada en el elemento imagen *Headlamps* sea la correspondiente a cuando las luces delanteras están encendidas *(images/HeadlampsOn.gif)*, y cambia el color de la etiqueta *HeadlampsLabel* a "#ffff00" (amarillo), para que sea más visible la adaptación para el conductor (todo ello especificado en la parte derecha de la regla). Finalmente la condición de aplicación negativa evita que la transformación se aplique una y otra vez en un bucle infinito, indicando que si la imagen asociada a *Headlamps* ya es *images/HeadlampsOn.gif*, entonces la regla ya ha sido aplicada y no debe volver a aplicarse.

Las reglas de adaptación serán transformadas en planes para el agente *AgentAdaptationProcess* de forma automática.

Evaluación del compromiso de usabilidad

En el compromiso de usabilidad diseñado en el apartado 6.2.2.2 hay tres criterios que influyen sobre la meta final de ofrecer adaptatividad al usuario: (1) visibilidad *(visibility)*, (2) continuidad *(continuity)*, y (3) accesibilidad *(accesibility)*, y que por lo tanto tendrán que ser evaluados tras la aplicación de cada adaptación para comprobar que se mantienen.

Ya que la visibilidad tiene un fuerte impacto *(make)* sobre la calidad de la adaptación, cualquier adaptación que se aplique tendrá que mantener al menos la misma visibilidad que existe en la interfaz de usuario actual. Es necesario recordar que la evaluación de los criterios de calidad se realiza siempre comparando la interfaz de usuario actual con la interfaz de usuario resultante tras la aplicación de las adaptaciones. La evaluación de la visibilidad se realiza utilizando un proceso análogo al descrito en el apartado 5.2.6.1.

La continuidad tiene un impacto medio *(Some+)* sobre la calidad de las adaptaciones. En este caso cualquier adaptación aplicada tendrá que mantener la discontinuidad producida por debajo del 0.5 (la discontinuidad varía entre 0 y 1). La discontinuidad producida por una adaptación es evaluada usando un proceso análogo al descrito en el apartado 5.2.6.1.

Finalmente, el criterio de accesibilidad *(accessibility)* tiene un impacto bajo *(Help)* sobre la calidad de las adaptaciones. La accesibilidad es evaluada mediante la especificación de los subcriterios de accesibilidad que se desea mantener usando restricciones atómicas (véase la sección 5.2.7). En este caso ya que el criterio de accesibilidad tiene un impacto bajo, cualquier adaptación tendrá que mantener al menos un 25% de los subcriterios establecidos para la accesibilidad. En el caso de Car UI, se han creado restricciones atómicas para algunos de los criterios especificados en el estándar de accesibilidad WAI[37] definido por el consorcio W3C. En la figura 5.23 de la sección 5.2.7.1 se describe una restricción atómica para forzar a que todas las imágenes de la interfaz de usuario contengan un

[37] http://www.w3.org/WAI/

texto alternativo (guardado en el atributo *defaultTooltip*), de forma que se cumpla con una de las directrices propuestas dentro del estándar WAI.

6.2.4 Implementación

Una vez concluida la fase de diseño se deben transformar los modelos creados en código ejecutable o interpretable por la plataforma destino. Por una parte, la interfaz de usuario diseñada debe ser transformada a un código que permita al usuario interactuar con ella, y por otra parte es necesario introducir las capacidades de adaptación diseñadas dentro del sistema multi-agente que hace las funciones de motor de adaptación.

Renderización de la interfaz de usuario concreta

La interfaz de usuario concreta creada en la etapa de diseño (véase el Apéndice B) es transformada en código final. En el Apéndice C se pueden observar el código en el lenguaje de XML de representación de interfaces de usuario XUL generado por la herramienta desarrollada: *usiXMLTransformer*. El código XUL generado pueden ser visualizado en cualquier navegador compatible con los navegadores basados en el motor desarrollado por la fundación *Mozilla*[38]. La figura muestra la interfaz de usuario visualizada en el navegador *Mozilla*.

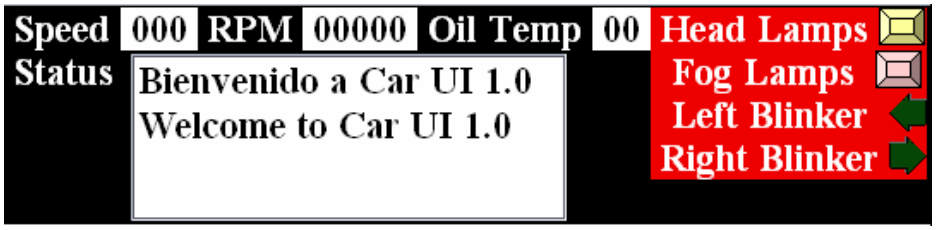


Figura 6.23 Presentación de la interfaz de usuario de Car UI.

Integración de las capacidades de adaptación en el motor de adaptación

La integración de las capacidades de adaptación debe transformar los modelos creados en la etapa de diseño y análisis dentro del motor de adaptación.

Las reglas de adaptación son las opciones que el agente encargado de realizar el proceso de adaptación tiene para afrontar los cambios en el contexto de uso. Por lo tanto las reglas de adaptación deben convertirse

[38] http://www.mozilla.org

en planes para dicho agente *(AgentAdaptationProcess)*. Para ello se genera un plan basada en cada regla de adaptación que será después será convertido en código Java por el precompilador del entorno de programación de agentes *JACK* [Bus99], y entre los que el sistema multi-agente tendrá que escoger en tiempo de ejecución a la hora de aplicar las adaptaciones más apropiadas a cada situación.

6.3 Caso de estudio: ATM UI

En este segundo caso de estudio se presenta la especificación de una interfaz de usuario para una aplicación que facilita la realización de las tareas más habituales disponibles en un cajero automático de un banco: sacar dinero, depositar dinero en la cuenta, hacer transferencias a otras cuentas, realizar consultas sobre el balance y los movimientos de las cuentas o recargar la tarjeta de un teléfono móvil. La interfaz de usuario es capaz de funcionar tanto usando la plataforma suministrada por el banco como dispositivos móviles. La aplicación se denomina ATM UI *(Automatic Teller Machine User Interface)*, y permite al usuario personalizar la apariencia de la aplicación cambiando el aspecto de la interfaz de usuario presentada. A continuación se describen algunos de los diagramas que ilustran cada una de las fases.

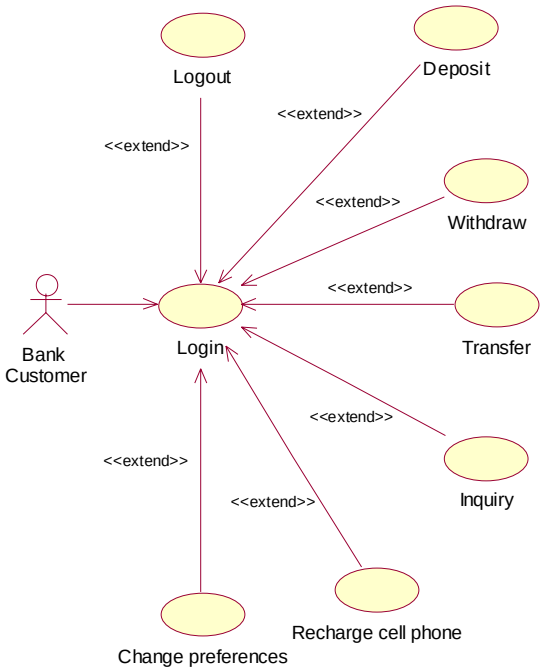


Figura 6.24 Modelo de casos de uso para ATM UI.

6.3.1 Adquisición de requisitos

A continuación se describen qué tareas podrá realizar el cliente del banco a través de la interfaz de usuario, y los modelos correspondientes al

modelos que describen el contexto estático de uso (modelos de usuario, plataforma y entorno).

6.3.1.1 Casos de uso

En la figura 6.24 se muestra el diagrama de casos para la aplicación ATM UI. Como se observa en el diagrama, el usuario puede entrar en el sistema *(login)*. Además de entrar en el sistema el usuario puede desconectarse del sistema *(logout)*, hacer un deposito en su cuenta *(deposit)*, sacar dinero *(withdraw)*, transferir dinero a otra cuenta *(transfer)*, consultar el balance y los últimos movimientos en la cuenta *(inquiry)*, recargar la tarjeta de un teléfono móvil *(recharge cell phone)* o cambiar sus preferencias *(change preferences)*.

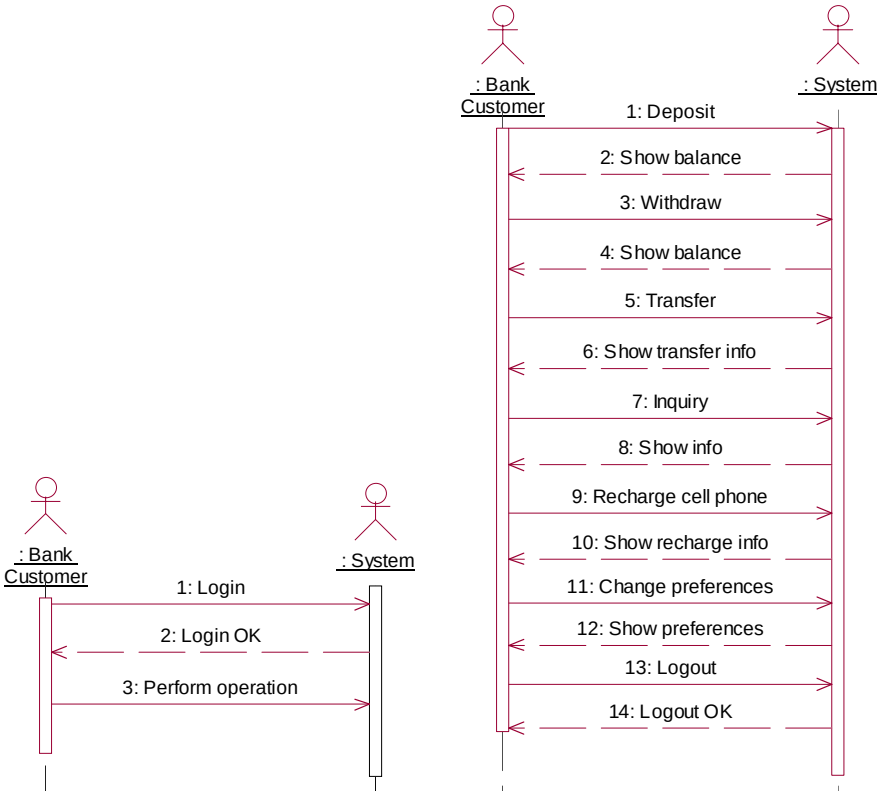


Figura 6.25 Relaciones temporales entre los casos de uso de ATM UI.

En la figura 6.24 solamente se describen los casos de uso, aunque no se describen las relaciones temporales entre los distintos casos de uso. En la figura 6.25 se muestran las relaciones temporales para los casos de uso de ATM UI. En dicha figura se observa como el usuario debe conectarse al

sistema *(login)* antes de poder realizar cualquier otra operación. A continuación el usuario puede hacer un depósito, sacar dinero, consulta el estado de la cuenta, hacer una transferencia, recargar la tarjeta de un teléfono móvil, cambiar las preferencias o desconectarse del sistema.

Los diagramas mostrados en las figuras 6.24 y 6.25 servirán como punto de partida para mostrarle al usuario cómo será el futuro sistema, y una vez refinados darán lugar al modelo de tareas de la aplicación.

6.3.1.2 Modelos de usuario, plataforma y entorno

Dentro de ATM UI se han considerado dos tipos de plataformas destino. El propio cajero del banco y un dispositivo móvil (una PDA). Existe un estereotipo de usuario *(Customer*). Derivados de las dos plataformas consideradas, se tienen en cuenta dos entornos distintos. Por una parte el propio banco donde se encuentra el cajero automático, y por otra parte las calles, donde el usuario puede interactuar con su dispositivo móvil.

Platform	Bank ATM
platformId	*0025P*
platformName	*BankATM*
isColorCapable	*true*
isImageCapable	*true*
screenSize	*400x350*

User	Customer
userId	*0025U*
userName	*BankCustomer*
Font	*Arial*
FontColor	*Black*
Modality	*Graphical*

Platform	PDA
platformId	*0026P*
platformName	*PDA01*
isColorCapable	*true*
isImageCapable	*true*
screenSize	*220x144*

Environment	Bank
environmentId	*0028E*
environmentName	*BankOffice*
lightingLevel	*low*
isNoisy	*false*

Environment	Street
environmentId	*0029E*
environmentName	*InTheStreets*
lightingLevel	*average*
isNoisy	*true*

Figura 6.26 Modelos de usuario, plataforma y entorno para ATM UI.

Para las plataformas usadas se han considerado, sus capacidades para mostrar imágenes y colores, y la resolución de la pantalla. Del usuario se han considerado sus preferencias, teniendo en cuenta qué fuente prefiere, de qué color, y si prefiere interactuar con una interfaz en modo textual o gráfico. Finalmente, de los entornos se ha considerado el grado de luminosidad y de ruido.

6.3.2 Análisis

En esta etapa se describen los objetos necearios para llevar a cabo las tareas presentadas en el modelo de casos de uso y el compromiso de usabilidad para las distintas plataformas consideradas.

6.3.2.1 Modelo de dominio

El modelo de dominio de ATM UI, mostrado en la figura 6.27, describen los clientes *(customers)* que acceden al banco usando el cajero automático y las cuentas que dichos clientes tienen (*account)*. Los clientes tienen unas preferencias *(preferences)* y cada cuenta del cliente tiene una serie de apuntes que describen los movimientos realizados sobre la cuenta.

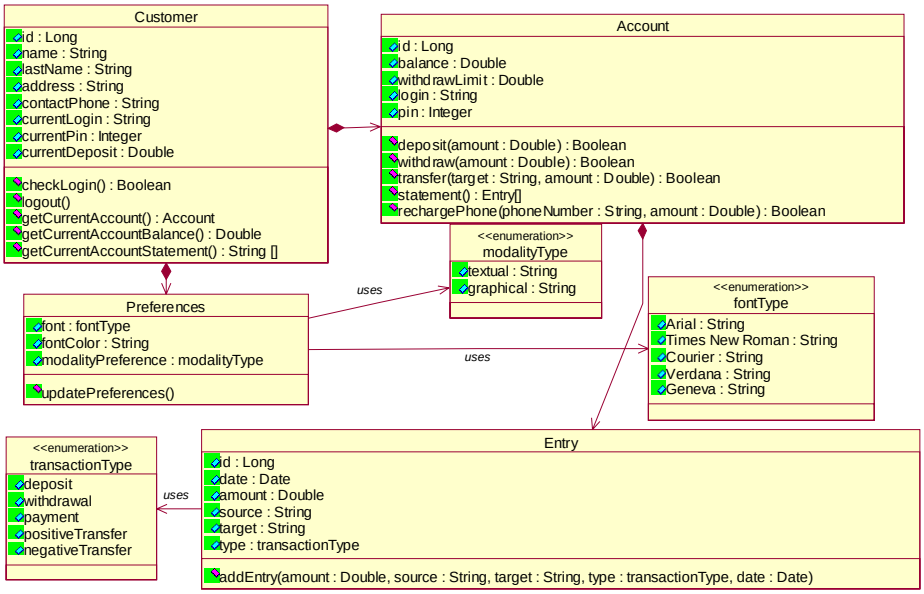


Figura 6.27 Modelo de dominio para ATM UI.

6.3.2.2 Compromiso de usabilidad

Para la interfaz de un cajero automático, uno de los factores principales de usabilidad es la accesibilidad del sistema, ya que cualquier persona debe ser capaz de realizar operaciones con su cuenta corriente. Otros de los factores importantes es la consistencia de la presentación. Las interfaces de usuario de los cajeros automáticos son usadas habitualmente, tanto por personas con experiencia en el uso de aplicaciones software como por personas sin ningún tipo de experiencia, y que basan la facilidad de uso de la aplicación en la memorización de cómo funciona la interfaz, y que por

lo tanto se les facilita la tarea sin se mantiene la consistencia de la presentación. Finalmente, es importante que se mantenga un alto grado de visibilidad de los elementos necesarios para la realización de las tareas, evitando tener que navegar en exceso para conseguir obtener los datos necesarios. La figura 6.28 ilustra el compromiso de usabilidad especificado para la plataforma *Bank ATM* (cajero automático del banco) usando la notación descrita en el apartado 4.2.2.3. En el compromiso de usabilidad mostrado se observa como cualquier adaptación debe mantener un alto grado de accesibilidad, y debe mantener en parte la visibilidad y la consistencia.

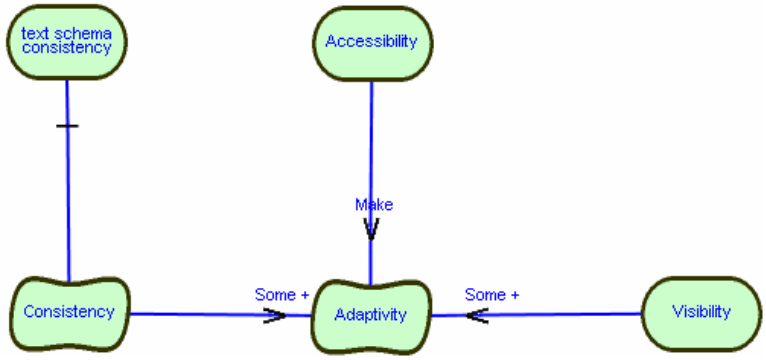


Figura 6.28 Compromiso de usabilidad para la plataforma *Bank ATM*.

Sin embargo, para el compromiso de usabilidad especificado para la plataforma *PDA* no se ha incluido el criterio de visibilidad, ya que al mantenimiendo de dicho criterio para un dispositivo con una resolución tan pequeña puede forzar a reducir el grado de accesibilidad de la interfaz adaptada.

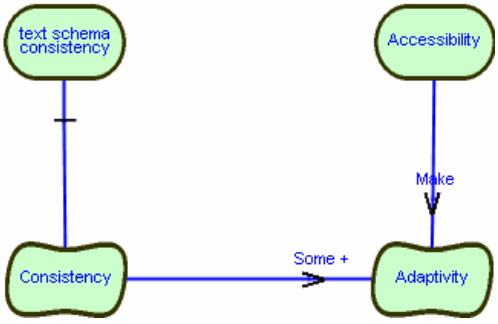


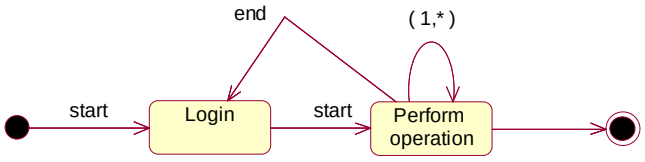
Figura 6.29 Compromiso de usabilidad para la plataforma *PDA*.

