Ecosistemas de Interfaz Adaptativos

TESIS DOCTORAL DE

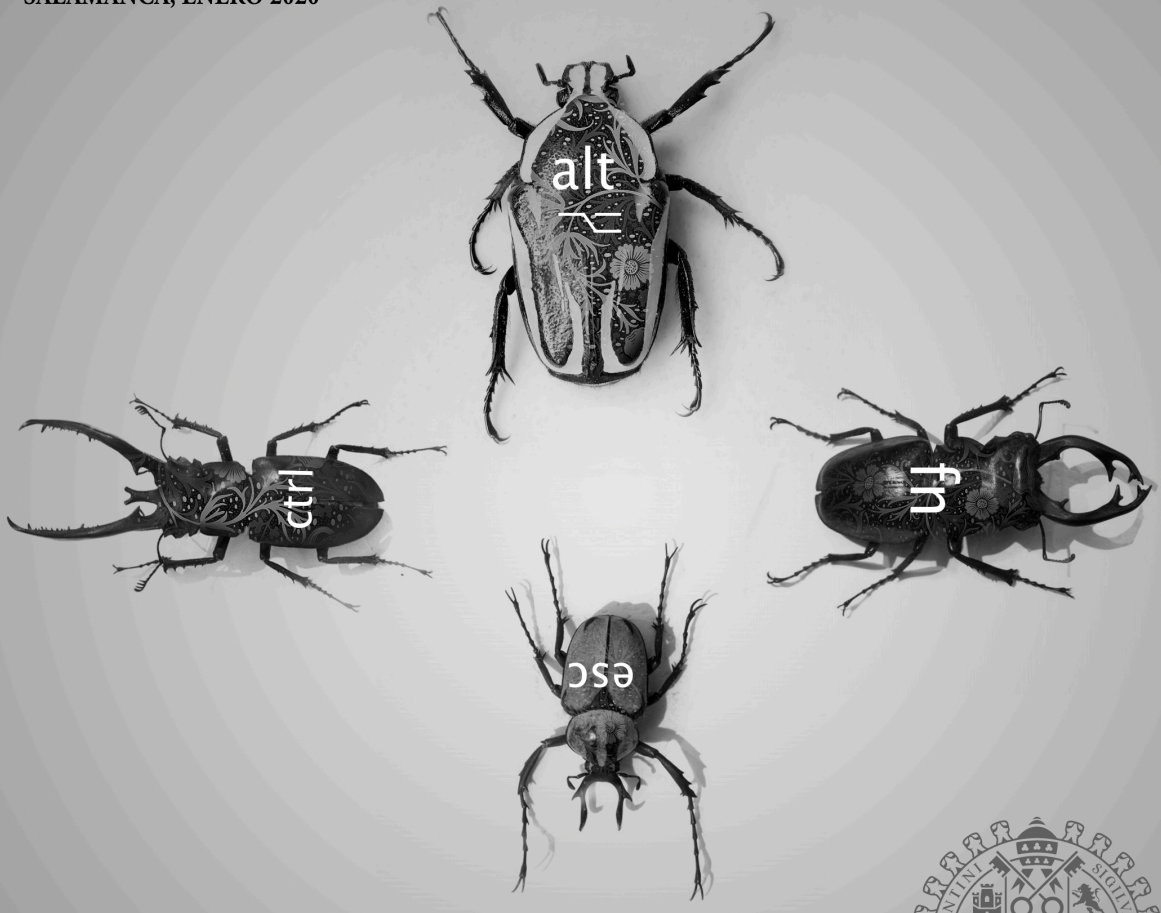
Antonio Juan Sánchez Martín

DIRECTORES

Dra. Sara Rodríguez González

Dr. Fernando de la Prieta Pintado

SALAMANCA, ENERO 2020



FACULTAD DE CIENCIAS

VNiVERSiDAD D SALAMANCA

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Departamento de Informática y Automática Facultad de Ciencias

Tesis doctoral

Ecosistemas de interfaz adaptativos

Autor

Antonio Juan Sánchez Martín

Directora

Dra. Sara Rodríguez González

Codirector

Dr. Fernando de la Prieta Pintado

Salamanca, marzo 2020

Agradecimientos

Es paradójico que durante un proceso tan largo como hacer una tesis sólo figure como autor uno de los implicados, cuando en su justa medida ha habido personas que queriendo o sin querelo han sido arrastradas a esta vorágine de complicaciones.

Los principales implicados han sido mis aguerridos tutores Sara y Fernando, compañeros de batallas en múltiples ocasiones, que no han desesperado en su función de guía y salvaguardia. Mis padres y hermanos también han colaborado prestándome su tiempo y esfuerzo al librarme de muchas obligaciones cotidianas para poder finalizar esta empresa, en la que no han cesado nunca de dar su apoyo y ánimo. Y finalmente es inevitablemente citar a Soraya, que como en todos los proyectos que emprendo, ha sido colaboradora habitual leyendo y corrigiendo todo lo escrito, siendo, además de mi faro en este mar que es la vida, la mejor de las editoras.

Muchas gracias a Pablo, buen científico y mejor amigo, que siempre ha sido una inspiración por su dedicación a la ciencia y su buen hacer. No me atrevo a citar aquí también al resto de amigos cercanos y lejanos en el espacio, familia y compañeros de trabajo, porque son muchos los que me han animado en este largo camino, sin su apoyo no hubiera podido llegar hasta el final. Finalmente, gracias a Javier Burguillo, que me inició en el mundo de la investigación, y a todas las personas que hacen software libre y que llevan la ciencia a todos los rincones de este bello planeta.

A mi hermano

"We gaze continually at the world and it grows dull in our perceptions. Yet seen from another's vantage point, as if new, it may still take the breath away."
- Alan Moore, Watchmen

Índice de contenido

Resumen

El crecimiento a nivel mundial de número de dispositivos móviles inteligentes durante las últimas décadas ha llevado no solamente en la mejora del hardware, avanzando a los niveles marcados por la Ley de Moore, sino también ha traído una mejora cualitativa del software. Esta mejora resulta como consecuencia de la automatización de la mayoría de las industrias, que con la intención de acercar los sistemas productivos a los usuarios, sustituyen a los servicios intermediarios habituales por la tecnología.