6.3.3 Diseño

En esta etapa se refinan los modelos creados en las etapas anteriores para crear una descripción de la interfaz de usuario suficientemente detallada para permitir su implementación. Para ello será necesario especificar un modelo de tareas suficientemente detallado, derivar los objetos abstractos necesario para llevar a cabo las tareas, y derivar finalmente la interfaz de usuario concreta. Por otra parte se especificará las capacidades de adaptación que se ofrecerán al usuario descriendo los sensores, eventos del contexto y reglas implicadas.

6.3.3.1 Modelo de tareas

A continuación se describe el modelo de tareas obtenido mediante el refinamiento de los requisitos recogidos en el diagrama de casos de uso y secuencia.



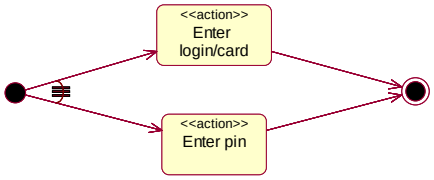
Task name	Login
Description	The bank customer enters the card or the login.
Type	Abstract
Frecuency	*High*
Precondition	NULL
Postcondition	Customer.checkLogin()
Presentation	FreeContainer

Task name	Perform operation
Description	The customer can do one of these operations one or several times: deposit, withdraw, transfer, inquiry, recharge cell phone or change preferences.
Type	Abstract
Frecuency	*High*
Precondition	NULL
Postcondition	NULL
Presentation	FreeContainer

Figura 6.30 Tareas raíz para ATM UI.

En la figura 6.30 se describen las tareas raíz para ATM UI. Inicialmente, el usuario sólo puede realizar la tarea *Login* (conectarse al sistema). Una vez realizada esa tarea con éxito (segunda transición *start)* el usuario podrá realizar el resto de operaciones (tarea *Perform Operaton)*. Si la tarea *Login* no es realizada con éxito el usuario tendrá que intentar otra vez la tarea (transición *end)*. Una vez realizada la tarea *Login* con éxito el usuario podrá realizar de 1 a *n* veces la tarea *Perform Operation*. El usuario debe realizar como mínimo una vez dicha tarea, porque al menos tendrá que

desconectarse del sistema (tarea *Logout)*. Nótese como tanto la tarea abstracta *Login* como la tarea abstracta *Perform Operation* se muestran en sus correspondientes *FreeContainers*.



Action name	EnterLogin_card
Description	The user types in the login or enters the card.
Type	*Input*
Frecuency	*High*
Precondition	NULL
Postcondition	*Customer.currentLogin!=“”*

Action name	EnterPin
Description	The user types in the password for the login.
Type	*Input*
Frecuency	*High*
Precondition	*Customer.currentLogin!=“”*
Postcondition	*Customer.currentPassword!=“”*

Figura 6.31 Acciones para la tarea *Login*.

En la figura 6.31 se describen las acciones asociadas a la tarea *Login*. Obsérvese como la realización de ambas acciones no terminará hasta que el usuario haya introducido un nombre de usuario y su contraseña *(pin)*.

Task name	Deposit
Description	The bank customer makes a deposit in the account.
Type	Abstract
Frecuency	*Low*
Precondition	NULL
Postcondition	NULL
Presentation	FreeContainer

Task name	Logout
Description	The bank customer disconnects from the ATM application.
Type	Abstract
Frecuency	*High*
Precondition	NULL
Postcondition	NULL
Presentation	Container

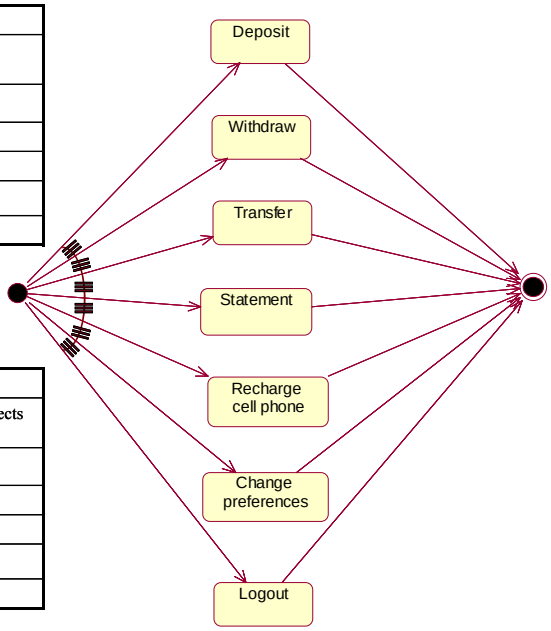
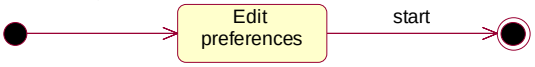
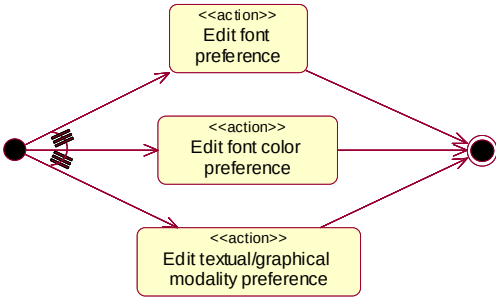


Figura 6.32 Submodelo de tareas para la tarea *Perform Operation*.

En la figura 6.32 se describe el submodelo de tareas asociado a la tarea *Perform Operation*. Todas las tareas serán mostradas en *FreeContainers*, excepto la tarea *Logout* que será mostrada en un *Container*.



Task name	Edit preferences
Description	The bank customer edits the preferences for the ATM application.
Type	Abstract
Frecuency	*Low*
Precondition	NULL
Postcondition	NULL
Presentation	Container



Action name	EditFontPreference
Description	The ATM shows last month entries in the account.
Type	*Modify*
Frecuency	*Low*
Precondition	
Postcondition	

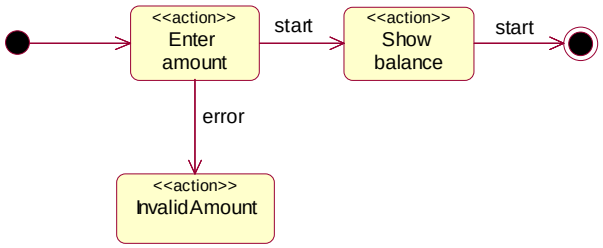
Action name	EditTextual_GraphicalModalityPreference
Description	The ATM shows last month entries in the account.
Type	*Modify*
Frecuency	*Low*
Precondition	
Postcondition	

Action name	EditFontPreference
Description	The ATM shows last month entries in the account.
Type	*Modify*
Frecuency	*Low*
Precondition	
Postcondition	

Figura 6.33 Modelo de tareas para la tarea *Edit Preferences*.

En la figura 6.33 se detalla el modelo de tareas para la modificación de las preferencias del usuario *(Edit Preferences)*. Dicha tareas permite la modificación de la fuente preferida por el usuario, el color del texto mostrado y si prefiere la modalidad gráfica o textual para que se le muestre la interfaz de usuario. Obsérvese como al igual que ocurre para la tarea *Login* las acciones involucradas pueden ser realizadas en cualquier orden, tal y como indica el operador ||| usado para relacionar las acciones.

En la figura 6.34 se puede observar el modelo de tareas para la realización de un depósito en la cuenta. La acción *Enter amount* debe ir acompañada de la introducción del dinero en la ranura apropiada del cajero. Esta tarea no podrá realizarse a través de la PDA, por no tener físicamente el dipositivo para la introducción del dinero físicamente. Por la misma razón tampoco se podrá sacar dinero a través de la PDA.



Action name	EnterAmount
Description	The customer enters the amount to be deposited in the account.
Type	*Input*
Frecuency	*Low*
Precondition	NULL
Postcondition	*Customer.getCurrentAccount.deposit(Customer.currentDeposit)*

Action name	ShowBalance
Description	The ATM shows the current balance after the deposit has been made.
Type	*Ouput*
Frecuency	*Low*
Precondition	
Postcondition	

Action name	InvalidAmount
Description	The user types in the password for the login.
Type	*Output*
Frecuency	*Low*
Precondition	
Postcondition	

Figura 6.34 Acciones asociadas a la tarea *Deposit*.

6.3.3.2 Definición de los objetos abstractos de interacción

A continuación se describen las relaciones existentes entre las tareas y los elementos del dominio que manipulan, es decir, los objetos de interacción. Los objetos de interacción permiten personalizar cómo se mostrarán los datos del dominio al usuario.

En la figura 6.35 se muestran los objetos de interacción asociados a la tarea *Login*. La acción *Enter login/card* es una acción de entrada. El objeto de interacción *Login* especifica que el valor de entrada para dicha acción

será almacenado en el atributo *currentLogin* de la clase *Customer*. Esta relación permitirá derivar qué tipo de objetos de interacción abtractos y concretos serán apropiados para permitir al usuario que sea capaz de introducir un valor con esas características. De forma totalmente análoga, la acción *Pin* está relacionada con el atributo *currentPin* de la clase *Customer*. Obsérvese como en este caso se ha especificado que el objeto de interacción *Pin* es de tipo contraseña *(password)*, lo que hará que cuando se derive la interfaz de usuario concreta se trate de una manera adecuada (por ejemplo, usando un campo de texto en el que aparezcan asteriscos en vez de los caracteres introducidos por el usuario).

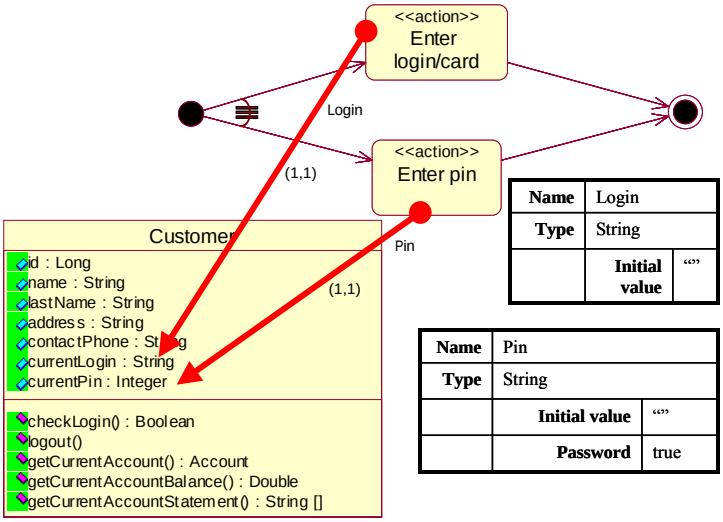


Figura 6.35 Objetos de interacción para la tarea *Login*.

Las figuras 6.36 y 6.37 ejemplifican la definición de los objetos de interacción para dos de las subtareas de la tarea *Perform Operation*.

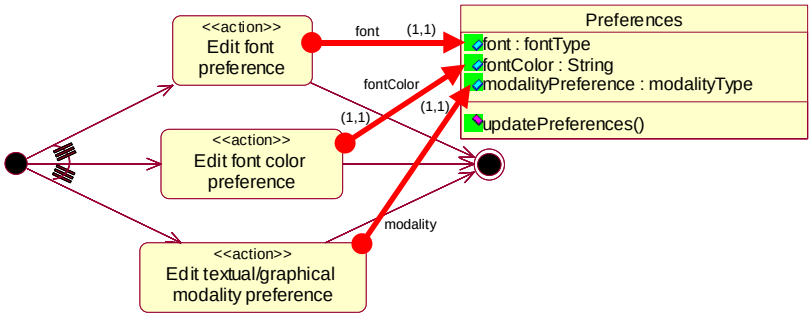


Figura 6.36 Objetos de interacción para la tarea *Change preferences*.

En la figura 6.36 se puede observar como cada una de las tres acciones de entrada de la tarea *Edit preferences* tiene asociado un objeto de interacción (*font*, *fontColor* y *modality*) donde se especifica el formato y la semántica de los datos que el usuario debe introducir para realizar dichas acciones.

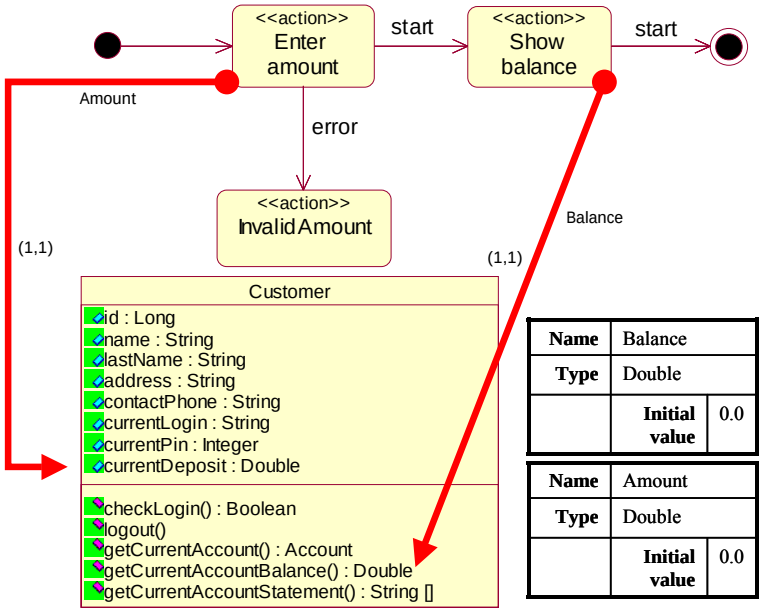


Figura 6.37 Objetos de interacción para la tarea *Deposit*.

Como se puede ver en la figura 6.37 aparecen dos objetos de interacción: *Amount* y *Balance*. *Amount* representa la cantidad de dinero que se desea ingresar en la cuenta actual, y *Balance* representa el saldo de la cuenta tras la realización del ingreso.

6.3.3.3 La interfaz de usuario abstracta

Tras la especificación de los objetos de interacción se deriva la interfaz de usuario abstracta que exprese en un alto nivel de abstracción qué elementos son necesarios para que el usuario pueda realizar las tareas.

Derivación de la estructura de contenedores

La estructura de contenedores para las tareas raíz *Login* y *Perform Operation* se muestra en la figura 6.38. En este caso de derivará un *FreeContainer* para cada una de las tareas, tal y como el diseñador especificó en la descripción de las tareas.

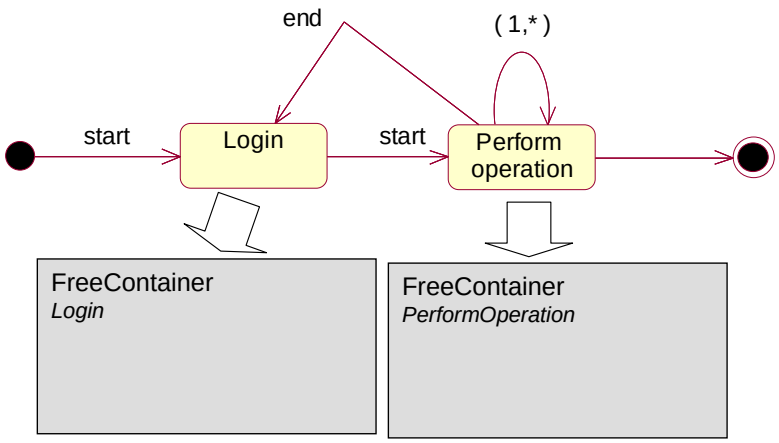


Figura 6.38 Contenedores abstractos asociados a las tareas raíz para ATM UI.

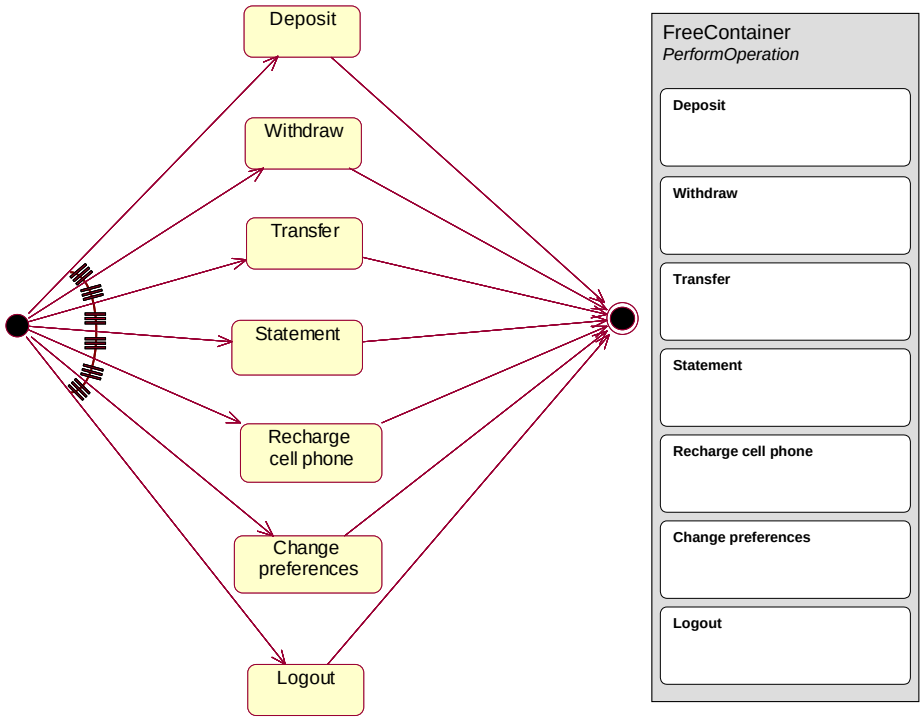


Figura 6.39 Estructura de contenedores para la tarea *Perform Operation*.

La estructura de los contenedores para la la tarea *Perform Operation* (realizar operación) se muestra en la figura 6.39. Obsérvese como aunque las tareas *Deposit*, *Transfer*, *Withdraw*, *Statement*, *Recharge phone* y *Change preferences* deben ser mostradas en otro *FreeContainer*, producen la derivación de un

contenedor dentro del *FreeContainer* asociado a la tarea padre. Esto sucede así porque para que dichas tareas puedan ser realizadas debe existir un mecanismo que haga que se muestre el *FreeContainer* donde finalmente serán ejecutadas, luego los contenedores asociados a las tareas *Deposit*, *Transfer*, *Withdraw*, *Statement*, *Recharge phone* y *Change preferences* contendrán los artefectos necesarios para permitir disparar la realización de dichas tareas en su correspondiente *FreeContainer* (véase la figura 6.40).

El contenedor asociado a la tarea *Logout* (desconectarse) sí que contendrá por el contrario los elementos necesarios para realizar dicha tarea dentro de del *Container logout*, tal y como se describen en la figura.

Derivación de los objetos abstractos de interacción

Los objetos abstractos de interacción representan de una forma abstracta los elementos de la interfaz que el usuario tendrá a su disposición para realizar las tareas, y por lo tanto ya permite dar una primera idea de cómo podría ser la interfaz de usuario final creada.

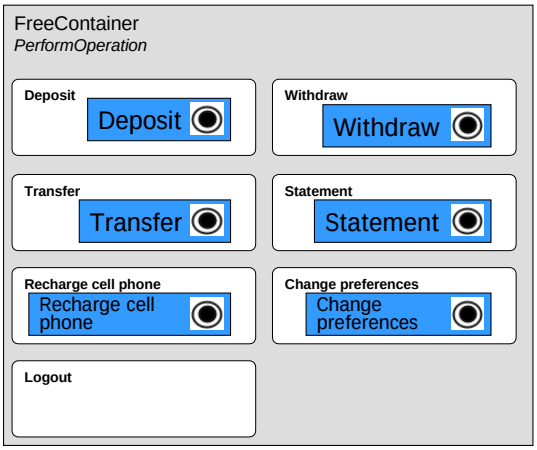


Figura 6.40 Objetos abstractos de interacción para las tarea abstracta *Perform Operation*.

Todas las tareas contenidas dentro de la tarea abstracta *Perform Operation*, excepto *Logout*, necesitan algún elemento que sea capaz de disparar su ejecución en el correspondiente *FreeContainer* asociado. Por lo tanto, a los contenedores asociados a dichas tareas dentro del *FreeContainer Perform Operation* se ha añadido un *ActionInvoker* que permita invocar el comportamiento necesario para que se dispare la ejecución de la tarea asociada a ese contenedor. Los componentes derivados para dicho propósito se ilustran en la figura 6.40.

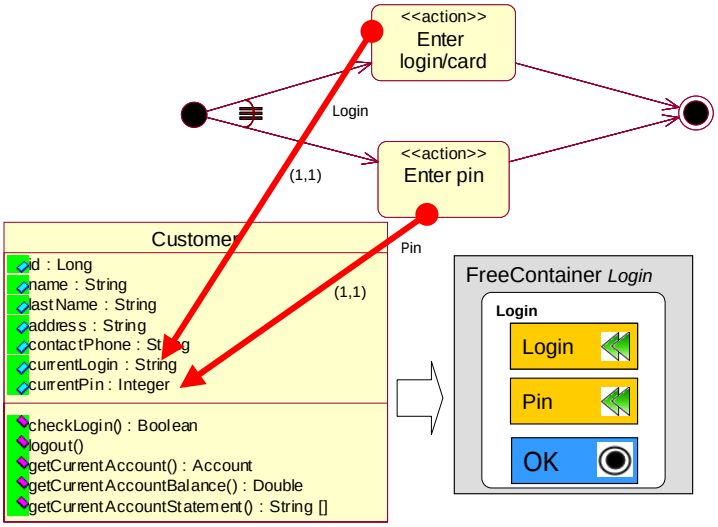


Figura 6.41 Objetos abstractos de interacción para la tarea *Login*.

Para la tarea *Login* se han derivado los objetos abstractos de interacción que aparecen en la figura 6.41. Para cada una de las acciones de entrada se ha derivado un *Inputter*. Para permitir la la transición *start* para confirmar el *Login* (véase la figura 6.30) se ha derivado un *ActionInvoker* que producirá la transición y la comprobación de las poscondiciones de la tarea *Login*.

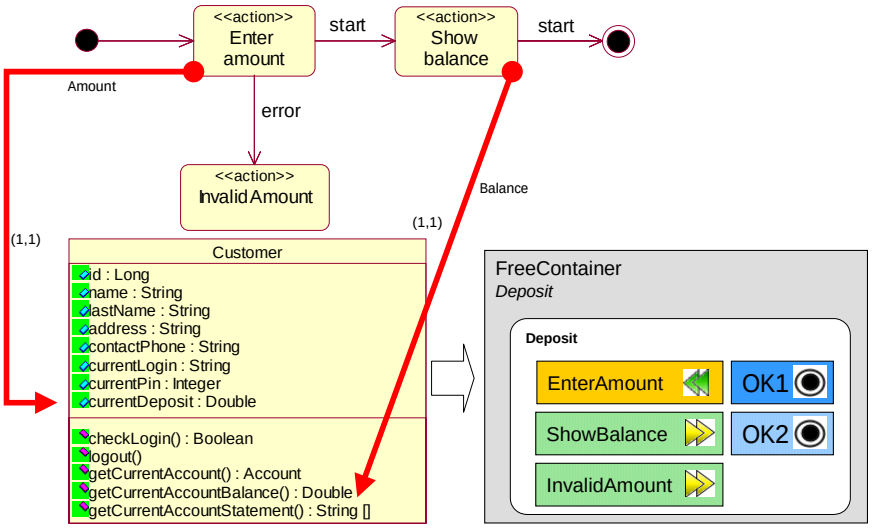


Figura 6.42 Objetos abstractos de interacción para la tarea *Deposit*.

La tarea *Deposit* consta de tres acciones: *Enter amount*, *Show balance* e *Invalid Amount* (véase la figura 6.42). Para la primera acción de entrada, *Enter*

amount, se ha derivado un *Inputter*, mientras que para las dos acciones de salida, *Show balance* e *InvalidAmount* se han derivado sus correspondientes *Displayers*. Sin embargo, obsérvese como dichos *Displayers* estarán inicialmente desactivados (en la figura se han representado los elementos desactivados con una trama con el color original atenuado), ya que para que el usuario pueda interactuar con ellos tiene que dispararse las transiciones correspondientes, y por lo tanto tendrá que ser la ejecución de las transiciones las que los activen. Finalmente se han derivado dos *ActionInvoker* para permitir disparar las dos transiciones de tipo *start* del modelo de tareas de *Deposit*.

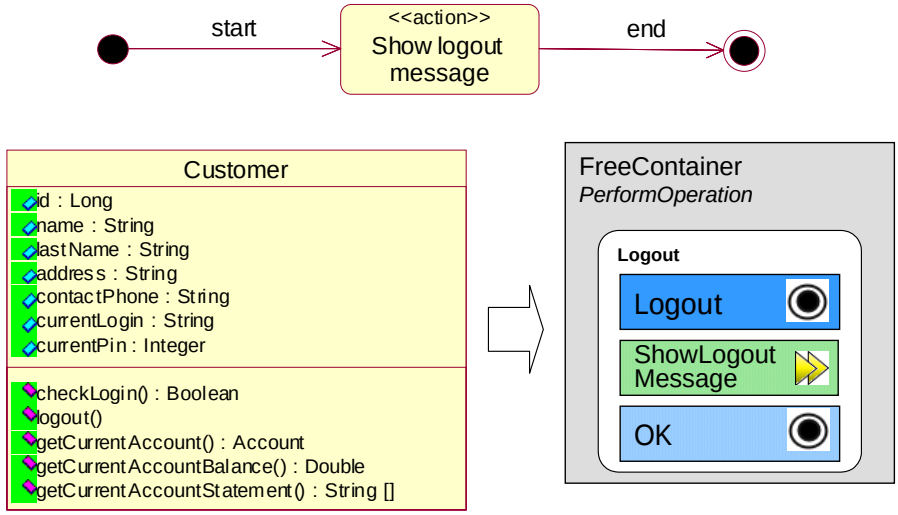


Figura 6.43 Objetos abstractos de interacción para la tarea *Logout*.

De forma análoga, se han derivado los objetos abstractos de interacción para la tarea *Logout* (véase la figura 6.43). Obsérvese como el *Displayer* derivado a partir de la acción *Show logout message* estará inicialmente desactivado, y será activado hasta que no se tome la transición *start*, disparado por el *ActionInvoker Logout*.

Finalmente, en la figura 6.44 se muestran los objetos abstractos de interacción derivados para la tarea *Edit preferences*. Las tres acciones de esta tarea deben permitir la edición de los tres objetos de interacción especificados, por lo cual se han derivado sus correspondientes *Editors*. Un *ActionInvoker* ha sido derivado para permitir que se pueda producir la transición *start* que termine la modificación de las preferencias del usuario.

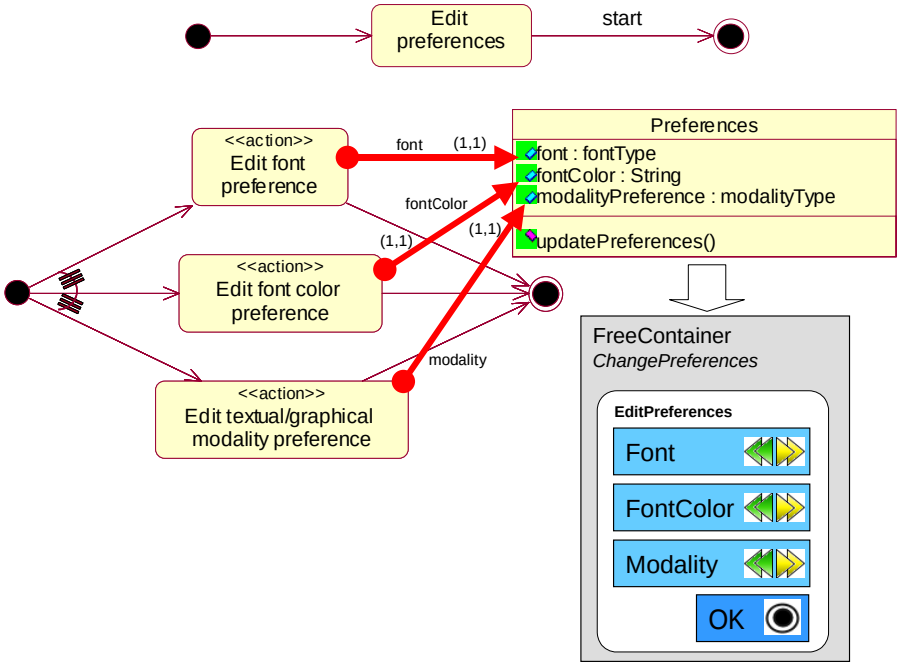


Figura 6.44 Objetos abstractos de interacción para la tarea *Edit preferences*.

6.3.3.4 La interfaz de usuario concreta

Múltiples interfaces concretas pueden ser derivada a partir de una misma interfaz de usuario abstracta, dependiendo de las correspondencias que se establezcan. Una misma acción podrá ser realizada usando distintos componentes concretos. Las heurísticas para la elección de unos u otros componentes concretos se basan principalmente en la especificación de los objetos de interacción que describe detalladamente el tipo de datos que cada acción manipula. Por ejemplo, para presentar una acción de entrada donde el objeto de interacción sea booleano, se podría usar un botón de radio, una casilla de verificación, una lista desplegable, o simplemente un campo de texto donde el usuario escriba "verdadero" o "falso".

A continuación se muestran las interfaces de usuario concretas derivadas para las tareas que han sido detalladas a lo largo de este caso de estudio

Derivación de la estructura de contenedores concretos

Para cada uno de los *FreeContainers* definidos en la interfaz de usuario abstracta se ha derivado una ventana, mientras que para cada *Container* se

ha derivado una caja *(box)* que permita la agrupación de los componentes relacionados de una tarea.

Derivación de los objetos concretos de interacción

La derivación de los objetos concretos de interacción incluye tanto la generación de la presentación estática de la interfaz como de su comportamiento. Las precondiciones y poscondiciones de las tareas y acciones deben convertirse en eventos en la interfaz de usuario concreta. Los tipos de los eventos son inicialmente generados por defecto, aunque el diseñador puede modificarlos en cualquier momento.

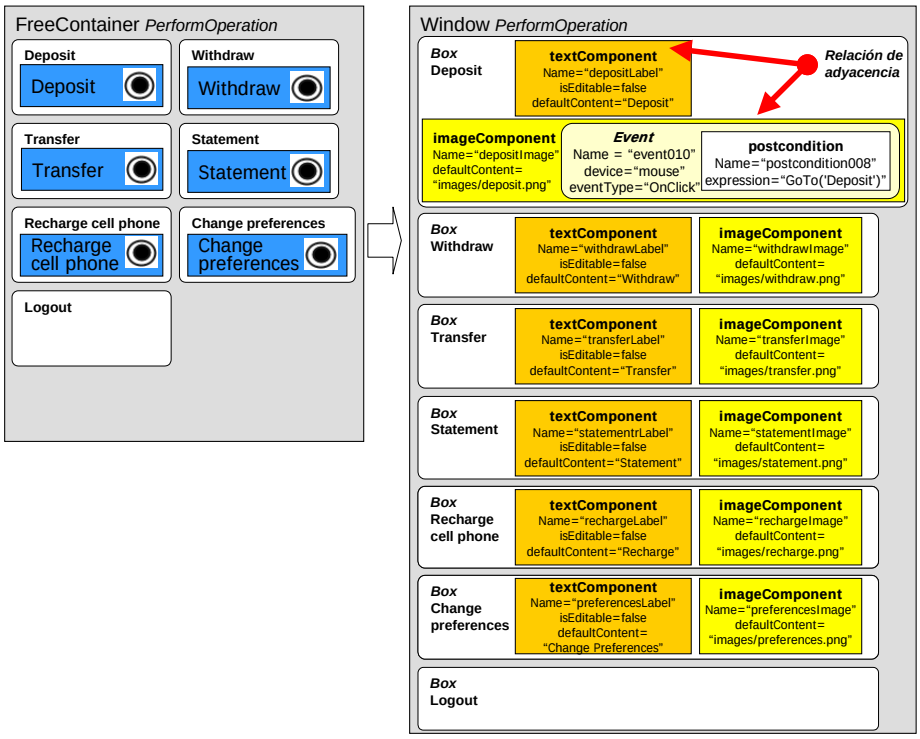


Figura 6.45 Objetos concretos de interacción para la tarea *Perform Operation*.

En la figura 6.45 se ilustra la derivación de los objetos de interacción concretos para la tarea *Perform Operation*. Para cada *ActionInvoker* se debe generar un conjunto de elementos de la interfaz de usuario capaces de perrmitir al usuario disparar un comportamiento. Para ello se pueden usar botones, enlaces de texto, o como en este caso, una etiqueta y una imagen que actúa como botón. A cada imagen se le ha asociado un comportamiento (en forma de evento), el cual produce la navegación

usando la función predefinida *GoTo*. Para facilitar la compresión de la figura sólo se ha incluido el comportamiento para la imagen *depositImage*, ya que la derivación de los objetos concretos de interacción para el resto de las imágenes sería totalmente análoga. Para facilitar la comprensión de la figura también se han omitido los objetos concretos de interacción para la tarea *Logout*. Dichos elementos se encuentran descritos en la figura 6.46.

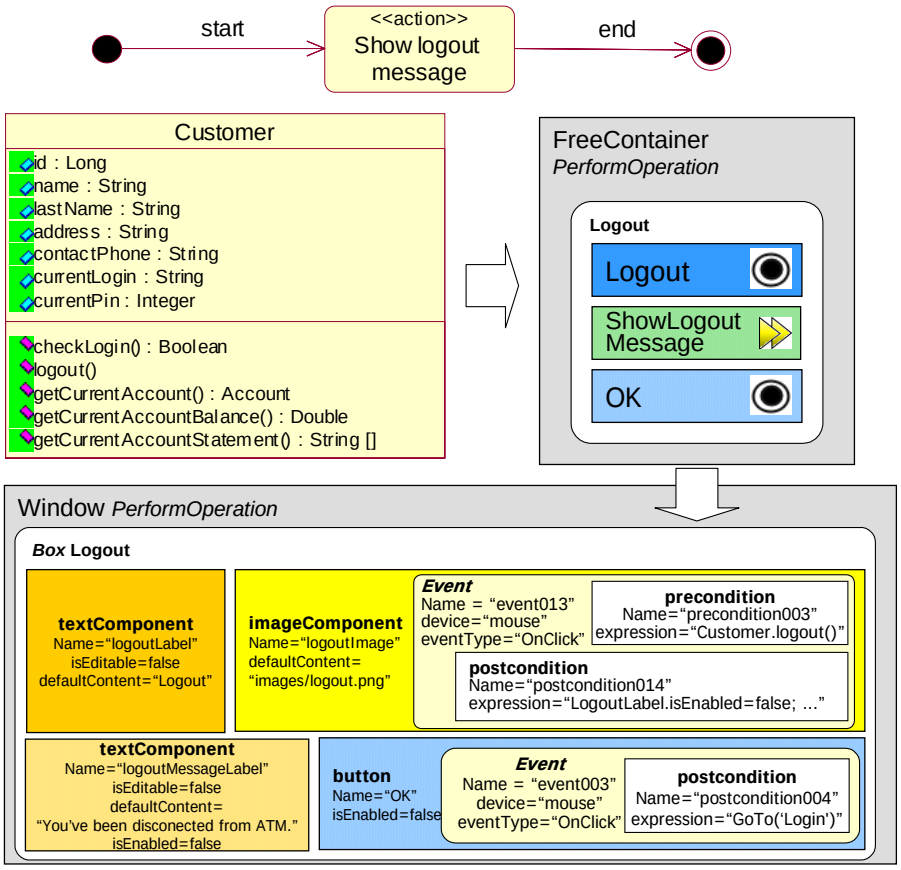


Figura 6.46 Objetos concretos de interacción para la tarea *Logout*.

Obsérvese como se ha reflejado el comportamiento descrito en el modelo de tareas para la tarea *Logout* en la figura 6.46. Cuando se pulsa en la imagen *logoutImage* primero se desconectará al usuario del sistema (precondición *precondition003)* y después se activará el mensaje de despedida y el botón para que el usuario comunique que ya ha leído el mensaje de despedida. La etiqueta *logoutLabel* y la imagen *logoutImage* serán desactivadas, puesto que dicha tarea no debe estar ya disponible.

Finalmente, cuando el usuario pulse el botón para comunicar que ya ha leído el mensaje de despedida se devolverá el control a la tarea *Login*.

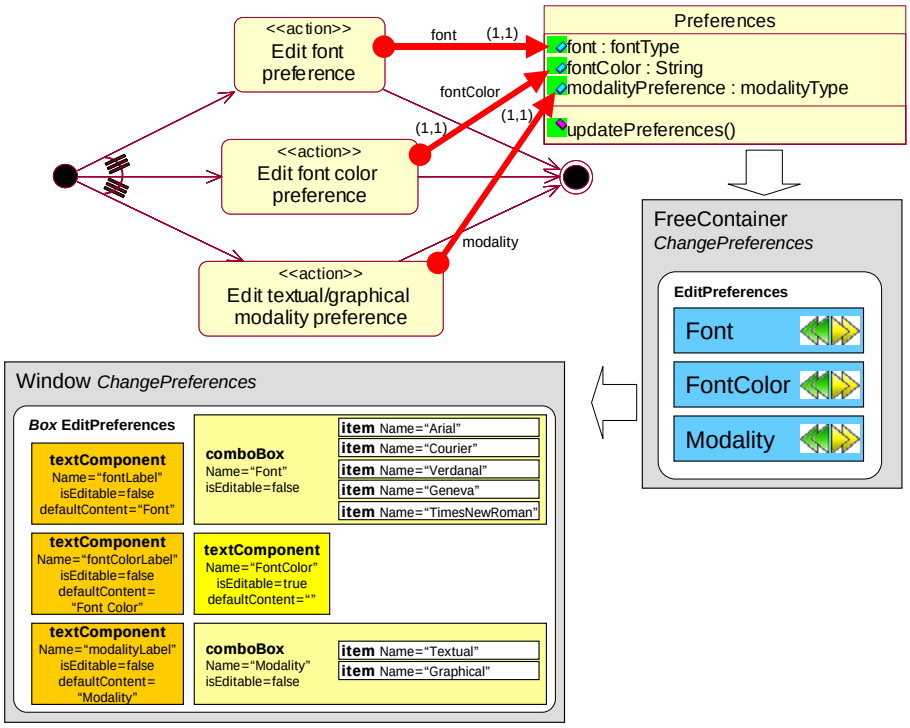


Figura 6.47 Objetos de interacción concretos para la tarea *Edit Preferences*.