Esto ha llevado a la creación de multitud de aplicaciones móviles y web cuyo diseño y funcionalidad se ha visto condicionada frecuentemente por limitados recursos económicos. Como consecuencia, la pequeña industria se ha visto desfavorecida por el hecho de competir contra corporaciones con muchos más recursos, y no en pequeños mercados como sucedía en los siglos precedentes, sino en un único mercado global de aplicaciones.

Uno de los déficits fundamentales en estas pequeñas aplicaciones es la falta de inversión en equipos de experiencia de usuario (UX), que ayudan a acercar la tecnología al usuario, adecuándose más a sus características personales. De esto derivan una serie de problemáticas, puestas de manifiesto cuando se evalúa el software para el usuario que encontramos en el mercado:

1) Se crean aplicaciones plenamente funcionales, a menudo muy útiles, pero difícilmente usables. Si estas aplicaciones, por la naturaleza de su funcionalidad, son muy técnicas y el usuario final es un usuario experto, no suele resultar un problema determinante para su fracaso, el mayor problema lo tienen las aplicaciones generalistas de bajo presupuesto sobre las que no se puede llevar a cabo una gran refactorización.

2) Las pequeñas aplicaciones en proceso de desarrollo suelen tener también el mismo problema, abocando su destino a un éxito prematuro que les asegure estabilidad económica para poder añadir al equipo nuevos perfiles transversales que aseguren la usabilidad en su expansión.

3) En esta tesis se pretende proponer una solución que mediante mecanismos automáticos ayude a mantener la usabilidad en las pequeñas aplicaciones (1), que sea fácilmente adaptable tanto para aplicaciones en desarrollo como para aquellas que se encuentren ya en fase de mantenimiento (2) y por otra parte plantear una metodología ágil, que sea adaptable o complementaria a las actuales pero ofreciendo mecanismos de crecimiento que ayuden a mejorar la usabilidad (3). Para todo ello se han diseñado los Entornos de Interfaces Adaptativos, un paradigma de definición de interfaces que aboga por la auto re-estructuración en función del uso.

Los funcionalidad básica de los Entornos de Interfaces Adaptativos está orientada a reelaborar la interfaz mediante un procesado continuo de la información de interacción del usuario, de tal forma que la interfaz se ajuste a la forma de utilización y monitorización propia de cada usuario. También puede servir para crear versiones de aplicaciones más adaptadas a segmentos de usuario heterogéneos. Además su uso combinado a un desarrollo guiado por flags presenta una mejora en el proceso de desarrollo dando más flexibilidad a la inclusión de nuevas funcionalidades, tanto en arquitecturas horizontales como verticales, manteniendo beneficios como compartimentación del código y aportando mayor flexibilidad en despliegues y aislamiento de errores.

Introducción

Durante la breve historia de la ciencia computacional uno de los retos constantes ha sido el de mecanizar procesos, generalmente industriales. A lo largo de los años, tanto el incremento de la potencia de los ordenadores como la creación de multitud de herramientas y lenguajes, hacen que a día de hoy los objetivos pasen a ser, más que la automatización de procesos repetitivos, la generación de procesos inductivos y deductivos. Uno de los procesos que se repiten más a menudo en la industria es el rediseño de interfaces -ver (Nielsen, 1993)-, para poco a poco irlas ajustando a los hábitos o necesidades del usuario: es una constante dentro de los hábitos de consumo que mueven la industria, de forma que si un usuario consume más a gusto y más rápido, consumirá más y también generará más dinero beneficios - ver (Law, van Schaik & Roto, 2014).

Para este proceso, a veces inductivo -normalmente está basado en datos de estudios del comportamiento del usuario-, a veces deductivo -pues en muchas ocasiones se experimenta buscando nuevas soluciones en la interfaz-, se han generado multitud de herramientas y metodologías (como *Design Spring*, *Design Thinking*, *Lean Design* o *Six Sigma* -ver el apartado de ***Contexto***) que hacen el trabajo del diseñador digital mucho más cómodo.

La creación de metodologías como las herramientas se han abordado casi siempre como un punto de apoyo a la recogida de estadísticas sobre el comportamiento del usuario y se han detectado un gran número de elementos parametrizables y procesos automatizables -ver (Kashfi, Feldt, Nilsson & Svensson, 2014).

Más allá de la industria, el mundo del freeware y shareware también se ha aceptado la mejora de la interfaces de usuario y la usabilidad como elementos fundamentales en su proyectos -ver (Tabassum & Mathew, 2014)-. Siguiendo la estela de

Google, al publicar las guías de estilo de Material Design,muchos de los grandes proyectos (que se pueden permitir un equipo de diseño) han publicado también en línea sus avances en sus guías de diseño -ver (Ciurana, 2008).

Otro de los factores a tener en cuenta es que durante los últimos años la diversidad de interfaces ha ido creciendo con el despertar de técnicas como el reconocimiento visual -ver (Balaban, 2019)- o las tecnologías del habla -ver (Yi & Maghoul, 2011) y (Rao, Ture, He, Jojic & Lin, 2017)-. A día de hoy tenemos ya interfaces visuales, auditivas y táctiles, y es fácil que en el futuro podamos utilizar todo el rango de sentidos del ser humano para comunicarnos con y a través de las computadoras (¿veremos algún día los whatsapps de sabores?).

La diversidad y rápida expansión de estas interfaces ha impedido poder llegar a soluciones, en cuanto a experiencia de usuario, que puedan ofrecer interfaces más amigables y usables. Soluciones que a su vez tengan bases comunes y sean fácilmente implementables independientemente de su naturaleza física.

El problema fundamental es que para para poder ofrecer al usuario un software de calidad en todos los niveles hace falta un equipo multidisciplinar que pueda abarcar la tarea, y esto desgraciadamente solo es posible en procesos de cierta envergadura.