Los objetos de interacción concretos derivados para la tarea *Edit preferences* se detallan en la figura 6.47. Obsérvese como en este caso tanto para el objeto de interacción *Font* como para el objeto de interacción *modality* se han escogido listas deplegables *(comboBox)*, ya que en este caso el usuario tendrá que elegir entre un pequeño conjunto de valores (aquellos conjuntos de valores definidos en la especificación de los tipos enumerados disponibles en el modelo de dominio). Otra posibilidad es la sustitución de las listas desplegables por grupos de botones de radio. La principal diferencia entre ambas posibilidades es el espacio que ocupan en pantalla y la cantidad de información visible simultáneamente. Los tres controles de entrada *(Font*, *FontColor* y *Modality)* tendrán asociados sus correspondiente eventos para garantizar que el valor introducido por el usuario es almacenado en el elemento del modelo de dominio con el que han sido relacionados en la definición de los objetos de interacción.

6.3.3.5 Reglas de adaptación

A continuación se describen algunas de las reglas de adaptación especificados para ATM UI, así como ejemplo de evaluación de dichas reglas. La regla descrita a continuación permite acomodar la presentación generada para la tarea *Edit preferences* cuando se produce un cambio en el área de presentación disponible debido al cambio de plataforma desde el cajero automático del banco a la plataforma móvil.

Sensores

En la figura 6.48 se muestra la especificación de un sensor que permite al sistema detectar los cambios en la resolución del dispositivo. En este caso se indica que la resolución es del tipo "_2x2_", y que la información será enviada por el sensor en el atributo *screenResolution* en el elemento cuya ruta dentro del fichero XML es:

/contextInfo/platformInfo/hardwarePlatform/screenResolution

```
<softwarePlatformSensor id="SPS001" name="screenResolutionPlatformSensor"
        dataName="/contextInfo/platformInfo/hardwarePlatform/screenResolution"
        dataType="_2x2_"
        description="Senses whenever screen resolution changes."/>
```

Figura 6.48 Especificación de un sensor para ATM UI.

Eventos del contexto

Los eventos del contexto son producidos por los sensores, y serán los que finalmente disparen las reglas de adaptación. La figura 6.49 describe la especificación de un evento del contexto *(contextEvent)* basado en el sensor anteriormente descrito.

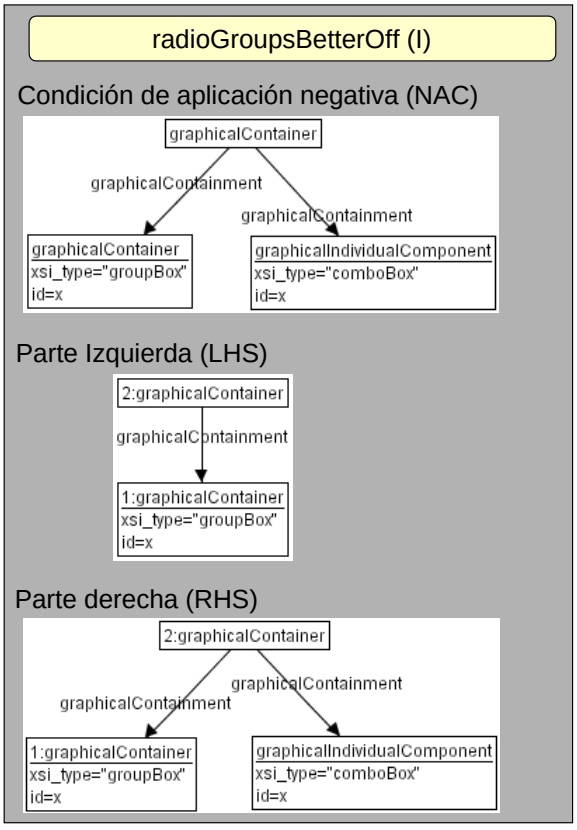
```
<contextEvent id="CE001" name="screenResolutionChangedContextEvent"
        description="Produced when screen resolution changes.">
        <contextEventSensors>
            <sensor id="SPS001"/>
        </contextEventSensors>
</contextEvent>
```

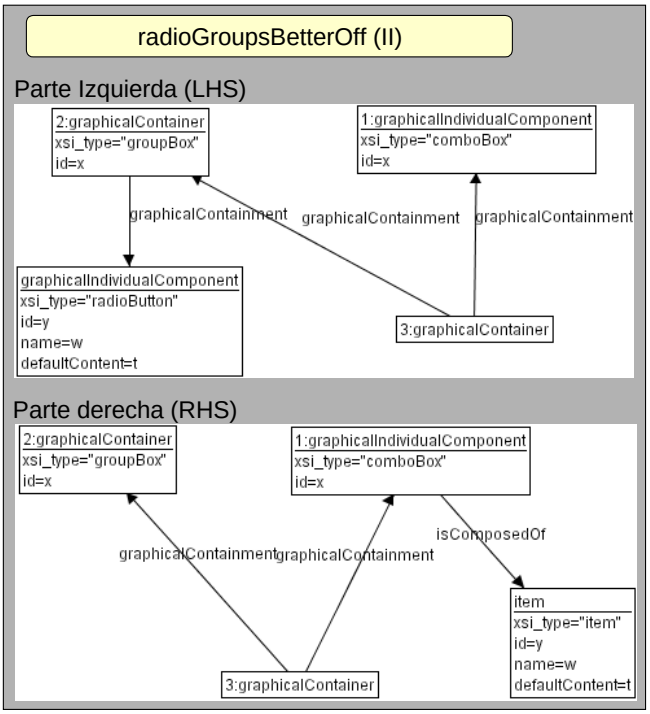
Figura 6.49 Especificación de un evento del contexto para ATM UI.

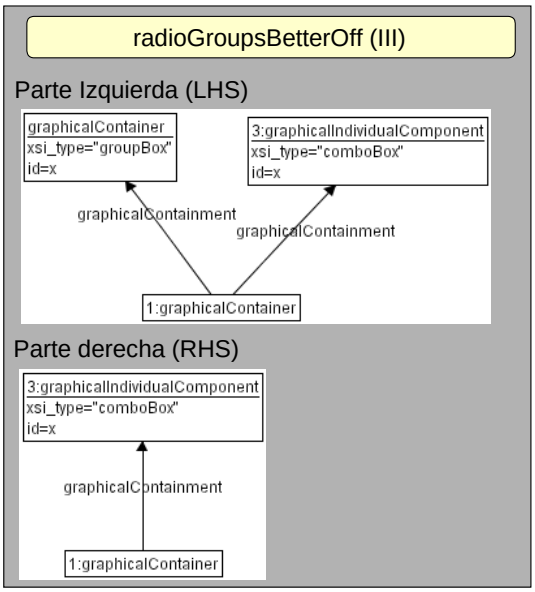
Reglas de adaptación

A continuación se describen dos de las reglas de adaptación aplicables cuando se produce el evento del contexto mostrado en la figura 6.49.

```
<adaptationRule id="AR001" name="radioGroupsBetterOff"
    description="When screen resolution is reduced, tries to fit the presentation on
                 screen by replacing radio buttons groups with comboboxes.">
    <contextEvents>
        <contextEvent id="CE001"/>
    </contextEvents>
    <contextPrecondition id="CP001" name="screenResolutionShrinked">
        <logicalSentence id="LS001" name="OldGreaterThanNew"
                         sentence="SPS001.oldValue > SPS001.newValue "/>
    </contextPrecondition>
    <transformation>
```





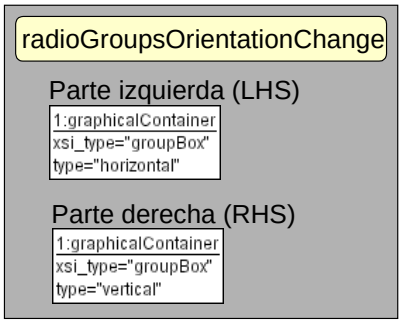



```
        </transformation>
</adaptationRule>
```

Figura 6.50 Regla de adaptación para ATM UI.

La figura 6.50 describe una regla de adaptación que transforma un grupo de botones de radio en una lista desplegable, de forma que se reduzca el espacio necesario para presentar los datos al usuario. La regla es disparada por el evento del contexto descrito en la figura 6.49. Aunque, según se especifica en la precondición del contexto *(contextPreconditon)* la regla sólo se disparará cuando la nueva resolución sea menor que la original. Obsérvese como la transformación en esta ocasión está compuesta por tres pasos que se aplicarán de forma secuencial. En el primer paso se creará una lista deplegable *(comboBox)* por cada grupo de botones. En el segundo paso se convertirá cada uno de los botones de radio *(radioButton)* del grupo de botones de radio *(groupBox)* en una opción *(item)* de la lista desplegable. Finalmente, se eliminará el grupo de botones de radio transformado.

```
<adaptationRule id="AR002" name="radioGroupsOrientationChange"
                description="When screen resolution is reduced, tries to fit the
                            presentation on screen by changing the orientation of the
                            radio button to 'vertical'." >
    <contextEvents>
        <contextEvent id="CE001"/>
    </contextEvents>
    <contextPrecondition id="CP001" name="screenResolutionShrinked">
        <logicalSentence id="LS001" name="OldGreaterThanNew"
                         sentence="SPS001.oldValue > SPS001.newValue "/>
    </contextPrecondition>
    <transformation>
```




```
    </transformation>
</adaptationRule>
```

Figura 6.51 Regla de adaptación para ATM UI.

La regla *radioGroupsOrientationChange* mostrada en la figura 6.51 es aplicable cuando se produce el evento del contexto mostrado en la figura 6.49 y la nueva resolución es menor que la original. En este caso la regla simplemente cambiará la orientación de los botones de radio para que se

sitúen de forma vertical, aunque ello puede hacer que el usuario tenga que usar la barra de desplazamiento para poder acceder a toda la información.

6.3.4 Implementación

Tras la finalización de la fase de diseño se debe producir una aplicación final ejecutable. Para ello será necesario convertir la especificación de la interfaz de usuario concreta en el lenguage destino. En la figura 6.52 se muestra el aspecto de la interfaz de usuario para la tarea *Perform operation* y la interfaz de usuario para la tarea *Change preferences* renderizadas en el lenguaje XUL usando la herramienta desarrollada *usiXMLTransformer*.

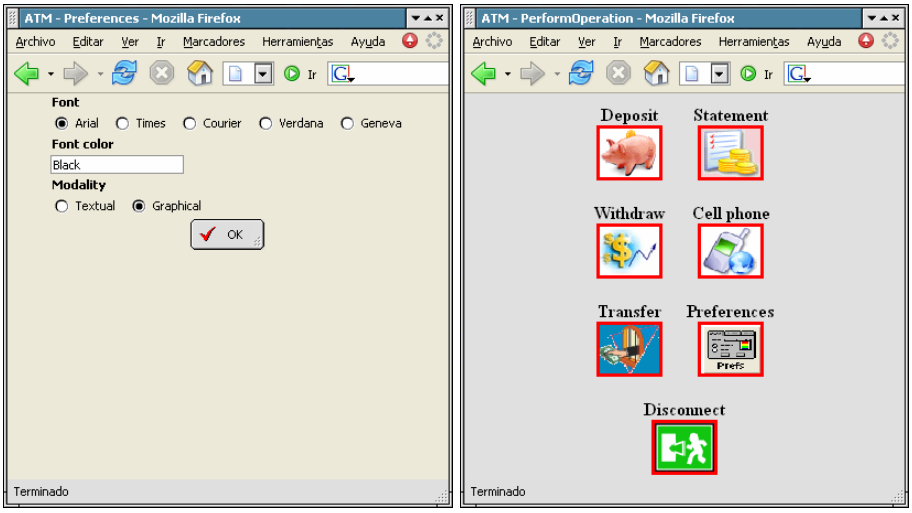


Figura 6.52 Renderización en XUL de las tareas *Change preferences* y *Perform Operation*.

Integración de las capacidades de adaptación en el motor de adaptación

Las reglas de adaptación diseñadas serán convertidas en planes para el agente *AgentAdaptationProcess* mediante la generación de código para el lenguaje de programación de sistemas multi-agente usado: *JACK* [Bus99]. En tiempo de ejecución será el sistema multi-agente el que decida qué adaptación aplicar mediante la evaluación de las adaptaciones aplicables en cada ocasión.

En la figura 6.53 se puede observar el resultado de la aplicación de las dos reglas de adaptación descritas en el apartado 6.3.3.5 sobre la interfaz de usuario creada para la tarea *Edit preferences* (modificar las preferencias). Las adaptaciones han sido renderizadas para la plataforma PDA, es decir,

reduciendo la resolución a 220x144, tal y como se especificó en el modelo de plataforma. Obsérvese como en el primer caso se ha cambiado la orientación de los botones de radio para intentar reducir la anchura de la presentación, mientras que en el segundo caso se han sustituido los grupos de botones de radio por listas desplegables que ocupan un menor espacio.

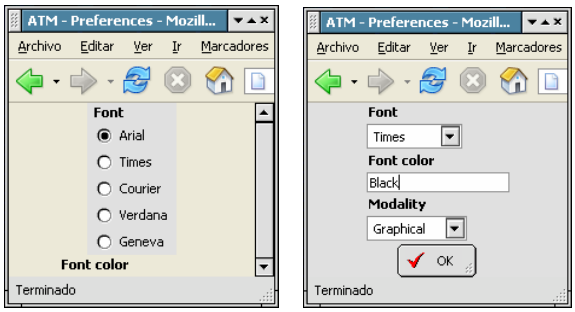


Figura 6.53 Ejemplos de aplicación de las reglas de adaptación descritas para ATM UI.

6.4 Conclusiones del capítulo

Dentro de este capítulo se han presentado dos casos de estudio que ejemplifican el proceso seguido dentro del método de diseño de interfaces de usuario propuesto: AB-UIDE. A lo largo de la presentación de los casos de estudio también se ilustra el efecto de la aplicación de las adaptaciones dieñadas sobre las interfaces de usuario de los dos casos de estudio.

CAPÍTULO 7
CONCLUSIONES Y TRABAJO FUTURO

"Me interesa el futuro porque es el sitio donde voy a pasar el resto de mi vida."
(Woody Allen)

7.1 Conclusiones

El trabajo realizado se encuadra dentro del grupo LoUISE (Laboratory of User Interfaces and Software Engineering), donde se ha plantea un esfuerzo conjunto en la mejora de la calidad del software, y especialmente una mejora en la calidad de las interfaces de usuario diseñada, mediante la mejora de su usabilidad.

Para ello, durante estos últimos años se ha ahondado en el estudio de propuestas que permitan mejorar la usabilidad del sistema desde distintos frentes. En sentido, se han realizado interesantes trabajos en la mejora de la usabilidad de las interfaces de usuario mediante la incorporación de la experiencia [Mon02][Mon03][Mon05][Mon05b], el estudio del diseño de interfaces de usuario post-WIMP [Mol02][Mol03][Mol03a]Mol05, soluciones metodológicas para el diseño de interfaces de usuario [Loz00], y finalmente la mejora de la usabilidad del sistema a través de la adaptación del sistema a las características del contexto de uso (usuario, plataforma, entorno y tarea actual) y sus cambios [Lop03a][Lop03b][Lop04][Lop05]. Es este último apartado, donde esta tesis se encuadra, dando una solución al diseño de interfaces de usuario capaces de adaptarse al contexto de uso donde están siendo utilizadas de forma automática, y manteniendo su usabilidad.

Dentro de la línea de investigación arriba esbozada, se han abordado los principales retos que se presentan para alcanzar la meta perseguida.

Uno de los puntos en los que se realizan nuevas aportaciones es en diseño de interfaces de usuario basado en modelos. Aunque el diseño basado en

modelo comienza a ser una tecnología madura, como demuestra la disponibilidad de algunas herramientas comerciales como Olivanova, VisualWADE o webML, sólo contemplan todavía algunos aspectos de personalización, pero sus capacidades de adaptación son muy limitadas. Es por ello que dentro de este trabajo se propone el enriquecimiento de los métodos de diseño de interfaces de usuario basados en modelos con los resortes necesarios para facilitar al diseñador la especificación de las capacidades de adaptación de la interfaz de usuario. Para ello se ha propuesto:

- Un modelado del contexto apropiado que permite relacionar cómo influyen unas características del contexto en otras.

- Una especificación de cómo se adquieren los datos del contexto a través del modelado de sensores, los cuales especifican cómo se capturan los datos, qué tipo de datos reciben, y cómo deben ser tratados por le motor de adaptación.

- Un metamodelo de regla de adaptación que permite al diseñador describir cómo se adapta la interfaz de usuario de acuerdo a los cambios del contexto detectados a través de los sensores.

- Una aproximación para la especificación de un modelo de calidad en tiempo de ejecución (compromiso de usabilidad) que garantiza que para cada plataforma no se aplicarán reglas de adaptación inapropiadas de acuerdo al compromiso de usabilidad especificado por el diseñador para cada plataforma.

- Un modelo de tareas que permite la especificación del diálogo entre el usuario y la interfaz de usuario.

- Se propone un modelo abstracto de interfaz de usuario, y unas heurísticas de transformación, a partir del modelo de tareas propuesto y sus relaciones con el modelo de dominio, a dicho modelo. De igual manera, se describe el paso desde el modelo abstracto propuesto a un modelo concreto de interfaz de usuario.

- El método de diseño de interfaces de usuario propuesto permite la trazabilidad desde los requisitos a la implementación de la interfaz. Es más, mediante el diagrama de conectores propuesto dicha

trazabilidad, establecida mediante la creación de relaciones entre los diferentes modelos usados en las distintas fases del desarrollo, puede ser visualizada gráficamente, y refinada o modificada.

- Se ha implementado una herramienta, *usiXMLTransformer*, capaz de tomar una especificación en el lenguaje de especificación de interfaces de usuario usiXML, aplicarle una serie de reglas de adaptación, y convertir la especificación adaptada de nuevo a usiXML, a una presentación final en el lenguaje XUL o a una presentación final en el lenguaje Java.

La ejecución de interfaces de usuario adaptativas plantea de igual manera, grandes retos dentro de la comunidad investigadora. La plataforma de ejecución de dichas interfaces debe ser capaz de detectar los cambios en el contexto de uso, seleccionar las adaptaciones aplicables dados los cambios detectados y las necesidades del usuario, elegir qué adaptación es más apropiada en cada momento entre las aplicables, y finalmente aplicarla. En este punto, dentro de esta tesis se ha propuesto una arquitectura de ejecución de interfaces de usuario basada en el concepto de agente, la cual propone un sistema multi-agente, donde los agentes colaboran para proporcionar las capacidades de adaptación necesarias. Dentro de este campo se ha contribuido con:

- El diseño de un sistema multi-agente para la ejecución de interfaces de usuario adaptativas.