Por otro lado, como ya se ha mencionado antes, el mantenimiento de interfaz es un proceso casi constante y algunas aplicaciones que se encuentran en producción únicamente tienen un equipo técnico pequeño encargado del mantenimiento del software y hardware y no están capacitados, desde el punto de vista temporal, para abarcar grandes cambios en la estructura de las aplicaciones -pensemos sobretodo en aplicaciones donde los recursos son muy limitados: aplicaciones no comerciales, proyectos personales, proyectos de investigación, etc. Además encontramos que no existen metodologías ágiles que incluyan la usabilidad en el proceso de desarrollo (Magues, Castro & Acuna, 2016), lo cual hubiera facilitado mucho la implantación de diseños más adaptados.

2.1. Objetivos

Si bien el estudio de la usabilidad y la experiencia de usuario es relativamente reciente y abarca muchos campos, que van desde el puramente técnico al psicológi-

co, se han identificado una serie de procesos que pueden automatizarse resolviendo un alto porcentaje de los problemas de diseño de interfaces que no cuentan con un equipo de diseño detrás (ver el apartado **Los problemas de la usabilidad en el mundo de las aplicaciones masivas**). Esta automatización supondría un gran avance en la creación de proyectos informáticos de pequeña y mediana envergadura, ya que actualmente no existen estudios que incluyan las metodologías ágiles dentro del proceso de desarrollo. Por otro lado si que se están utilizando técnicas de análisis de datos en ámbitos como la mercadotecnia que podrían ser aplicadas en la mejora de las interfaces y la experiencia de usuario -ver (Ratchford, 2019).

2.1.1. Hipótesis

Localizando los puntos en común que tienen los distintos tipos de interfaz, y comprendiendo la forma que tiene el usuario de percibir dichas interfaces, es posible desarrollar un sistema que de forma autónoma diseñe interfaces que se adapten, no a un conjunto amplio de usuarios, sino a cada usuario de forma personalizada y automática, basándose en su comportamiento.

2.1.2. Objetivos

Por otro lado se han trazado una serie de objetivos secundarios relacionados con esta tarea central:

- Estudiar de las tendencias en el desarrollo de interfaces adaptativas, metodologías de diseño y experiencia de usuario.
- Revisar, fuera del ámbito de la computación, teorías y técnicas que puedan ayudar a la automatización de tareas de diseño de interfaz y experiencia de usuario.
- Identificar y evaluar los principales los problemas más comunes de interacción persona-ordenador.
- Modelar un sistema que permita dar solución a los problemas identificados partiendo de los datos de interacción del usuario.

- Adaptar el sistema propuesto para que sea posible integrarlo en proyectos ya en funcionamiento con un coste de tiempo y recursos pequeño, de tal forma que puedan darse soluciones a interfaces de proyectos.
- El sistema debe mantener un grado de autonomía con respecto al proyecto en que se use. De esta forma podrá asegurarse una mayor integración dentro de sistemas modulares o crear la menor interferencia posible cuando se añada a proyectos ya en funcionamiento.
- El sistema debe ser lo suficientemente genérico para cubrir el mayor tipo de interfaces posibles, con independencia de su naturaleza física, así como el mayor tipo de plataformas posibles.
- El sistema debe poder integrarse dentro de los ciclos de vida de las metodologías actuales sin necesidad de la interacción con un equipo de diseño o UX, y es posible ayudar a solucionar problemas de estas metodologías.
- Buscar soluciones que se adapten a los estándares que a día de hoy se utilizan tanto en la industria como en sistemas de producción de software no comerciales.
- El sistema debe dar soluciones a la creación de un mínimo producto viable, para su puesta en producción partiendo de una idea generalista del producto creado a partir de las experiencias colectivas de múltiples y heterogéneos usuarios.
- Evaluar los resultados en modelos de producción.

2.2. Metodología

La metodología escogida para el desarrollo de esta tesis ha sido Action-Research -ver (Lewin, 1946)- aunque sobre ella se han realizado pequeñas pero frecuentes variaciones. La base de Action-Research radica en el enfoque experimental de la ciencia con programas de acción social. En este caso el objetivo fundamental de la investigación, como se ha descrito en la sección anterior, es dar solución a problemas de origen industrial (dentro del mundo de la ingeniería informática) pero de

gran calado social, puesto que se está trabajando en el mundo de la aplicaciones masivas a través de Internet. El propio investigador se considera parte del grupo de consumidores a los que afectará el resultado de la investigación. De acuerdo con (Coughlan & Coghlan, 2002), las caracteristicas que definen la metodologia de Action-Research:

- Se trata de investigación en acción mas que de investigación acerca de la acción, involucrando a quienes experimentan estas situaciones directamente.
- El investigador participa activamente, a diferencia de lo que sucede en la investigación tradicional.
- Se buscan soluciones a la vez que se genera conocimiento cientifico.
- Constituye una secuencia de acontecimientos y un enfoque para la resolución de problemas.

De acuerdo con (Eden & Ackermann, 2018), la metodología de Action-Research se estructura en fases sobre las que se puede iterar:

- Diagnóstico y reconocimiento de la situación inicial.
- Planificación de un plan de acción.
- Actuación para poner el plan en práctica y la observación de sus efectos en el contexto que tiene lugar.
- La reflexión en torno a los efectos como base para una nueva planificación.

Cada una de las partes de la investigación (detalladas en la sección siguiente) ha seguido este proceso iterativo.

2.2.1. Estructuración de la investigación

Durante el proceso que ha supuesto la creación de este estudio se ha aplicado una metodología de trabajo dividida en cinco partes:

La **primera** parte ha consistido en observación y recolección de datos. Resultaba fundamental reconocer los procesos que pueden ser mecanizados. Para ello se

ha observado cuidadosamente el trabajo diario de un equipo de UX, diseño y SEO (*Search Engine Optimization*), labor realizada durante los últimos 4 años dentro de una empresa con el rol de programador front-end (programación del lado del cliente), con un equipo fantástico de UX, diseño y SEO en ElParking S.L. Este trabajo codo con codo ha servido para seguir la actividad diaria y procesos de ambos equipos, así como las fórmulas para la elaboración de soluciones y las herramientas utilizadas.