- La integración del conocimiento adquirido durante el desarrollo de la interfaz de usuario dentro de un motor de ejecución de interfaces de usuario adaptativas, el cual es creado para cada aplicación a partir de la especificación que el diseñador crea para las capacidades de adaptación.

- Se ha diseñado un método de evaluación de la bondad de las adaptaciones plausibles en cada momento, de acuerdo al coste de migración y el beneficio de adaptación, y que además es refinada en tiempo de ejecución de acuerdo a un método de aprendizaje Bayesiano.

Por lo tanto, se ha aportado una solución completa al diseño e implementación de interfaces de usuario adaptativas que cubre todo el

ciclo de desarrollo de la interfaz de usuario, que abarca desde las fases de adquisición de requisitos a la fase de implementación y despliegue.

Fruto de la estancia de siete meses realizada en el grupo BCHI (*Belgian Computer-Human Interaction*), y especialmente de la colaboración con dos de sus miembros Quentin Limbourg y Jean Vanderdonckt, se ha participado en el desarrollo del lenguaje de especificación de interfaces de usuario usiXML, y de ha desarrollado en colaboración con Quentin Limbourg *transformiXML*[39], una herramienta de transformación de especificaciones de usuario mediante transformación de grafos.

7.2 Publicaciones realizadas

A lo largo de los últimos cinco años se han realizado numerosas contribuciones en conferencia eminentemente internacionales dentro de los campos de la Ingeniería del Software, la Interacción Hombre-Máquina y los Sistemas Multi-agente, que a continuación se detallan.

Una descripción de la propuesta metodológica en su conjunto presentada para AB-UIDE puede encontrarse en:

- López-Jaquero, V., Montero, F., Molina, J.P., González, P., Fernández-Caballero, A. A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties. Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005.

El modelo de conectores usado para mantener la trazabilidad entre las distintas etapas, mostrando visualmente las correspondencias entre los modelos está descrito en:

- López Jaquero, V., Montero, F., Fernández Caballero, A., Lozano, M.D. Towards Adaptive User Interfaces Generation: One Step Closer to People. En Enterprise Information Systems V. Kluwert Academia Publishers, Dordrecht, Holanda, 2004. pp. 226-232. ISBN: 1-4020-1726-X.

- López Jaquero, V., Montero, F., Molina, J.P., Fernández-Caballero, A., González, P. Model-Based Design of Adaptive User Interfaces through Connectors. Design, Specification and Verification of Interactive Systems 2003,

[39] http://www.usixml.org/index.php?view=page&idpage=19

DSV-IS 2003. In *DSV-IS 2003 : Issues in Designing New-generation Interactive Systems Proceedings of the Tenth Workshop on the Design, Specification and Verification of Interactive Systems*. J.A. Jorge, N.J. Nunes, J. F. Cunha (Eds). Springer Verlag, LNCS 2844, 2003. Madeira, Portugal June 4-6, 2003.

- López Jaquero, V., Montero, F., Fernández, A., Lozano, M. Towards Adaptive User Interface Generation: One Step Closer To People. 5th International Conference on Enterprise Information Systems, ICEIS 2003. Proccedings of 5th International Conference on Enterprise Information Systems, ICEIS 2003, vol. 3, pp. 97-103. Angers, France, April 23-26, 2003.

El modelo de aplicación de las adaptaciones sobre la especificación de la interfaz de usuario mediante transformaciones de grafos se describe en:

- Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., López-Jaquero, V., UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 207-228.

La arquitectura multi-agente presentada para la ejecución de interfaces de las interfaces de usuario adaptativas creadas siguiendo la propuesta de AB-UIDE se encuentra descrita en:

- López-Jaquero, V, Montero, F., Molina, J.P., González, P., Fernández-Caballero, A. A Multi-Agent System Architecture for the Adaptation of User Interfaces. 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005). 15-17 September 2005, Budapest, Hungary. In *Multi-Agents Systems and Applications IV*. M. Pechoucek, P. Petta, L. Zsolt Varga (Eds.) LNAI 3690, Springer-Verlag, Berlin.

- López-Jaquero, V., Fernández-Caballero, A. Métricas de Usabilidad y Sistemas Multiagente en Hipermedia Adaptativa. XIII Escuela de Verano de Informática. Tendencias Actuales en la Interacción Persona-Ordenador: Accesibilidad, Adaptabilidad y Nuevos Paradigmas. ISBN: 84-921873, pp. 21-34, Albacete, España, 2003.

- Fernández-Caballero, A., López Jaquero, V., Montero, F. , González, P. Adaptive Interaction Multi-agent Systems in E-learning/E-teaching on the Web. International Conference on Web Engineering, ICWE 2003. In *Web Engineering: International Conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003. Proceedings*. J.M. Cueva Lovelle, B.M. González Rodríguez, L. Joyanes Aguilar, J.E. Labra Gayo, M. del Puerto Paule Ruiz (Eds.). Springer Verlag, LNCS 2722, pp. 144-154. ISSN:0302-9743. Oviedo, Spain, June, 2003.

- López Jaquero, V., Montero, F., Fernández, A., Lozano, M. Usability Metrics in Agent-Based Intelligent Tutoring Systems. Human-Computer Interaction: Theory and Practice (part 1). J. Jacko, C. Stephanidis (Eds.). Lawrence Erlbaum Associates. Londrés, Reino Unido, 2003. ISBN: 0-8058-4931-9. pp. 539-543.

Distintos casos de uso donde se ha experimentado con las capacidades de adaptación en distintos tipo de entornos, y especialmente con entornos de aprendizaje y virtuales, se encuentras descritas en:

- Robles, A., Molina, J. P., López-Jaquero, V., García, A. S. Even Better Than Reality: The Development of a 3-D Online Store that Adapts to Every User and Every Platform. HCI International 2005, Las Vegas, Nevada, USA, July, 2005. Volume 7 - Universal Access in HCI: Exploring New Interaction Environments.

- López-Jaquero, V., Fernández-Caballero, A., Montero, F., Molina, J.P., González, P. Towards Adaptive E-learning / E-teaching on the Web. International Conference on Technology-Enhanced Learning (TEL 2003). Procedings of International Conference on Technology-Enhanced Learning (TEL 2003). Milán, Italia, noviembre, 2003.

- González, P., Montero, F., López Jaquero, V., Fernández, A., Montañés, J., Sánchez, T. A Virtual Learning Environment for Short Age Children. IEEE International Conference on "Advanced Learning Technologies", ICALT 2001. Proccedings of the IEEE International Conference on Advanced Learning Technologies, ICALT 2001, Okamoto, T., Hartley, R., Kinshuk, Klus, J. (eds.). IEEE Computer Society, Los Alamitos, CA., Agosto 2001, pp. 283-285. ISBN:0-7695-1013-2. Madison, USA, August 6-8, 2001.

7.2.1 Publicaciones relacionadas realizadas

Fruto del trabajo realizado durante estos cinco años de investigación, y de la colaboración con otros miembros del grupo de investigación LoUISE, otros grupos nacionales, como GRIHO[40], e internacionales como el BCHI[41] (*Belgian Laboratory of Computer-Human Interaction*) se han realizado distintas contribuciones a conferencias, eminentemente internacionales, relacionadas con la investigación actual y los incipientes trabajos futuros.

Dentro de la integración de los patrones dentro del proceso de desarrollo de interfaces de usuario, y su futura inclusión en el desarrollo de interfaces

[40] http://griho.udl.es/castella/index.html
[41] http://www.isys.ucl.ac.be/bchi/

de usuario con capacidades de adaptación se han realizado las siguientes contribuciones:

- Montero, F., López-Jaquero, V., Ramírez, Y., Lozano, M., González. P. Patrones de Interacción: para usuarios y para diseñadores. VI Congreso de Interacción Persona-Ordenador. 13-16 de septiembre, Granada, España. 2005.

- Montero, F., López-Jaquero, V., Molina, J.P., Lozano, M. Improving e-shops environments by usign usability patterns. 2nd workshop on software and usability cross-polination. The role of usability patterns. September, 1-2, 2003, Zürich, Switzerland. 2003.

De igual manera se han realizado contribuciones en la búsqueda de la incorporación de mejoras en el diseño de interfaces de usuario y su evaluación:

- Montero, F., López-Jaquero, V., Vanderdonckt, J., González, P., Lozano, M.D., Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. 12th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'2005), Newcastle upon Tyne, England, July 13-15, 2005. Springer-Verlag, Berlin, 2005 (in print).

- Montero, F., López-Jaquero, V., Lozano, M., González, P. A User Interfaces Development and Abstraction Mechanism. Artículo seleccionado en el V Congreso Interacción Persona Ordenador para su publicación en Springer-Verlag, Berlin, 2005 (in print).

- Montero, F., López-Jaquero, V., Lozano, M., González, P. De Platón al desarrollo de Interfaces de Usuario. V Congreso Interacción Persona Ordenador. 3 - 7 de mayo de 2004, Lérida, Spain, 2004.

- Sendín, M., Lorés, J., Montero, F., López-Jaquero, V. Using Reflection on Dynamic Adaptation of User Interfaces. 4th International Workshop on Mobile Computing. IMC Workshop. Assistance, Mobility, Applications. Rostock, Germany, 2003. Human-Computer Interaction with Mobile Devices and Services. Springer Verlag, LNCS 2895, 2003. ISBN: 3-540-40821-5. pp. 428 - 433.

- Sendín, M., Lorés, J., Montero, F., López Jaquero, V., González, P. User Interfaces: A Proposal for Automatic Adaptation. International Conference on Web Engineering, ICWE 2003. In *Web Engineering: International Conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003. Proceedings.* J.M. Cueva Lovelle, B.M. González Rodríguez, L. Joyanes Aguilar, J.E. Labra Gayo, M. del Puerto Paule Ruiz (Eds.). Springer Verlag, LNCS 2722. Oviedo, Spain, July 14-18, 2003.

- Montero, F., Lozano, M., López-Jaquero, V., González, P. A Quality Model For Testing The Usability Of The Web Sites. 10th International Conference on Human - Computer Interaction (HCI, 2003). 22 a 27 de Julio, Creta, Grecia. 2003. Human-Computer Interaction: Theory and Practice (part 1). J. Jacko, C. Stephanidis (Eds.). Lawrence Erlbaum Associates. Londrés, UK, 2003.

- Montero, F., López Jaquero, V., Molina, J.P., González, P. An approach to develop User Interfaces with plasticity. Design, Specification and Verification of Interactive Systems 2003, DSV-IS 2003. In *DSV-IS 2003 : Issues in Designing New-generation Interactive Systems Proceedings of the Tenth Workshop on the Design, Specification and Verification of Interactive Systems*. J.A. Jorge, N.J. Nunes, J. F. Cunha (Eds). Springer Verlag, LNCS 2844, 2003. Madeira, Portugal June 4-6, 2003.

- Molina, J.P., González, P., Lozano, M.D., Montero, F., López Jaquero, V. Bridging the gap: developing 2D and 3D user interfaces with the IDEAS methodology. Design, Specification and Verification of Interactive Systems 2003, DSV-IS 2003. In *DSV-IS 2003 : Issues in Designing New-generation Interactive Systems Proceedings of the Tenth Workshop on the Design, Specification and Verification of Interactive Systems*. J.A. Jorge, N.J. Nunes, J. F. Cunha (Eds). Springer Verlag, LNCS 2844, 2003. Madeira, Portugal June 4-6, 2003.

- Montero, F., Lozano, M., López Jaquero, V., González, P. Usability And Web Site Evaluation: Quality Models And User Testing Evaluations. 5th International Conference on Enterprise Information Systems, ICEIS 2003. Proccedings of 5th International Conference on Enterprise Information Systems, ICEIS 2003, vol. 1, pp. 525-238. Angers, France, April 23-26, 2003.

- López Jaquero, V., Montero, F., Fernández, A., Lozano, M. Adaptabilidad de Interfaces de Usuario por Reflexión. III Jornadas de Trabajo Dolmen. Actas de III Jornadas de Trabajo Dolmen, 2002, pp. 65-70. El Escorial, Spain, November 20, 2002.

7.3 Trabajo futuro

Los resultados desarrollados en esta tesis dejan un el camino abierto a nuevas líneas de trabajo que complementen el trabajo realizado.

Una de las direcciones que se antojan interesantes es el desarrollo de interfaces de usuario adaptativas colaborativas. La adaptación de varias interfaces de usuario que deben estar coordinadas plantea retos en su modelado de gran interés.

Aunque dentro del trabajo se ha propuesto un metamodelo para la especificación de reglas de adaptación, desgraciadamente no existe

ninguna herramienta que permite diseñar las adaptaciones de forma visual y sencilla para el diseñador. El diseño de la teoría y la herramienta capaz de permitir el diseño visual de la adaptación se plantea como un reto importante de cara al futuro.

La creación de una herramienta que permita la especificación visual de reglas de adaptación daría pie a la creación de un corpus de reglas de adaptación suficientemente amplio para que los diseñadores puedan diseñar sus aplicaciones adaptativas reutilizando las reglas contenidas en un repositorio, y minimizar así el tiempo de desarrollo de la aplicación adaptativa.

En AB-UIDE existen editores que cubren la inmensa mayoría de los diagramas utilizados en el diseño. Sin embargo, no existe un entorno integrado como tal que permita la creación de interfaces de usuario desde una herramienta centralizada. Sin duda, uno de los trabajos futuro es la integración de todos los editores en una herramienta única y centralizada.

El sistema multi-agente ha sido desarrollado utilizando el lenguaje JACK. Aunque dicho lenguaje es muy potente, su licencia de uso es comercial, lo cual hace difícil compartir los resultados de la investigación en ese campo con otros investigadores. Por lo tanto, sería interesante realizar una conversión del código actualmente desarrollado en la plataforma de JACK a otro lenguaje de libre disposición, seguramente JADEX[42], ya que es de código abierto y también basado en el paradigma BDI de agentes.

Uno de los aspectos no tratados a lo largo de esta tesis es el modelo de atención. Cuando el sistema aplica una adaptación debería intentar aplicarla en el momento más ventajoso para el usuario. La aplicación de una adaptación en un momento inadecuado produce una impresión de falta de "inteligencia" por parte del sistema, y el usuario puede perder la confianza en el sistema. Actualmente las adaptaciones se aplican cuando el usuario no está realizando ninguna acción directamente sobre la interfaz. Sin embargo, sería incluso más ventajoso si el sistema fuera capaz de conocer cuándo el usuario está directamente prestando atención a la interfaz y cuando no.

[42] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/

Aunque dentro de esta tesis se contempla el modelado del usuario y se contempla su evolución no se plantean técnicas de extracción de conocimiento que permitan mejorar el modelado del usuario. En este sentido sería muy interesante incorporar ese tipo de técnicas de modelado del usuario al presente trabajo, en este sentido se han realizado algunos trabajo preliminares que ha sido recogidos en [Lop03c][Fer03].

El trabajo realizado en usiXML es tremendamente interesante, como demuestra el interés que dicho lenguaje de especificación de interfaces de usuario ha despertado en la comunidad de interacción persona-ordenador. Sin embargo, todavía queda camino por andar en su diseño, y esperamos poder seguir aportando tanto ideas para su diseño como ayuda en el diseño e implementación de las herramientas que hagan de dicho lenguaje un lenguaje de amplia utilización en entornos industriales.

Finalmente, la inclusión del grupo dentro de la red de excelencia europea SIMILAR[43], abre nuevas posibilidades a la colaboración con un grupo multidisciplinar de investigadores europeos en la mejora de las interfaces de usuario actuales desde distintos puntos de vista, intentando acercar la manera en que se interacciona con los dispositivos a la forma en que se realiza habitualmente la interacción humana.

[43] http://www.similar.cc

REFERENCIAS

[Agr97] P. Agre. *Computation and Human Experience*.: Cambridge University Press. 1997.

[All97] R. Allen, D. Garlan. A Formal Basis for Architectural Connectors, *ACM TOSEM*, 6(3), pp. 213-249, July, 1997.

[Ant01] P. Anthony, W. Hall, V. Dang, N. R. Jennings. *Autonomous Agents for Participating in Multiple On-Line Auctions*. Proc. IJCAI Workshop on E-Business and the Intelligent Web, Seattle, WA, 54-64. 2001.

[Arm95] R. Armstrong, D. Freitag, T. Joachims, T. Mitchell. *WebWatcher: A Learning Apprentice for the World Wide Web*. AAAI Spring Symposium on Information Gathering, 6-12. 1995,

[Arg03] ArgoUML. 2003. http://argouml.tigris.org

[Bai93] G. Bailey. *Iterative Methodology and Designer Training in Human-Computer Interface Design*. In Human Factors in Computing Systems. Proceedings INTERCHI'93. Amsterdam, The Netherlands, Abril, 1993.

[Bal95] M. Balabanovic, Y. Shoham. *Learning Information Retrieval Agents: Experiments with Automated Web Browsing*. AAII Spring Spring Symposium on Information Gathering, 1995.

[Bal96] H. Balzert. From OOA to GUIs: The JANUS System. Journal of Object-Oriented Programming, vol 8(9), pp. 43–47, feb, 1996.

[Bas95] J.M.C. Bastien, D.L. Scapin. Evaluating a user interface with ergonomic criteria. *International Journal of Human-Computer Interaction*, *7*, 105-121, 1995.

[Bau96] B. Bauer. Generating User Interface from Formal Specifications of the Application, in: "Computer-Aided Design of User Interfaces", Proceedings of CADUI'96 (Namur, 5-7 June 1996), J. Vanderdonckt (Ed.), Presses Universitaires de Namur, Namur, 1996

[Bau01a] B. Bauer, J. P. Müller, J. Odell. *Agent UML: A Formalism for Specifying Multiagent Interaction*. Agent-Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer, Berlin, pp. 91-103, 2001.

[Bau01b] B. Bauer. *UML Class Diagrams: Revisited in the Context of Agent-Based Systems*. Proc. of Agent-Oriented Software Engineering (AOSE) 2001, Agents 2001, Montreal, pp.1-8.

[Ben93] D. Benyon, D. Murray. Developing adaptive systems to fit individual aptitudes. Proceedings of the 1st international conference on Intelligent User Interfaces, pp. 115-121, Orlando, Florida, United States, ACM Press, 1993.

[Bod93] F. Bodart, A. M. Hennebert, J. M. Leheureux, I. Sacre, J. Vanderdonckt. Architecture Elements for Highly-Interactive Business-Oriented Applications, in Lecture Notes in Computer Science, Vol. 153, L. Bass, J. Gornostaev & C. Unger (éds.), Springer-Verlag, Berlin, 1993, pp. 83-104.

[Bod94] F. Bodart, A.M. Hennebert, J.M. Leheureux, J. Vanderdonckt. Towards a Dynamic Strategy for Computer-Aided Visual Placement, in Proc. of 2nd ACM Workshop on Advanced Visual Interfaces AVI'94 (Bari, 1-4 juin 1994), T. Catarci, M.F. Costabile, S. Levialdi & G. Santucci (éds.), ACM Press, New York, 1994, pp. 78-87.