De igual forma se ha realizado una recolección de la bibliografía, consultando qué fórmulas, metodologías o herramientas de las existentes podrían ser beneficiosas, tanto a nivel técnico como ideológico para la llevar a cabo los objetivos.

La **segunda** parte ha consistido en la selección de los procesos que podrían ser automatizados, se ha comprobado la utilidad y función de las herramientas que utiliza el equipo de UX, diseño y SEO. Una vez seleccionados los procesos, se ha propuesto un modelo que pueda dar solución desde el punto de vista técnico al proceso inductivo que se realizaba de forma manual.

En **tercer** lugar se ha diseñado e implementado el modelo propuesto, en forma de biblioteca, apoyándose en conceptos como la pregnancia -de la que se hablará más adelante-. Para medir la eficacia del modelo se ha creado una herramienta que permite monitorizar su evolución en un sistema en producción, y se han aplicado los resultados obtenidos en un caso de estudio real.

En **cuarto** lugar se ha desarrollado una arquitectura que permite la recogida y fusión de datos de Entornos de Interfaces Adaptativos de distintos usuarios, para así poder conformar aplicaciones que, ya en su primera salida a producción, puedan adaptarse al mayor número posible de usuarios.

En quinto y **último lugar** se ha desarrollado una metodología que, con el apoyo de los Entornos de Interfaces Adaptativos, consigue solucionar algunos de los problemas más usuales en cuanto a la puesta en marcha de nuevas funcionalidades en interfaces de aplicaciones de gran número de usuarios.

2.3. Estructura de la memoria

El presente documento está dividido fundamentalmente en tres bloques temáticos más un glosario de términos del cual se aconseja su lectura antes de comenzar

la siguiente sección, puesto que muchos de los conceptos allí citados solamente serán tratados en su sección correspondiente pero serán utilizados durante todo el documento.

El **primer bloque** hace referencia al módulo de documentación y estado del arte, y se corresponde con el capítulo 3. En él se pone en situación el contexto en el que se desarrolla el estudio y se recoge la información más significativa obtenida de la revisión bibliográfica.

El **segundo bloque** que abarca los capítulos 4, 5 y 6 es donde se diseña y desarrolla el modelo, la arquitectura y metodología propuestas.

El capítulo 4 está íntegramente dedicado a los Entornos de Interfaces Adaptativos, su concepción, su modelo analítico y su funcionamiento. Además se incluye la descripción y funcionamiento de la herramienta creada para poder ver la evolución de los sistemas AIE.

El capítulo 5 se presenta una metodología denominada Feature Flag-Driven Development en la que Entornos de Interfaces Adaptativos son un elemento esencial. Finalmente en el capítulo 6 se centra en qué arquitectura seguir para la recolección de datos de distintos usuarios y así poder conseguir un mínimo producto viable previo a la primera salida en producción, basado en los resultados de la aplicación de Entornos de Interfaces Adaptativos sobre la fusión de datos de los usuarios de prueba.

El **tercer bloque** comprende los capítulos 7 y 8. En el capítulo 7 se recogen tanto los resultados de la aplicación de los Entornos de Interfaces Adaptativos sobre elementos aislados, como ejemplos de su uso en casos de estudio reales en sistemas en producción. El capítulo 8 contiene las conclusiones de la investigación, los resultados y las líneas de investigación que quedan aún abiertas.

Estado del arte

3.1. Introducción

En la oficina de Proyectos S.L., que regentan los ingenieros Juan Martínez y Luis Hernández acaba de entrar un pliego de condiciones: José González de la cooperativa agroalimentaria local, dispuesto a vender los productos ecológicos de la tierra de una forma más acorde a los tiempos modernos, está dispuesto a ofrecer a los clientes de las tiendas a los que distribuye la cooperativa que representa, su sistema de compra online con reparto a domicilio.

En el pliego se detalla correctamente quiénes son los clientes a los que desea llegar y se incluye un informe socioeconómico de la región que avala la inversión cofinanciada mediante una ayuda del gobierno central a la pequeña industria para el desarrollo de nuevas tecnologías. El pliego también incluye un listado de almacenes, la descripción técnica del sistema de envasado y almacenamiento de productos que pretende dar trazabilidad al sistema, así como información del sistema de reparto e información extra que ayudará a definir la arquitectura del sistema.

Después de cuatro meses de análisis, diseño y desarrollo se ha creado una arquitectura cliente servidor que une todo el sistema de producción con el e-commerce, y cuya cabeza visible de cara al comprador final son una web-app y una aplicación mobile realizada con Xamarin, que da cobertura a iOS y Android (en la región había un escaso 0,1 % de usuarios que utilizaban Windows Phone) desde las que hacer las compras. Para las pruebas Juan y Luis han formado en las aplicaciones a cinco personas de la comarca que funcionarán como cliente 0 *haciendo pedidos, y han distribuido manuales técnicos sobre la instalación y funcionamiento a todas las partes implicadas sobre cómo se ha realizado la integración de los distintos sistemas y cómo*

obtener la información de los pedidos a través de una aplicación web que, además, realiza parte de la contabilidad.

Tras un mes de pruebas, los clientes 0 *dieron su veredicto positivo, quedando muy contentos con la atención servida, los productos suministrados -la mayoría orgánicos-, y la puesta en producción de la aplicación.*

El resultado tras un año en producción fue nefasto, solo se recibieron un 2 % de los pedidos esperados: a los clientes les resultaba difícil la utilización de las aplicaciones, pese a valorar de forma positiva la calidad de los productos y la velocidad en la entrega. El 99 % del comercio online de la región se lo llevaron las aplicaciones de las grandes superficies, que tenían un servicio postventa similar y peor calidad de productos, pero una mejor experiencia de usuario. El proyecto fue cancelado, no disponía de los recursos suficientes para llevar a cabo una reestructuración de las apps. La experiencia de usuario resultó una barrera insalvable.

Esta es la realidad de muchas pequeñas aplicaciones que se lanzan al mercado global con recursos inferiores a los deseados. Atendiendo a los valores obtenidos a través de la plataforma statista.com -ver (Clement, 2019)- vemos como el crecimiento exponencial de las apps dentro de los *stores* de Google y Apple, por citar las dos plataformas más utilizadas, no se corresponde con una diversificación similar en cuanto al número de descargas, donde solo el 0,1 % de las aplicaciones logró acumular más de cinco millones de descargas.

Figura 3.1: Crecimiento del número de aplicaciones móviles en los últimos años

El mercado es voraz y como de costumbre favorece a aquellos que disponen de más músculo financiero. De acuerdo con los estudios de Onward Search, ver (Miller, 2014) y (Trifilio, 2014), y Nielsen Norman Group (Nielsen & Farrell, 2014), la dedicación de equipos a labores de UX en las empresas es creciente, dando soporte a labores de marketing, desarrollo de productos y desarrollo web.

3.2. Contexto

El proceso de diseño de una interfaz es a menudo un proceso largo en el que se ven involucrados, al menos, un diseñador de UX y los usuarios. Generalmente se piensa que una aplicación debe presentar una UX que se adapte al mayor número posible de usuarios, y esta interfaz suele variar a través de las versiones de la aplicación modificándose de acuerdo al uso y las necesidades de los usuarios. La mayoría de los equipos de diseño piensa de esta manera.

El proceso de diseño de la interfaz ha pasado por muchas etapas, en inicio como bien define (Hassenzahl, 2008). Un producto interactivo debía apoyarse en el logro de objetivos, pero se comprueba que sin una propuesta clara de calidad estilística, el producto no resulta interesante. Por tanto, la estrategia pasó por poner algo útil dentro una interfaz que resultara atractiva. La UX aportó elementos con-

ceptuales, como el de estimular del usuario para que pudiera deducir por sí mismo el funcionamiento de la interfaz en lugar de darle todo ya dirigido.

A medida que crecía el interés tanto técnico como comercial por la UX se desarrollaron metodologías para ello.

Desde grandes empresas como Google están favoreciendo, difundiendo e implementado metodologías como *Design Sprint* -ver (Banfield, Lombardo & Wax, 2015) y (Brown & Katz, 2011)-, iniciada en 2008 en la Universidad de Stanford por Tim Brown.

El Design Sprint es un proceso de cinco días de duración -aunque puede hacerse más corto atendiendo a la dificultad del problema-, orientado a un equipo pequeño. La estructuración del trabajo se realiza por días o sesiones: En la primera sesión se crea un mapa del problema. En la segunda, cada integrante del equipo bosqueja las soluciones. En la tercera se decide en equipo qué soluciones son las más oportunas. En la cuarta se construye un prototipo realista. En la quinta se prueba ese prototipo con cinco clientes objetivo.

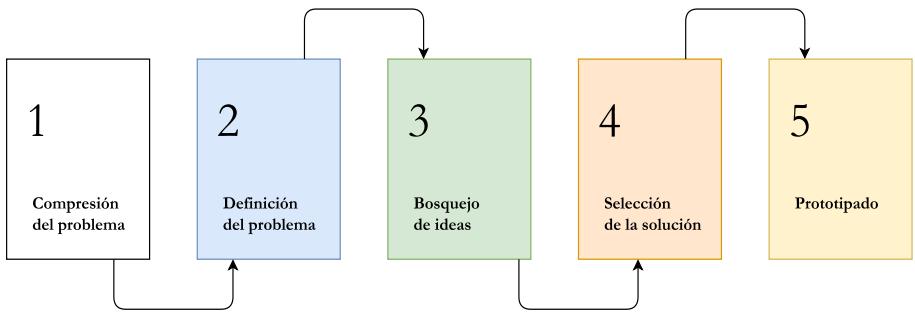


Figura 3.2: Fases del Design Sprint

Design Thinking, una metodología derivada de *Design Sprint*, basada en la holística del diseño que afirma que cualquier problema puede ser examinado a través de métodos de diseño, véase (Meinel & Leifer, 2011).

Design Thinking define tres fases superpuestas para todo proyecto: la inspiración -que incluye la identificación del problema y su descripción adecuada-, la ideación -que en paralelo con la fase de inspiración genera los conceptos de solución para estos problemas identificados- y la implementación -que concreta las partes más abstractas de las soluciones de forma práctica y test.

La diferencia fundamental entre *Design Thinking* y *Design Sprint* es que el *Design Thinking* define las herramientas y alienta a la retroalimentación entre las fases, mientras que *Design Sprint* se centra en el paso a paso.

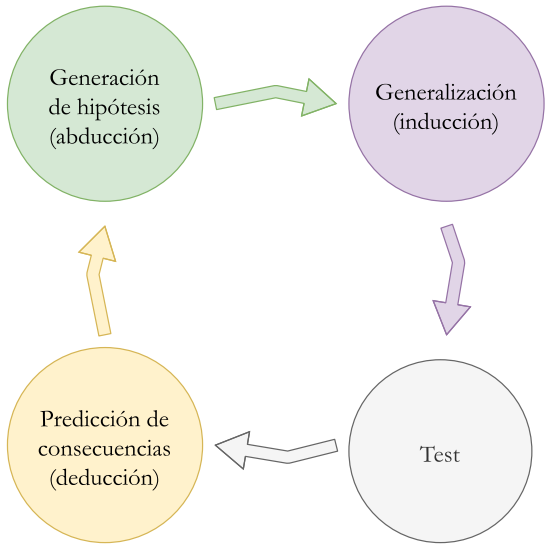


Figura 3.3: Ciclo del Design Thinking

Otras metodologías como la *Lean Design*, ver (Stull, 2018) o (May, 2012), están más centradas en el diseño de producto, y se acercan más a las metodologías clásicas de diseño de producto software: recogida de requisitos, análisis del problema, establecimiento de una tesis a demostrar, etc. (ver)