[Bod94] F. Bodart, J. Vanderdonckt. On t*he Problem of Selecting Interaction Objects*, in Proc. of BCS Conf. HCI'94 "People and Computers IX" (Glasgow, 23-26 août 1994), G. Cockton, S.W. Draper & G.R.S. Weir (éds.), Cambridge University Press, Cambridge, 1994, pp. 163-178.

[Bod95] F. Bodart, A.M. Hennebert, J.M. Leheureux, I. Provot, B. Sacre, J. Vanderdonckt. Towards a Systematic Building of Software Architectures: the Trident methodological guide, in Proc. of 2nd Eurographics Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'95 (Toulouse, 7-9 juin 1995), Ph. Palanque & R. Bastide (éds.), Springer-Verlag, Vienne, 1995, pp. 262-278.

[Boh88] B. Bohem. *A Spiral Model of Software Development and Enhancement*. IEEE Computer 21(2). 1988.

[Bor05] A. Boronat, J.A.Carsí, I. Ramos. Automatic Reengineering in MDA Using Rewriting Logic as Transformation Engine. CSMR 2005: 228-231

[Bro87] F.P. Brooks. *No Silver Bullet: Essence and Accidents of Software Engineering*". IEEE Computer, Vol. 20, No. 4, April 1987, pp. 10-19.

[Boo99] G. Booch, J. Rumbaugh, I. Jacobson. "*El Lenguaje Unificado de Modelado*". Addison Wesley, 1999.

[Bra87] M.E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA. 1987.

[Bra97] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, J. Treur. *DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework*. Int Journal of Cooperative Information Systems, vol. 6, nº 1, 67-94, 1997.

[Bre04a] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. In Journal of Autonomous Agents and Multi-Agent Systems. May 2004. Kluwer Academic Publishers.

[Bre04b] P. Giorgini, M. Kolp, J. Mylopoulos, M. Pistore. The Tropos Methodology: an overview. In Methodologies And Software Engineering For Agent Systems. 2004. Kluwer Academic Publishing.

[Bro99] R. Brooks. *Cambrian Intelligence: The Early History of the New AI.*: Massachusetts Institute of Technology. 1999.

[Buc84] Bruce G. Buchanan, Edward H. Shortliffe (Editores). Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project (The Addison-Wesley series in artificial intelligence). Addison-Wesley, 1984.

[Bus99] P. Busetta, R. Ronnquist, A. Hodgson, A. Lucas. Jack intelligent agents - components for intelligent agents in java. AgentLink News Letter, January 1999. White paper. http://www.agent-software.com.

[Col96] A. Collinot., A. Drogoul, P. Benhamou. *Agent Oriented Design of a Soccer Robot Team*, Proceedings ICMAS-96, 41-47. 1996.

[Bru03] P. Brusilovsky. Adaptive Web-based Systems: Technologies and Examples. The Twelfth International World Wide Web Conference 20-24 May 2003, Budapest, Hungría.

[Cac03a] C. Cachero. OO-H: una extensión a los métodos OO para el modelado y generación automática de interfaces hipermediales. Tesis doctoral. J. Gómez y O. Pastor (directores), Universidad de Alicante, 2003.

[Cac03b] C. Cachero, I. Garrigós, J. Gómez. Modelado de Estrategias de Personalización en OO-H. I+D Computación Journal. 2003.

[Cal01a] G. Calvary, J. Coutaz, D. Thevenin. A Unifying Reference Framework for the Development of Plastic User Interfaces. In Proceedings of IFIP WG2.7 (13.2) Working Conference EHCI'2001 (Toronto, May 2001), M. Reed Little & L. Nigay (Eds.), Springer Verlag Publ.,LNCS 2254, pp.173-192.

[Cal03] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, J. Vanderdonckt. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15,3 (2003) 289–308

[Car83] S. Card, T. Moran, A. Newell. The Phycology of Human Computer Interaction, Lawrence Erlbaum, Hillsdale, 1983.

[Cer00] S. Ceri, P. Fraternali, A. Bongio. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. WWW9 Conference, Amsterdam, May 2000.

[Cha96] A. Chavez, P. Maes. "*Kasbah: An Agent Marketplace for Buying and Selling Goods*", Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, April 1996.

[Che76] Peter P. Chen: The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. Database Syst. 1(1): 9-36(1976)

[Cho57] N. Chomsky. *Syntactic Structures*. The Hague: Mouton. Reprint (1985). Berlin and New York, 1957.

[Chu97] L. Chung, B. Nixon, and E. Yu, Dealing with Change: An Approach Using Non-Functional Requirements' Requirement Engineering, Springer-Verlag, vol. 1, no. 4, 1997. pp. 238-260.

[Gla96] N. Glaser. *Contribution to Knowledge Modelling in a Multi-Agent Framework: The CoMoMAS Aproach*. PhD. Thesis. L'Université Henri Poincaré, Nancy I. France. 1996

[Con99] L.L. Constantine, L.A.D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* Addison-Wesley. 1999.

[Con03] L. Constantine. Canonical Abstract Prototypes for Abstract Visual and Interaction Design. Design, Specification and Verification of Interactive Systems 2003, DSV-IS 2003. In DSV-IS 2003 : Issues in Designing New-generation Interactive Systems Proceedings of the Tenth Workshop on the Design, Specification and Verification of Interactive Systems. J.A. Jorge, N.J. Nunes, J. F. Cunha (Eds). Springer Verlag, 2003. Madeira, Portugal June 4-6, 2003.

[Cou87] J. Coutaz. PAC, an Object Oriented Model for Dialog Design. In H. J. Bullinger and B. Shackel (eds), Human-Computer Interaction - INTERACT '87, Elsevier Science Publishers, 1987, pp. 431-436.

[Cra98] I.B. Crabtree, S.J. Soltysiak, M.P. Thint. "*Adaptive Personal Agents*". Personal Technologies Journal, vol. 2, nº 3, 141-151. 1998.

[Dar91] A. Dardenne, S. Fickas, A. van Lamsweerde. *Goal-Directed Concept Acquisition in Requirements Elicitation*. Proc. IWSSD-6 - 6th Intl. Workshop on Software Specification and Design, Como, 1991, 14-21.

[Das03] M. Dastani, B. van Riemsdijk, F. Dignum, J.J. Meyer, *A Programming Language for Cognitive Agents: Goal Directed 3APL*. Proceedings of the First Workshop on Programming Multiagent Systems: Languages, frameworks, techniques, and tools (ProMAS03) to be held at AAMAS'03, Melbourne, July 2003, 2003.

[Day98] T. Dayton, A. McFarland, J. Kramer. *Bridging User Needs to Object Oriented GUI Prototype via Task Object Design*. In User Interface Design, L. Wood (ed.). CRC Press, 1998.

[Der98] K.W. Derr. "*Applying OMT: A Practical Step-by-Step Guide to Using the Object Modeling Tecnique*". Cambridge University Press, Nueva York, 1998.

[Die93] H. Dieterich, U. Malinowski, T. Khme, M. Schneider-Hufschmidt. State of the Art in Adaptive User Interfaces. In: Schneider-Hufschmidt, M., Khme, T. and Malinowski, U., eds.: Adaptive User Interfaces: Principle and Practice. Amsterdam: North Holland, 1993.

[Dro98] A. Drogoul, J.D. Zucker. *Methodological Issues for Designing Multi-Agent Systems with Machine Learning Techniques: Capitalizing Experiences from RoboCup Challenge*. Technical Report of the LIP6 n°041, 1998.

[Duy02] D. Van Duyne, J. Landay, J. Hong. The Design of Sites: Patterns, Principles and Proceses for Crafting a Customer-Centered Web. Addison-Wesley Pub. Co. 2002.

[Fer03] A. Fernández-Caballero, V. López Jaquero, F. Montero, P. González. Adaptive Interaction Multi-agent Systems in E-learning/E-teaching on the Web. International Conference on Web Engineering, ICWE 2003. In *Web Engineering: International Conference, ICWE 2003, Oviedo, Spain, July 14-18, 2003. Proceedings*. J.M. Cueva Lovelle, B.M. González Rodríguez, L. Joyanes Aguilar, J.E. Labra Gayo, M. del Puerto Paule Ruiz (Eds.). Springer Verlag, LNCS 2722, pp. 144-154. ISSN:0302-9743. Oviedo, Spain, June, 2003.

[Fis93] G. Fisher. Shared knowledge in cooperative problem-solving systems – integrating adaptive and adaptable components. In *Adaptive User Interfaces*. Elsevier Science Publishers, 1993, pp. 49-68.

[Fis00] G. Fischer. User Modeling in Human-Computer Interaction. User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, 2000.

[Fol91] J.D. Foley, W.C. Kim, S. Kovacevic, K. Murray. Intelligent User Interfaces. Chapter 15.UIDE an Intelligent User Interface Design Environment, pp. 339–384. ACM Press, Addison Wesley, Reading, MA, USA., 1991. 18, 20, 375.

[Fra96] S. Franklin, A. Graesser. "*Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents*". In Agent Theories, Architectures, and Languages, 21-35. 1996.

[Fre91] J.A Freeman, D.M. Skapura. 1991. Neural Networks. Algorithms, Applications, and Programming Techniques. Addison-Wesley.

[Gab94] T. Gabric, N. Howden, E. Norling, G. Tidhar, E. Sonenberg. *Agent-oriented Design of a Traffic-Flow Control System*. Technical Report 94/24, Department of Computer Science. University of Melbourne. November 1994.

[Gom01] J. Gómez, C. Cachero, O. Pastor. On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach. IEEE Multimedia 8(2): 20-32. Special Issue on Web Engineering. 2001.

[Gra96] C. Gram, G. Cockton. (Eds.) Design Principles for Interactive Software. Chapman & Hall, 1996.

[Gra00] T.C.N. Graham, L. Watts, G. Calvary, J. Coutaz, E. Dubois, L. Nigay. A Dimension Space for the Design of Interactive Systems within their Physical Environments, DIS2000, 17-19 August 2000, ACM Publ. New York, pp. 406-416

[Gra04] T. Granollers. MPIu+a. Una metodología que integra la Ingeniería del Software, la Interacción Persona-Ordenador y la Accesibilidad en el contexto de equipos de desarrollo multidisciplinares. Universidad de Lérida, Jesús Lorés (director). España, Julio 2004.

[Hab96] A. Habel, R. Heckel, G. Taentzer. Graph grammars with negative application conditions. Fundamenta Infomaticae 26(3/4): 287-313, 1996.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8:231--274, 1987.

[Hin99] K.V. Hindriks, F.S. de Boer, W. van der Hoek and J.J.Ch. Meyer. Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems, 2(4):357-401, 1999

[Hol90] J. D. Hollan. User Models and User Interfaces: A Case for Domain Models, Task Models, and Tailorability. In Proceedings of AAAI-90, Eighth National Conference on Artificial Intelligence, AAAI Press/The MIT Press, Cambridge, MA, p. 1137, 1990.

[Hor98] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, K. Rommelse. The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, July 1998, Morgan Kaufmann: San Francisco, pp. 256-265.

[Hor99a] E. Horvitz. Principles of Mixed-Initiative User Interfaces. Proc. ACM SIGCHI Conf. Human Factors in Computing Systems, ACM press, New York, pp. 159-166, 1999.

[Hor99b] E. Horvitz. Uncertainty, Action, and Interaction: In Pursuit of Mixed-Initiative Computing. In Intelligent Systems, September/October 1999, IEEE Computer Society, pp. 17-20.

[IBM92] IBM Corporation. Object–oriented interface design: IBM common user access guidelines. Que Corp., Carmel, IN, 1992.

[Igl98] C.A. Iglesias. *Definición de una Metodología para el Desarrollo de Sistemas Multiagente*. Tesis Doctoral. Universidad Politécnica de Madrid. España. 1998.

[Igl99] C.A. Iglesias, M. Garijo, J.C. González. *A Survey of Agent-Oriented Methodologies*. In Intelligent Agents V -- Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg. 1999.

[ISO88] Information Process Systems - Open Systems Interconection - LOTOS - A Formal Description Based on Temporal Ordering of Observational Behaviour. ISO/IS 8807. 1988.

[ISO98] ISO/IEC 9126, *Software product evaluation - Quality characteristics and guidelines for their use*, 1991.

[Jac92] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. "*Object-Oriented Software Engineering. A Use Case Driven Approach*". ACM Press, 1992.

[Jac99] I. Jacobson, G. Booch, J. Rumbaugh. *El Proceso Unificado de Desarrollo de Software*. Addison-Wesley, 1999.

[Jan93] C. Janssen, A. Weisbecker, and J. Ziegler. Generating User Interfaces from Data Models and Dialogue Net Specifications, *Proc. of the ACM Conference on Human Factors in Computing Systems INTERCHI'93*, ACM Press, New York, 1993, pp. 418-423.

[Jen95] N. R. Jennings, J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek, L. Z. Varga. Using ARCHON to develop real-word DAI applications for electricity transportation management and particle accelerator control. IEEE Expert - Special Issue on Real World Applications of DAI. 1995.

[Jen00] N. R. Jennings. "*On Agent-Based Software Engineering*". Artificial Intelligence 117 (2000), pp. 277-296. Elsevier Press. April, 2000.

[Jen01] Finn V. Jensen. Bayesian networks and decision graphs. Springer Verlag. 2001.

[Joh98] W.L. Johnson, J. Rickel, R. Stiles, A. Munro. Integrating Pedagogical Agents into Virtual Environments. Presence: Teleoperators and Virtual Environments, vol. 7, nº 6. Pp 523-546. 1998.

[Kar95] C.Karagiannidis, A. Koumpis, C. Stephanidis. Supporting Adaptivity in Intelligent User Interfaces: The case of Media and Modalities Allocation. ERCIM Workshop On "User Interfaces For Heraklion", Greece, October 30-31, 1995.

[Ken95] E. A. Kendall, M. T. Malkoun, C. H. Jiang. *A Methodology for Developing Agent Based Systems*. First Australian Workshop on Distributed Artificial Intelligence, Canberra, Australia. 1995.

[Kna03] A. Knapp, N. Koch, F. Moser, G. Zhang. ArgoUWE: A Case Tool for Web Applications. First Int. Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE´03), Sept. 2003.

[Kin96] D. Kinny, M. Georgeff, A. Rao. "*A methodology and modelling technique for systems of BDI agents*". In W. Van de Velde and J. W. Perram, editors, Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a MultiAgent World, (LNAI Volume 1038), pages 56--71. Springer-Verlag: Berlin, Germany, 1996.

[Koc01] N. Koch. Software engineering for adaptive hypermedia systems: Reference model, modelling techniques and development process. Ph.D. Thesis, Ludwig-Maximilians-Universität Munchen. 2001.

[Kol02] M. Kolp, P. Giorgini, and J. Mylopoulos. Information Systems Development through Social Structures, In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ishia, Italy, July 2002.

[Kre96] C. Kreitzberg. *Managing for Usability*. In Multimedia: A Management Perspective, Alber, F. Antone (Eds), Wadsworth, 1996.

[Let98] P. Letelier, I. Ramos, P. Sánchez, O. Pastor. OASIS version 3: A Formal Approach for Object Oriented Conceptual Modeling. SPUPV-98.4011. Edited by Universidad Politécnica de Valencia, Spain, 1998.

[Lam00] A. van Lamsweerde: Requirements Engineering in the Year 00: A Research Perspective. Invited Paper for ICSE'2000 - 22nd International Conference on Software Engineering, Limerick, ACM Press, 2000.

[Lie97] H. Lieberman. "*Autonomous Interface Agents*". ACM Conference on Human-Computer Interface [CHI-97], Atlanta, March 1997.

[Lim04a] Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon, V. López-Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005, pp. 207-228.

[Lim04b] Q. Limbourg. Multi-Path Development of User Interfaces. Universidad Católica de Lovaina, J. Vanderdonckt (director), Bélgica, 2004.

[Lim05] Q. Limbourg, J. Vanderdonckt. Transformational development of user interfaces with graph transformations. Computer-Aided Design of User Interfaces IV. Jacob, R, Limbourg, Q., Vanderdonckt, J. (eds). Kluwert academic publishers, 2005.

[Lon95] F. Lonczewski. PLUG-IN: Using Tcl/Tk for Plan-Based User Guidance, Proceedings of the Tcl/Tk Workshop, July 6-8, USENIX Association, Toronto, Canada, 1995.

[Lop03a] V. López Jaquero, F. Montero, J.P. Molina, A. Fernández-Caballero, P. González. Model-Based Design of Adaptive User Interfaces through Connectors. Design, Specification and Verification of Interactive Systems 2003, DSV-IS 2003. In DSV-IS 2003: Issues in Designing New-generation Interactive Systems Proceedings of the Tenth Workshop on the Design, Specification and Verification of Interactive Systems. J.A. Jorge, N.J. Nunes, J. F. Cunha (Eds). Springer Verlag, 2003. Madeira, Portugal June 4-6, 2003.

[Lop03b] V. López Jaquero, F. Montero, A. Fernández, M. Lozano. *Towards Adaptive User Interface Generation: One Step Closer To People*. 5th International Conference on Enterprise Information Systems, ICEIS 2003. Angers, France. April 23-26, 2003.

[Lop03c] V. López Jaquero, F. Montero, A. Fernández, M. Lozano. Usability Metrics in Agent-Based Intelligent Tutoring Systems. Human-Computer Interaction: Theory and Practice (part 1). J. Jacko, C. Stephanidis (Eds.). Lawrence Erlbaum Associates. Londrés, Reino Unido, 2003. ISBN: 0-8058-4931-9. pp. 539-543.

[Lop04] V. López-Jaquero, F. Montero, J.P. Molina, P. González, A. Fernández-Caballero. A Seamless Development Process of Adaptive User Interfaces Explicitly Based on Usability Properties. Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004). Lecture Notes in Computer Science, Vol. 3425, Springer-Verlag, Berlin, 2005.

[Lop05] V. López-Jaquero, F. Montero, J.P. Molina, P. González, A. Fernández-Caballero. A Multi-Agent System Architecture for the Adaptation of User Interfaces. 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005). 15-17 September 2005, Budapest, Hungary. Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, 2005.

[Loz00] M.D. Lozano, I. Ramos, P. González. User Interface Specification and Modeling in an Object Oriented Environment for Automatic Software Development. IEEE 34th International Conference on Technology of Object-Oriented Languages and Systems, pp. 373-381, 2000.

[Löw93] M. Löwe. Algebraic Approach to Single-Pushout Graph Transformation. Theoretical Computer Science 109(1&2): 181-224 (1993)

[Luc95] M. Luck, M. d'Inverno. *Structuring a Z Specification to Provide a Formal Framework for Autonomous Agent Systems*. In ZUM'95: The Z Formal Specification Notation, J. Boewn & M. Hinhecy (eds.), Lecture Notes in Computer Science, 967, 47-62. Springer-Verlag, Heildelberg. 1995.