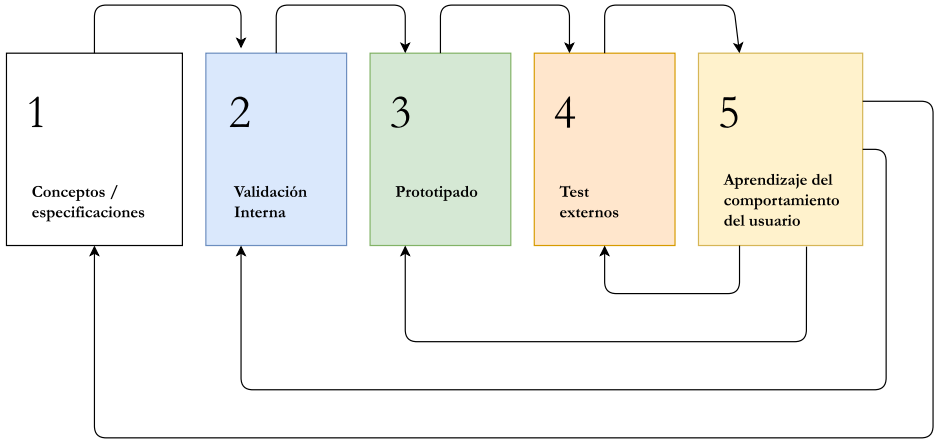


Figura 3.4: Fases del Lean Design

La fusión del *Lean Design* con otras como *Six Sigma*, ver (M. C. Lee & Chang, 2010), genera procesos más complejos generalmente adaptados a la mejora de productos que se encuentran ya en producción. El objetivo de esta fusión es determinar las necesidades de los clientes y de la empresa, e impulsar esas necesidades en la solución de mejora y evolución continua.

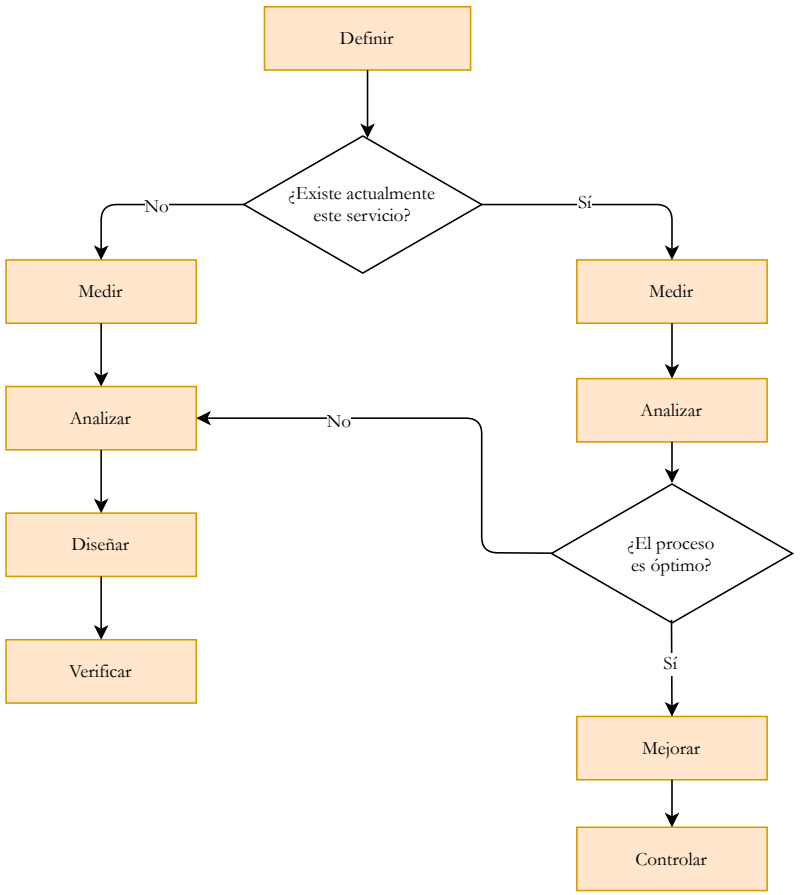


Figura 3.5: Proceso de mejora y evolución continua con Lean Design

El desarrollo software unió las metodologías ágiles al diseño de UX (Ferreira, Sharp & Robinson, 2012). Mientras que los métodos software describen qué pasos se han de hacer para transcribir las acciones a código -ver (Morrison, Holmgreen, Massey & Williams, 2013)-, los métodos de diseño de interfaces describen actividades para diseñar la interacción del producto con un usuario. Pese a la coincidencia en algunas partes del proceso existen disonancias entre orientación sobre

la integración de estas dos perspectivas, y no existen apenas estudios que ofrezcan un informe detallado de cómo combinar desarrollo ágil y el diseño de UX en la práctica. De hecho en el estudio de (Magues y col., 2016), encontramos que tras revisar la bibliografía, y pese a que la comunidad que normalmente trabaja con metodologías ágiles está muy interesada en la adopción de técnicas de usabilidad en su proceso de desarrollo, ninguna de las diversas propuestas de adopción que se han presentado hasta la fecha hace adaptaciones formales para su adopción en desarrollos ágiles.

3.3. La usabilidad en el Siglo XXI

3.3.1. El concepto de interfaz

Previo al estudio de la evolución de la usabilidad, se analizará aquello a lo que hace referencia: la interfaz. Existen muchas definiciones de interfaz, en el diccionario de la Real Academia de la Lengua Española (RAE) -ver (Española, 2005)- queda definida de la siguiente manera:

La voz inglesa interface, que significa, en informática, 'conexión física y funcional entre dos aparatos o sistemas independientes', se ha adaptado al español en la forma interfaz: «Su interfaz gráfica y capacidades de acceso a Internet facilitarán aún más el uso del PC» (Vanguardia [Esp.] 30.8.95).

La definición del concepto es muy correcta, y basándonos en ella podemos hacer hincapié en dos elementos que tendrán relevancia en este estudio. Por un lado la interfaz es definida como *conexión física*, sin hacer referencia al tipo de naturaleza ni tangibilidad de la interfaz, esto es importante porque a día de hoy nos encontramos que pueden existir interfaces para diversos medios físicos de comunicación. Por otro lado, en la definición se hace referencia a, quizás, la variedad más importante de interfaces que son utilizadas en el lenguaje común: las *interfaces gráficas de usuario*. Estas últimas son las que se analizarán a continuación, dado que ha sido su estudio el que más interés ha suscitado en las últimas décadas.