[Mac91] W. Mackay. Triggers and barriers to customizing software. In *Proceedings of ACM CHI 91*, pp. 153-160.

[Mae95] P. Maes. *Artificial Life meets Entertainment: Lifelike Autonomous Agents*. Communications of the ACM, Special Issue on New Horizons of Commercial and Industrial AI, Vol. 38, No. 11, November, 1995.

[Maj97] R. Majan, B. Shneiderman. Visual and Textual Consistency Checking Tools for Graphical User Interfaces.

[Mil98] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, J. White, "*MASIF The OMG Mobile Agent System Interoperability Facility*", Proceedings of the International Workshop on Mobile Agents (MA'98), Stuttgart, September 1998. It also appeared as Personal Technologies, Springer Verlag, 2:117-129. 1998.

[Min98] N. Minar. "*Designing an Ecology of Distributed Agents*". M.S. Thesis, Massachusetts Institute of Technology, 1998.

[Min88] M. Minsky. *The Society of Mind*.: Simon & Shuster. 1988.

[Mol02] J.P. Molina Massó, P. González, M.D. Lozano. A Unified Envisioning of Future Interfaces. Proceedings of the 2nd IASTED International Conference Visualization, Imaging and Image Processing, VIIP 2002, Málaga, Spain, 9-12 September, 2002, pp. 185-190. 2002.

[Mol03] J.P. Molina, P. González, M.D. Lozano, F. Montero, V. López Jaquero. Bridging the gap: developing 2D and 3D user interfaces with the IDEAS methodology. Design, Specification and Verification of Interactive Systems 2003, DSV-IS 2003. In *DSV-IS 2003: Issues in Designing New-generation Interactive Systems Proceedings of the Tenth Workshop on the Design, Specification and Verification of Interactive Systems*. J.A. Jorge, N.J. Nunes, J. F. Cunha (Eds). Springer Verlag, LNCS 2844, 2003. Madeira, Portugal June 4-6, 2003.

[Mol03a] J.P. Molina, P. González, M.D. Lozano. Developing 3D UIs using the IDEAS Tool: A case of study. Human-Computer Interaction. Theory and Practice ISSN: (ISBN 0-8058-4930-0), Ed. Lawrence Erlbaum Associates, New Jersey, USA, Vol 0, nº0, pp.1193-1193 (2003).

[Mol03b] P.J. Molina Moreno. *Especificación de Interfaz de Usuario: De los Requisitos a la Generación Automática*. Tesis doctoral, Universidad de Valencia, 2003.

[Mol05] J.P Molina, J. Vanderdonckt, F. Montero, P. Gonzalez. UsiXML: Towards Virtualization of User Interfaces. In: Proc. of 10th ACM Int. Conference on 3D Web Technology Web3D'2005 (Bangor, March 29-April 1, 2005). ACM Press, New York, 2005.

[Mon02] F. Montero, M.D. Lozano, P. González, I. Ramos. A first approach to design web sites by using patterns. In Proceedings of the First Nordic Conference On Pattern Languages of Programs. VikingPLoP. Hojstrupgard. 2002. pp. 137-158. ISBN: 87-7849-769-8.

[Mon03] F. Montero, V. López-Jaquero, J.P. Molina, M. Lozano. Improving e-Shops Environments by Using Usability Patterns. 2nd Workshop on Software and Usability Cross-Pollination: The Role of Usability Patterns. Official Workshop of IFIP working group 13.2. INTERACT 2003. September, Zürich, Switzerland. http://wwwswt.informatik.uni-rostock.de/deutch/interact/index.html.

[Mon05a] F. Montero, V. López-Jaquero, J. Vanderdonckt, P. González, M. Lozano. Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML. In Proc. of Design, Specification and Verification of Interactive Systems 2005 (DSV-IS 2005). July, 13-15, Newcastle Upon Tyne, United Kingdom, 2005.

[Mon05b] F: Montero, V. López-Jaquero, M. Lozano, P. González. A user interfaces development and abstraction mechanisms. Artículo seleccionado en el V Congreso Interacción Persona Ordenador para su publicación en Springer-Verlag, Berlin, 2005 (por aparecer).

[Mor02] G. Mori, F. Paternò, C. Santoro. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. IEEE Transactions on Software Engineering (August 2002, pp.797-813).

[Moz03] Mozilla Project. http://www.mozilla.org, 2003.

[Mye93] B. A. Myers. "*Why Are Human-Computer Interfaces Difficult to Design and Implement?*". CMU-CS-93-183. Julio, 1993.

[Mye98] B. A. Myers. "*A Brief History of Human Computer Interaction Technology*." ACM interactions. Vol. 5, no. 2, pp. 44-54. March, 1998.

[Myl92] J. Mylopoulos, L. Chung, B. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach, IEEE Trans. on Sofware. Engineering, Vol. 18 No. 6, June 1992, pp. 483-497.

[New85] A. Newell, S.K. Card. The prospects for psychological science in human-computer interaction. Human-Computer Interaction, 1, 209-242, 1985.

[Nie93] J. Nielsen. *Usability Engineering.* Academic Press. 1993.

[Nor86] D. A. Norman, S. W. Draper. (Eds.). User centered system design: New perspectives on human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986.

[Nun98] N. Nunes, J. Cunha. *Case Study: SITINA – A Software Engineering Project Using Evolutionary Prototyping*. In CaiSE'98/IFIP 9.1 EMMSAD'98 Workshop, 1998.

[Nun00] N. Nunes, J. Cunha. *Wisdom: a UML based Architecture for Interactive Systems*. In *Interactive Systems: Design, Specification, and Verification* (7th International Workshop DSV-IS, Limerick, Ireland, June, 2000), Ph. Palanque and F. Paternò (Eds.). LNCS Vol. 1946, Springer, 2000.

[Nun01] N. Nunes. *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach*. Tesis doctoral, Universidad de Madeira, Abril, 2001.

[Ode00] J. Odell, H. Van Dyke Parunak, B. Bauer. *Extending UML for Agents*. AOIS Workshop at AAAI 2000.

[Oes02] I. Oeschger, E. Murphy, B. King, P. Collins, D. Boswell. Creating Applications With Mozilla. O'Reilly, September, 2002.

[Ols90] J.R.Olson, G. Olson. The Growth of Cognitive Modeling in Human-Computer Interaction since GOMS. Human-Computer Interaction, 1990, Volume 5, pp. 221-265, Lawrence Erlbaum, 1990.

[Pad02] L. Padgham, M. Winikoff. Prometheus: A Methodology for Developing Intelligent Agents, Proceedings of the Third International Workshop on AgentOriented Software Engineering, at AAMAS 2002. July, 2002, Bologna, Italy.

[Bas90] R. Bastide, P. Palanque. Petri nets with objects for the design, validation and prototyping of userdriven interfaces. In D. Diaper et al., (editor), Human-Computer Interaction: INTERACT'90, pages 625-631, Amsterdam, North-Holland, Elsevier, 1990.

[Pas97] O. Pastor, E. Insfrán, V. Pelechano, J. Romero, J. Merseguer. OO-Method: An OO Software Production Environment Combining Conventional and Formal Methods. 9th Conference on Advanced Information Systems Engineering (CAiSE'97). Barcelona, Spain. June 1997. LNCS (1250), pages 145-159. Springer-Verlag 1997. ISBN: 3-540-63107-0.

[Pas03] O. Pastor, J. Fons, V. Pelechano. Generación Automática de Aplicaciones WEB a partir de Esquemas Conceptuales Orientados a Objetos, 84-921873-1-X, Tendencias Actuales en la Interacción Persona-Ordenador: Accesibilidad, Adaptabilidad y Nuevos Paradigmas, Isidro Ramos, Antonio Fernández y M. Dolores Lozano (editores), 8, 1 - 20, 2003

[Pat99] F. Paternò. Model-Based Design and Evaluation of Interactive Applications. Springer. 1999.

[Pin00] P. Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. In *Interactive Systems: Design, Specification, and Verification* (7th International Workshop DSV-IS, Limerick, Ireland, June, 2000), Ph. Palanque and F. Paternò (Eds.). LNCS Vol. 1946, pages 207-226, Springer, 2000.

[Pin02] P. Pinheiro da Silva. Object Modelling of Interactive Systems: The UML*i* Approach. Tesis doctoral. Universidad de Manchester, N.W. Paton (director), Reino Unido, 2002.

[Pue96] A.R.Puerta. The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. CADUI'96: Second International Workshop on Computer-Aided Design of User Interfaces, Namur, Belgium, June 1996, pp. 19-25.

[Pue97] A.R. Puerta. A Model-Based Interface Development Environment. IEEE Software, pp. 40-47, 1997.

[Pue98] A.R. Puerta. Supporting User-Centered Design of Adaptive User Interfaces Via Interface Models. First Annual Workshop On Real-Time Intelligent User Interfaces For Decision Support And Information Visualization, San Francisco, January 1998.

[Ram01] I. Ramos, A. Giret. *Full Software Agents as OASIS 3.0 Objects*. I Jornadas Dolmen, Sevilla, España. 2001.

[Ric98] C. Rich, C.L. Sidner. COLLAGEN: A Collaboration Manager for Software Interface Agents, *An International Journal: User Modeling and User-Adapted Interaction*, Vol. 8, Issue 3/4, pps 315-350, 1998

[Rob98] D. Roberts, D. Berry, S. Isensee y J. Mullaly. Designing for the User with OVID: Bridging User Interface Design and Software Engineering. New Riders Publishing, September 1998.

[Rob05] A. Robles, J.P. Molina, V. López Jaquero, A. García. Even Better Than Reality: The Development of a 3-D Online Store that Adapts to Every User and Every Platform. HCI International 2005. Las Vegas, EE.UU, Julio 22-27, 2005.

[Ros96] G. Rossi. "An Object-Oriented Method for Designing Hypermedia Applications". Tesis doctoral, Departamento de Informática, D. Schwabe (director). PUC-Río, Brasil, Julio 1996.

[Sch96] E. Schlungbaum. Model-Based User Interface Software Tools - Current State of Declarative Models. Technical Report 96-30, Graphics, Visualization and Usability Center, Georgia Institute of Technology, 1996.

[Sch94a] S. Schreiber. Specification and Generation of User Interfaces with the BOSS System.- In: Proceedings East-West International Conference on Human-Computer Interaction EWHCI'94, St. Petersburg, Russia, August 2-6, 1994. Eds.: J. Gornostaev et al. Moskau: ICSTI, 1994. Also in: Human Computer Interaction, Selected Papers EWHCI'94 Conference, Springer LNCS 876.

[Sch94b] A.T. Schreiber, B.J. Wielinga, J.M. Akkermans, W. Van de Velde. *CommonKADS: A Comprehensive Methodology for KBS Development*. Deliverable DM1.2a KADS-II/M1/RR/UvA/10/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels, 1994.

[Sch95] D. Schwabe, G. Rossi, The object-oriented hypermedia design model, Communications of the ACM 38(8), 1995, 45-46.

[Shn92] B. Shneiderman. Designing the User Interface - Strategies for Effective Human-Computer Interaction, second edition, Reading, MA: Addison-Wesley Publishing Company, 1992.

[Smi94] D. C. Smith, A. Cypher, J. Spohrer. *KidSim: Programming Agents Without a Programming Language*. Communications of the ACM, 37, 7, 55-67.1994.

[Smi86] S.L. Smith, J.N. Mosier. Design Guidelines for Designing User Interface Software. Technical Report MTR-10090 (August), The MITRE Corporation, Bedford, MA 01730, USA, 1986.

[Sou03] N. Souchon, J. Vanderdonckt. A Review of XML-Compliant User Interface Description Languages, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391

[Suk93] P. Sukaviriya, J. Foley. Supporting Adaptive Interfaces in a Knowledge-based User Interface Environment. In: W. Gray, W. Hefley, D. Murray (ed.). Proceedings of the 1993 International Workshop on Intelligent User Interfaces (Orlando, January 1993). New York: ACM Press, 1993, 107-114.

[Suk95] P. Sukaviriya, J. Muthukumarasamy, M. Frank, J. Foley. A Model-Based User Interface Architecture: Enhancing a Runtime Environment with Declarative Knowledge. In: F. Paterno (ed.): Interactive Systems: Design, Specification and Verification. Berlin: Springer, 1995, 181-197.

[Thi90] H. Thimbleby. User Interface Design. New York City: ACM Press, 1990.

[Tid99] J. Tidwell. Common Ground: A Pattern Language for Human-Computer Interaction. http://www.mit.edu/~jtidwell/. 1999.

[Tid02] J. Tidwell. UI Patterns and Techniques. http://www.mit.edu/~jtidwell/. 2002.

[You88] R.M. Young, A. MacLean. Choosing between methods: analysing the user's decisions space in terms of schemas and linear models. Proceedings of the CHI '88 Conference on Human Factors in Computing Systems, 139-143. New York, ACM, 1988.

[Van93] J. Vanderdonckt, F. Bodart. *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in ACM Proc. of the Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 avril 1993), S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel & T. White (eds.), ACM Press, New York, 1993, pp. 424-429.

[Van97] J. Vanderdonckt. Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive. PhD thesis, Facultes Universitaires Notre-Dame de la Paix, Namur, Belgium, July 1997.

[Van99a] J. Vanderdonckt. Advice-giving systems for selecting interaction objects, in Proc. of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edinbourg, 5-6 septembre 1999), N.W. Paton & T. Griffiths (eds.), IEEE Computer Society Press, Los Alamitos, 1999, pp. 152-157.

[Van99b] J. Vanderdonckt. Assisting Designers in Developing Interactive Business Oriented Applications, in Proc. of 8th Int. Conf. on Human-Computer Interaction of HCI International'99 (Munich, 22-26 août 1999), H.-J. Bullinger & J. Ziegler (eds.), Ergonomics and User Interfaces, Vol. 1, Lawrence Erlbaum Ass. Pub, Mahwah, 1999, pp. 1043-1047

[Van03] http://www.isys.ucl.ac.be/bchi/research/seguia.htm

[Von44] J. Von Neumann, O. Morgenstern. Theory of Games and Economic Behavior. Princeton University Press. 1944.

[War99] J. Warmer, A. Kleppe. The Object Constraint Language: Precise Modeling with UML. Object Technology Series. Addison-Wesley. 1999.

[Weg97] P. Wegner. Why interaction is more powerful than algorithms.: *Communications of the ACM*, 40:5 (1997) 80-91.

[Wel03] M. Welie. Patterns in interaction design. http://www.welie.com. 2003.

[W3C03] W3C. DOM. http://www.w3.org/DOM/, 2003.

[Wer00] M. Wermelinger, A. Lopes, J.L. Fiadeiro. Superposing connectors, *in Proc. 10h International Workshop on Software Specification and Design*, IEEE Computer Society Press, pp. 87-94, 2000.

[Woo94] M. Wooldridge, N.R. Jennings. Agent Theories, Architectures, and Languages: A Survey, Proc. ECAI-Workshop on Agent Theories, Architectures and Languages (eds. M.J. Wooldridge and N.R. Jennings), Amsterdam, The Netherlands, pp. 1-32, 1994.

[Woo95] M. Wooldridge, N.R. Jennings. *Intelligent Agents: Theory and Practice*. In *Knowledge Engineering Review* 10(2), 1995.

[Woo99] M. Wooldridge, N. R. Jennings, D. Kinny. *A Methodology for Agent-oriented Analysis and Design*, Proc. of the Third International Conference on Autonomous Agents (Agents'99). ACM Press, Seattle, WA, USA. 69-76, 1999.

[Woo00a] Wooldridge, M. *Reasoning About Rational Agents*. MIT Press, 2000.

[Woo00b] M. Woolridge, N.R. Jennings, D. Kinny. *The Gaia Methodology for Agent-Oriented Analysis and Design*. Autonomous Agents and Multi-Agent System, Vol 3, 285-312, 2000.

[Woo01] M. Wooldridge. P. Ciancarini. "*Agent-Oriented Software Engineering: The State of the Art*". Handbook of Software Engineering and Knowledge Engineering. World Scientific Publishing Co., 2001.

[Yu 97] E. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering' Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97) Jan. 6-8, 1997, Washington D.C., USA. pp. 226-235.

[Yu 04] http://www.cs.toronto.edu/km/GRL/

[Yue87] K. Yue, "What Does It Mean to Say that a Specification is Complete?", Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design, Monterey, 1987.

[Zav97] P. Zave. Classification of research efforts in requirements engineering. ACM Computing Surveys, 29(4):315--321, 1997.

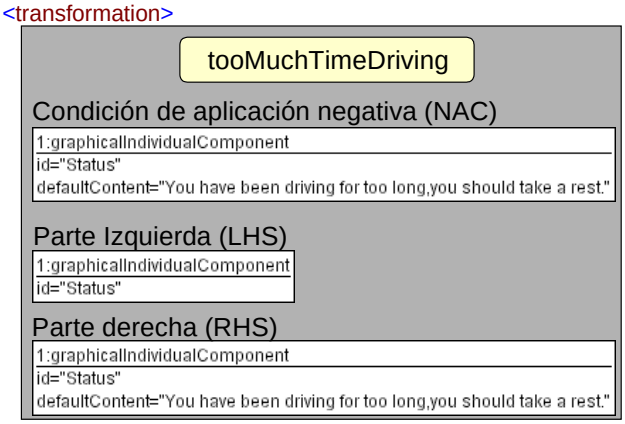
APÉNDICES

Apéndice A. Especificación de la adaptación de Car UI

```
<?xml version="1.0" encoding="UTF-8"?>
<adaptationRules xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:noNamespaceSchemaLocation="adaptivityRules-1.1.xsd">
    <sensors>
        <hardwarePlatformSensor id="HPS001"
                name="speedometerHardwarePlatformSensor"
                dataName="/contextInfo/platformInfo/hardwarePlatform/car/speed"
                dataType="int"
                description="Senses whenever speed changes."/>

        <hardwarePlatformSensor id="HPS002"
                name="RPMHardwarePlatformSensor"
                dataName="/contextInfo/platformInfo/hardwarePlatform/car/rpm"
                dataType="int"
                description="Senses whenever rpm change."/>

        <hardwarePlatformSensor id="HPS003"
                name="oilTemperatureHardwarePlatformSensor"
                dataName="/contextInfo/platformInfo/hardwarePlatform/car/oilTemperature"
                dataType="int"
                description="Senses whenever oil temperature changes."/>

        <hardwarePlatformSensor id="HPS004"
                name="headLampsHardwarePlatformSensor"
                dataName="/contextInfo/platformInfo/hardwarePlatform/car/headLamps"
                dataType="boolean"
                description="Senses whether head lamps are on or off"/>

        <hardwarePlatformSensor id="HPS005"
                name="fogLampsHardwarePlatformSensor"
                dataName="/contextInfo/platformInfo/hardwarePlatform/car/fogLamps"
                dataType="boolean"
                description="Senses whether fog lamps are on or off"/>

        <hardwarePlatformSensor id="HPS006"
                name="leftBlinkerHardwarePlatformSensor"
                dataName="/contextInfo/platformInfo/hardwarePlatform/car/leftBlinker"
                dataType="boolean"
                description="Senses whether left blinker is on or off"/>

        <hardwarePlatformSensor id="HPS006"name="leftBlinkerHardwarePlatformSensor"
                dataName="/contextInfo/platformInfo/hardwarePlatform/car/rightBlinker"
                dataType="boolean"
                description="Senses whether right blinker is on or off"/>
```

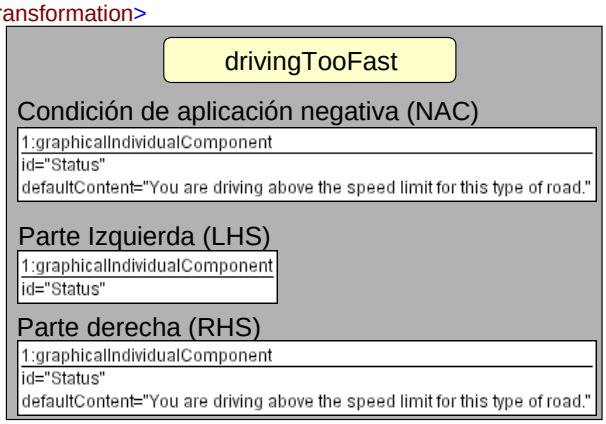
```
        <hardwareEnvironmentSensor id="HES001"
                name="lightingConditionsHardwareEnvironmentSensor"
                dataName="/contextInfo/environmentInfo/lightingConditions"
                dataType="LightingConditionsType"
                description="Senses changes in the lighting conditions"/>

        <hardwareEnvironmentSensor id="HES002"
                name="roadTypeHardwareEnvironmentSensor"
                dataName="/contextInfo/environmentInfo/roadType"
                dataType="roadTypeType"
                description="Senses the type of road the car is going through by means of a
                GPS."/>

        <hardwareEnvironmentSensor id="HES003"
                name="weatherConditionsHardwareEnvironmentSensor"
                dataName="/contextInfo/environmentInfo/weatherConditions"
                dataType="weatherConditionsType"
                description="Senses the current weather conditions."/>

        <hardwareEnvironmentSensor id="HES004"
                name="roadBendingHardwareEnvironmentSensor"
                dataName="/contextInfo/environmentInfo/roadBending"
                dataType="int"
                description="Senses the current bending of the road the car is going
                through."/>

        <softwareUserSensor id="SUS001" name="minutesDrivingSoftwareUserSensor"
                dataName="/contextInfo/userInfo/minutesDriving"
                dataType="int"
                description="Senses the time the driver is been driving (this event is
                produced by the timer one time per minute)."/>

    </sensors>
    <contextEvents>
        <contextEvent id="CE001" name="minutesDrivingContextEvent"
                description="Produced when the user has driven for too many hours.">
            <contextEventSensors>
                <sensor id="SUS001"/>
            </contextEventSensors>
        </contextEvent>

        <contextEvent id="CE002" name="speedChangesContextEvent"
                description="Produced when the speed changes.">
            <contextEventSensors>
                <sensor id="HPS001"/>
            </contextEventSensors>
        </contextEvent>

        <contextEvent id="CE003" name="minutesDrivingContextEvent"
                description="Produced when the user has driven for too many hours.">
            <contextEventSensors>
                <sensor id="SUS001"/>
            </contextEventSensors>
        </contextEvent>
```

```
    <contextEvent id="CE004" name="lightingConditionsContextEvent"
            description="Produced when lighting conditions in the road change.">
        <contextEventSensors>
            <sensor id="HES001"/>
        </contextEventSensors>
    </contextEvent>

    <contextEvent id="CE005" name="weatherConditionsContextEvent"
            description="Produced when the weather conditions change.">
        <contextEventSensors>
            <sensor id="HES003"/>
        </contextEventSensors>
    </contextEvent>
</contextEvents>

<adaptationRule id="AR001" name="tooMuchTimeDriving"
            description="If the drivers has been driving for too much time suggest taking
            a break.">
    <contextEvents>
        <contextEvent id="CE001"/>
    </contextEvents>
    <contextPrecondition id="CP001" name="moreThan3ContextPrecondition">
        <logicalSentence id="LS001"
            name="GreaterThan3" sentence="SUS001.newValue > 180"/>
    </contextPrecondition>
    <transformation>
```

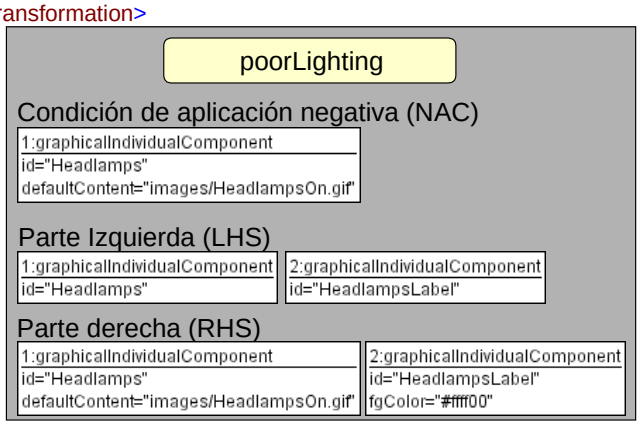



```
    </transformation>
</adaptationRule>

<adaptationRule id="AR002" name="drivingTooFast"
            description="If the driver is going faster than the speed limit for the current
            road type the system suggest slowing down.">
    <contextEvents>
        <contextEvent id="CE002"/>
    </contextEvents>
    <contextPrecondition id="CP002" name="highSpeedContextPrecondition">
        <logicalSentence id="LS002" name="highSpeed"
            sentence="HPS001.newValue >
            road.getSpeedLimit(carEnvironment.roadType)"/>
    </contextPrecondition>
```

```
    <transformation>
```

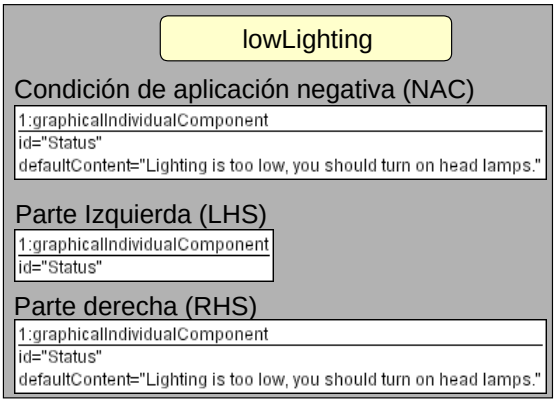



```
    </transformation>
</adaptationRule>

<adaptationRule id="AR003" name="poorLighting"
            description="When the lighting conditions are too low Car UI turns on
            automatically the head lamps.">
    <contextEvents>
        <contextEvent id="CE004"/>
    </contextEvents>
    <contextPrecondition id="CP003" name="poorLightingContextPrecondition">
        <logicalSentence id="LS003" name="poorLighting"
                sentence="HES001.newValue == 'poor' "/>
        <logicalSentence id="LS004" name="headLampsOff"
                sentence="HPS004.Value == 'false' "/>
    </contextPrecondition>
    <transformation>
```

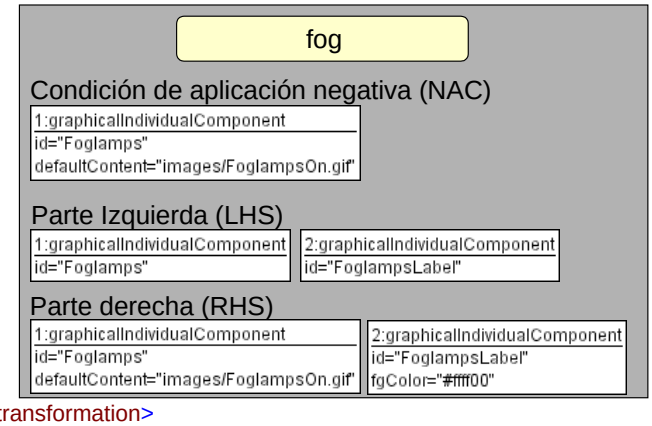



```
    </transformation>
</adaptationRule>
<adaptationRule id="AR004" name="lowLighting"
            description="When the lighting conditions are low Car UI suggest turning on
            the head lamps.">
    <contextEvents>
        <contextEvent id="CE004"/>
    </contextEvents>
```

```
    <contextPrecondition id="CP004" name="lowLightingContextPrecondition">
        <logicalSentence id="LS005" name="lowLighting"
                sentence="HES001.newValue == 'low' "/>
        <logicalSentence id="LS006" name="headLampsOff"
                sentence="HPS004.Value == 'false' "/>
    </contextPrecondition>
    <transformation>
```

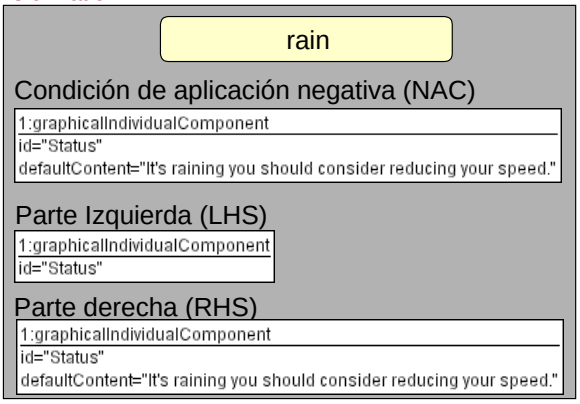



```
    </transformation>
</adaptationRule>
<adaptationRule id="AR005" name="fog"
            description="When it's foggy Car UI turns on automatically fog lamps.">
    <contextEvents>
        <contextEvent id="CE005"/>
    </contextEvents>
    <contextPrecondition id="CP005" name="fogContextPrecondition">
        <logicalSentence id="LS007" name="fog"
                sentence="HES003.newValue == 'foggy' "/>
        <logicalSentence id="LS008" name="fogLampsOff"
                sentence="HPS005.Value == 'false' "/>
    </contextPrecondition>
    <transformation>
```




```
    </transformation>
</adaptationRule>
```

```
    <adaptationRule id="AR006" name="rain"
                description="When it's rainny Car UI suggest slowing down the car below
                the speed limit.">
        <contextEvents>
            <contextEvent id="CE005"/>
        </contextEvents>
        <contextPrecondition id="CP006" name="rainContextPrecondition">
            <logicalSentence id="LS009" name="rain"
                    sentence="HES003.newValue == 'rainning' "/>
            <logicalSentence id="LS010" name="goingFast"
                    sentence="HPS001.Value >
                    road.getSpeedLimit(carEnvironment.roadType) -
                    road.getSpeedLimit(carEnvironment.roadType)*0.15"/>
        </contextPrecondition>
        <transformation>
```




```
        </transformation>
    </adaptationRule>
</adaptationRules>
```

Apéndice B. Especificación de la interfaz de usuario concreta de Car UI

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cuiModel creationDate="2005-09-03T16:05:04.644+01:00" name="CarUI" id="CarUI001"
          xsi:schemaLocation="usixml-cui-1.4.0.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://www.usixml.org/usixml">
    <version modifDate="2005-07-03T16:05:05.425+01:00" xmlns="">1</version>
    <authorName xmlns="">MoRLoP</authorName>
    <window isResizable="true" height="350" width="600" bgColor="#e0e0e0"
            id="windowWatchCarStatus" name="Car UI">
        <box id="boxRoot" name="boxRoot" type="vertical" bgColor="#e0e0e0">
            <box id="boxHorizontalGroup" name="boxHorizontalGroup" type="horizontal"
                 bgColor="#e0e0e0">
                <box id="boxWatchSpeedometer" name="boxWatchSpeedometer"
                     type="horizontal" bgColor="#e0e0e0">
                    <textComponent defaultContent="Speed" id="SpeedLabel"
                                   name="SpeedLabel" isEditable="false"/>
                    <textComponent defaultContent="0" id="Speed" name="Speed"
                                   isEditable="false"/>
                    <graphicalAdjacency id="gR001" name="gA_Speed">
                        <source sourceId="SpeedLabel"/>
                        <target targetId="Speed"/>
                    </graphicalAdjacency>
                </box>
                <box id="boxWatchRPM" name="boxWatchRPM" type="horizontal"
                     bgColor="#e0e0e0">
                    <textComponent defaultContent="RPM" id="RPMLabel"
                                   name="RPMLabel" isEditable="false"/>
                    <textComponent defaultContent="0" id="RPM" name="RPM"
                                   isEditable="false"/>
                    <graphicalAdjacency id="gR002" name="gA_RPM">
                        <source sourceId="RPMLabel"/>
                        <target targetId="RPM"/>
                    </graphicalAdjacency>
                </box>
                <box id="boxWatchOilTemperature" name="boxWatchOilTemperature"
                     type="horizontal" bgColor="#e0e0e0">
                    <textComponent defaultContent="Oil Temp" id="OiltempLabel"
                                   name="OiltempLabel" isEditable="false"/>
                    <textComponent defaultContent="0" id="Oiltemp" name="Oiltemp"
                                   isEditable="false"/>
                    <graphicalAdjacency id="gR003" name="gA_Oiltemp">
                        <source sourceId="OiltempLabel"/>
                        <target targetId="Oiltemp"/>
                    </graphicalAdjacency>
                </box>
                <box id="boxWatchLightIndicators" name="boxWatchLightIndicators"
                     type="horizontal" bgColor="#e0e0e0">
                    <textComponent defaultContent="Head Lamps" id="HeadlampsLabel"
                                   name="HeadlampsLabel" isEditable="false"/>
                    <imageComponent defaultContent="images/HeadlampsOff.gif"
```

```
                                id="Headlamps" name="Headlamps"/>
                    <graphicalAdjacency id="gR004" name="gA_Headlamps">
                        <source sourceId="HeadlampsLabel"/>
                        <target targetId="Headlamps"/>
                    </graphicalAdjacency>
                    <textComponent defaultContent="Fog Lamps" id="FoglampsLabel"
                                name="FoglampsLabel" isEditable="false"/>
                    <imageComponent defaultContent="images/FoglampsOff.gif"
                                id="Foglamps" name="Foglamps"/>
                    <graphicalAdjacency id="gR005" name="gA_Foglamps">
                        <source sourceId="FoglampsLabel"/>
                        <target targetId="Foglamps"/>
                    </graphicalAdjacency>
                    <textComponent defaultContent="Left Blinker" id="LeftblinkerLabel"
                                name="LeftblinkerLabel" isEditable="false"/>
                    <imageComponent defaultContent="images/BlinkerOff.gif"
                                id="Leftblinker" name="Leftblinker"/>
                    <graphicalAdjacency id="gR006" name="gA_Leftblinker">
                        <source sourceId="LeftblinkerLabel"/>
                        <target targetId="Leftblinker"/>
                    </graphicalAdjacency>
                    <textComponent defaultContent="Right Blinker" id="RightblinkerLabel"
                                name="RightblinkerLabel" isEditable="false"/>
                    <imageComponent defaultContent="images/BlinkerOff.gif"
                                id="Rightblinker" name="Rightblinker"/>
                    <graphicalAdjacency id="gR007" name="gA_Rightblinker">
                        <source sourceId="RightblinkerLabel"/>
                        <target targetId="Rightblinker"/>
                    </graphicalAdjacency>
                </box>
            </box>
            <box id="boxWatchSystemMessages" name="boxWatchSystemMessages"
                 type="horizontal" bgColor="#e0e0e0">
                <textComponent defaultContent="Status" id="StatusLabel"
                            name="StatusLabel" isEditable="false"/>
                <textComponent defaultContent="Bienvenido a Car UI 1.0" id="Status"
                            name="Status" isEditable="false"/>
                <graphicalAdjacency id="gR008" name="gA_Status">
                    <source sourceId="StatusLabel"/>
                    <target targetId="Status"/>
                </graphicalAdjacency>
            </box>
        </box>
    </window>
</cuiModel>
```

Apéndice C. Interfaz de usuario final en xul de Car UI

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<window xmlns=http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul
        id="windowWatchCarStatus" title="Car UI" height="150" width="540">
    <script src="initialize.js"/>
    <vbox id="boxRoot" style="background-color:#000000;">
        <hbox align="start" id="boxHorizontalGroup" style="background-color:#000000;">
            <vbox id="boxVerticalGroup" style="background-color:#000000;">
                <hbox align="start" id="boxHorizontalGroup1"
                      style="background-color:#000000;">
                    <hbox align="start" id="boxWatchSpeedometer"
                          style="background-color:#000000;">
                        <label id="SpeedLabel" value=" Speed "
                               style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                        <label id="Speed" value=" 000 "
                               style="color:#000000;background-color:#ffffff;font-weight:
                               bold;font-size:22px;"/>
                    </hbox>
                    <hbox align="start" id="boxWatchRPM"
                          style="background-color:#000000;">
                        <label id="RPMLabel" value=" RPM "
                               style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                        <label id="RPM" value=" 00000 "
                               style="color:#000000;background-color:#ffffff;font-weight:
                               bold;font-size:22px;"/>
                    </hbox>
                    <hbox align="start" id="boxWatchOilTemperature"
                          style="background-color:#000000;">
                        <label id="OiltempLabel" value=" Oil Temp "
                               style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                        <label id="Oiltemp" value=" 00 "
                               style="color:#000000;background-color:#ffffff;font-weight:
                               bold;font-size:22px;"/>
                    </hbox>
                </hbox>
                <hbox align="start" id="boxWatchSystemMessages"
                      style="background-color:#000000;">
                    <label id="StatusLabel" value=" Status "
                           style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                    <textbox maxlength="60" rows="3" multiline="true" readonly="true"
                             id="Status" value="Bienvenido a Car UI 1.0 Welcome to Car UI
                             1.0" style="color:#000000;background-color:#ffffff;font-weight:
                             bold;font-size:22px;"/>
                </hbox>
            </vbox>

            <vbox id="boxWatchLightIndicators" style="background-color:#ee0000;">
                <hbox align="start" id="boxWatchLightIndicators1a">
                    <label id="HeadlampsLabel" value=" Head Lamps "
                           style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                    <image id="Headlamps" src="images/HeadlampsOff.gif"
```

```
                    height="22" width="27"/>
            </hbox>
            <hbox align="start" id="boxWatchLightIndicators2a">
                <label id="FoglampsLabel" value="  Fog Lamps  "
                    style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                <image id="Foglamps" src="images/FoglampsOff.gif"
                    height="22" width="27"/>
            </hbox>
            <hbox align="start" id="boxWatchLightIndicators1b">
                <label id="LeftblinkerLabel" value="  Left Blinker  "
                    style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                <image id="Leftblinker" src="images/LBlinkerOff.gif"
                    height="22" width="22"/>
            </hbox>
            <hbox align="start" id="boxWatchLightIndicators2b">
                <label id="RightblinkerLabel" value=" Right Blinker "
                    style="color:#ffffff;font-weight: bold;font-size:22px;"/>
                <image id="Rightblinker" src="images/RBlinkerOff.gif"
                    height="22" width="22"/>
            </hbox>
        </vbox>
    </hbox>
</vbox>
</window>
```