Otra buena definición del concepto de interfaz es la que aporta el *Special Interest Group on Computer- Human Interaction* (SIGCHI) de la Association of Computer Machinery (ACM) -ver (Rusu, Rusu & Roncagliolo, 2008)-:

"Interacción Persona-Ordenador (HCI) es el diseño, evolución e implementación de sistemas interactivos para el uso humano y el estudio del fenómeno que le rodea."

Esta definición resulta un poco más amplia que la anterior, al incluir el ambiente como un elemento importante en el estudio de las interfaces, que como veremos tendrá su repercusión en la computación ubicua.

3.3.2. Interfaces para todos los gustos

En la actualidad podemos encontrar distintas formas de agrupación de los tipos de interfaces. La más usual, de acuerdo con (Want, 2018), es la agrupación de acuerdo con los sentidos a los que afecta: visuales, auditivas, táctiles, olfativas, gustativas, mixtas.

Lo más usual es encontrar interfaces mixtas. El auge de la computación ubicua -ver (Poslad, 2009)- hace que hoy en día encontremos interfaces en multitud de dispositivos en cualquier ubicación y formato. Dentro de ellas, predomina habitualmente el componente visual, aunque las interfaces auditivas comienzan a tener más importancia a día de hoy con la creciente expansión de los asistentes de voz -ver (Harris, 2005)-. Algunos de los dispositivos más comúnmente utilizados, los smartphones o los laptops, son un ejemplo muy claro de interfaz mixta, si bien como hemos visto son eminentemente visuales, disponen también de interfaces auditivas, y táctiles (con táctiles no nos estamos refiriendo a que la pantalla sea táctil, puesto que la pantalla puede considerarse un periférico, sino de elementos característicos de la interfaz, como la inclusión de gestos en la forma de interaccionar con ella, o las notificaciones en forma de vibración).

Figura 3.6: Proyecto Metacookie de la Universidad de Tokio con interfaz olfativa (izquierda) y Brain-Computer-Interface de la compañía austríaca GTEC.

Al grupo de las interfaces sensitivas se puede añadir las interfaces Cerebro-Ordenador (BCI) puras -ver (Guger, Allison & Müller-Putz, 2015)-, que están basadas tanto en la recepción como en el envío de información directamente desde el cerebro, sin necesidad de reinterpretarla a través de otros órganos, por medio de dispositivos que pueden resultar más o menos invasivos. Estas interfaces se han creado originariamente para la adaptación de aplicaciones a personas con discapacidades motrices severas, aunque en la actualidad se está extendiendo su uso a más ámbitos -ver (Sourin, Earnshaw, Gavrilova & Sourina, 2016)-. La agrupación de elementos de entrada/salida en este tipo de interfaces no se aplica en este estudio puesto que los impulsos cerebrales son interpretados directamente con el sistema electrónico que se encarga de captarlos.

Existen casos también en los que el instrumental que recoge los impulsos cerebrales no es una interfaz en sí -ver (Kosch, Funk, Schmidt & Chuang, 2018)-, sino que únicamente traduce los impulsos a eventos de entrada de otro tipo de interfaces (por ejemplo, el control del puntero de un ratón a partir de lecturas de la actividad cerebral).

3.3.3. Las interfaces gráficas de usuario

Hoy en día cada persona tiene una idea clara de lo que representa para ella una interfaz de usuario (UI), y esa idea viene normalmente dada por la edad de esa persona. Si le preguntamos a una persona hace 80 años quizás el término «gráfica»

pensaría le venía grande después de trabajar con tarjetas perforadas en un IBM 1954 o, con suerte, con una pantalla en línea de comandos en un IBM s/350 -para obtener más información sobre los primeros años de la UI consultar (Myers, 1998).

Las interfaces gráficas para una persona de los años 80 en adelante tienen uno de sus primeros exponentes en los trabajos de Douglas Carl Engelbart en el *Augmentation Research Center Lab -ver (Engelbart, 1970)-* en SRI International, donde se introdujeron conceptos como el ratón, el puntero, el hipertexto o las ventanas que más tarde serían la base del primer sistema basado en ventanas desarrollado por Xerox en sus laboratorios de Palo Alto a comienzos de los años 70 -ver (Davidson, 2002).

Figura 3.7: Trabajo con tarjetas perforadas. Década de 1950

La expansión de los ordenadores personales (PC) en los años 80 hicieron que estos sistemas de ventanas se popularizaran, llegando a convertirse en un estándar, arropados por los sistemas operativos de Apple y Microsoft, líderes en la industria del PC desde entonces. Ello forzó, como sería habitual, que el diseño del software que se ejecutaba sobre estos sistemas siguiera el mismo patrón de diseño.

Los sistemas que llegaron después, como el X-Windows en entornos UNIX/-Linux o los sistemas operativos para móvil -Windows Phone, Symbian, Palm, Android, iOS o BlackBerry OS (por citar algunos de los más comunes)- no pudieron evitar la popularidad de las ventanas en su diseño.

Por lo general todos ellos han sufrido drásticos cambios a lo largo del tiempo. Durante estos últimos años en aras de mejorar la experiencia de usuario y de buscar una convergencia en los sistemas, estos han ido rediseñando su interfaz - a veces de manera drástica-, e incluyendo un grado mayor de personalización en sus configuraciones. Aun así, el diseño de cada versión siempre se ha resistido en gran modo a la adaptabilidad, abogando más por la usabilidad y la personalización. De hecho, las interfaces de usuario han sido consideradas como entes estáticos a lo largo del tiempo, y eso parte de la propia naturaleza de lo que ha sido considerada una interfaz en el mundo de las comunicaciones durante las últimas décadas, un buen ejemplo de esta consideración es la definición de Interfaz Gráfica de usuario (GUI) recogida en "The Linux Information Project" [Linux 2008]:

A graphical user interface is a human-computer interface (i.e., a way for humans to interact with computers) that uses windows, icons and menus and which can be manipulated by a mouse (and often to a limited extent by a keyboard as well).

A día de hoy la importancia del buen diseño de una UI viene incluso tipificada en normas ISO (International Organization for Standardization) -véanse la normas ISO 9241[1], donde se estandarizan los principios de la ergonomía-. En 1987 la propuesta de un UIDE *(User Interface Design Environment*) regularizó el diseño de interfaz de acuerdo con los últimos avances tecnológicos de la época, modelando los objetos de datos de las aplicaciones, acciones y condiciones previas y posteriores asociadas con las acciones de los usuarios, pero sin tener en cuenta la existencia de tipos de usuarios y sus comportamientos -ver (Foley, 1998)-, siempre pensando en la existencia de un "tipo único de usuario", aplicando conceptos como «user-friendly» o «self-explanatory». En la mayoría de casos a día de hoy estas propuestas siguen siendo válidas puesto que las aplicaciones informáticas tienen usuarios finales muy definidos.

El molde de las interfaces se empieza a romper poco a poco con la llegada de

[1]Consultar https://www.iso.org/obp/ui/ iso:std:iso:9241:-11:ed-2:v1:en para ver la norma ISO completa

las interfaces que trascienden lo visual y que se alejan de las pantallas.

En 1999, se recogieron en (Liu, Hsu, Mun & Lee, 1999, 6) una serie de factores propios de las interfaces que contribuyen a proporcionar a estas características como «user-friendly»: cobertura, confianza, robustez, importancia a nivel estadístico, simplicidad, novedad y capacidad de acción.Los primeros cinco factores son medidas objetivas, es decir se pueden manejar con técnicas que no requieren conocimientos de dominio ni de aplicación. Los dos últimos factores, sin embargo, son medidas subjetivas y miden el interés subjetivo del usuario. Se definen de la siguiente manera -ver (Frawley, Piatetsky-Shapiro & Matheus, 1992, 3):

- Novedad e innovación: las funcionalidades nuevas o desconocidos resultan interesantes para el usuario.
- Capacidad de acción o accesibilidad: los patrones son interesantes si el usuario puede hacer algo con ellos en su beneficio.

Afirma también (Liu y col., 1999, 6) que aunque las medidas objetivas son útiles en muchos aspectos, son insuficientes para determinar el interés de los patrones descubiertos. Como veremos más adelante, uno de los objetivos fundamentales de los Entornos de Interfaces Adaptativos es tratar de cuantificar el interés del usuario, clasificando al fin y al cabo patrones de comportamiento.

3.3.4. Fragmentando la interfaz

Resulta difícil encontrar un consenso sobre la descripción de la interfaz de usuario. Los trabajos que más se han centrado en ello son los desarrollos de UIDE, aunque cada uno ha definido sus propios modelos y vocabulario, -véanse por ejemplo los trabajos de (Guerrero-Garcia, Gonzalez-Calleros, Vanderdonckt & Munoz-Arteaga, 2009), (Jaquero, Montero, Molina & Gonzlez, 2008), (Meixner, Paternò & Vanderdonckt, 2011) o (Ali, Perez-Quinones, Shell & Abrams, 2001)-. Algunos de los patrones más utilizados, siguiendo los estudios de (P. P. da Silva, 2001), (Antonio Delgado, Estepa, Troyano & Estepa, 2009), (Joshi, 2015) y (Vanderdonckt y col., 2005) son los siguientes:

- El **modelo centrado en el usuario** clasifica a los usuarios en estereotipos que comparten un rol común.

- El **modelo centrado en el dominio** define los objetos accesibles para los usuarios a través de la interfaz.
- El **modelo de tareas** describe el conjunto de tareas que los usuarios pueden realizar, su descomposición jerárquica, sus relaciones y las condiciones temporales.
- El **modelo de presentación** está dedicado a los aspectos de presentación de la interfaz de usuario. A su vez, se puede descomponer en:
    - El modelo de **presentación abstracta**, encargado de descripciones abstractas de nivel de la estructura y el comportamiento de los objetos de la interfaz de usuario.
    - El modelo de **presentación concreta**, que detalla las partes de la interfaz de usuario utilizando objetos de interacción concretos (Bodart & Vanderdonckt, 1993).
- El **modelo de diálogo** define el conjunto de acciones que el usuario puede realizar dentro de varios estados del sistema y la transición entre estos estados. Vincula las tareas con elementos de interacción formando un puente entre los modelos de tareas y el de presentación, conformando así el bloque con mayor influencia en el contenido y la apariencia de la interfaz de usuario.

En el Cameleon Reference Framework (CRF) -ver (Calvary y col., 2003) y (A. Delgado, Estepa, Troyano & Estepa, 2016)-, se describen diferentes capas de abstracción relacionadas con el desarrollo basado en modelos descritos anteriormente:

- La capa de **conceptos y tareas** especifica las jerarquías de tareas que deben realizarse en los objetos de dominio para un sistema interactivo en particular. Los modelos de **dominio** y **tarea** tradicionales pertenecen a esta capa de abstracción.
- La **interfaz de usuario abstracta** describe a la interfaz de usuario en términos de unidades de interacción sin hacer ninguna referencia a la implementación. La **presentación abstracta** o los modelos de **diálogo** pertenecen a esta capa de abstracción.

- La **interfaz de usuario concreta** describe concretamente cómo los usuarios perciben la interfaz de usuario mediante el uso de objetos de interacción **concretos**. Estos objetos son dependientes de la modalidad pero independientes del lenguaje de implementación.

- La **interfaz de usuario final** expresa la interfaz de usuario en términos de código fuente que depende de la implementación. Puede representarse en cualquier lenguaje de programación o marcado de UI.

La naturaleza de la interfaz explica que algunos de los elementos característicos de entrada/salida solo puedan existir para un determinado tipo (como por ejemplo, una ventana en las interfaces visuales), pero las estructuras básicas que agrupan elementos semejantes pueden encontrarse en la mayoría de interfaces, independientemente de su naturaleza. Por ello, dentro del modelo de **presentación concreta**, podemos considerar que estos elementos son:

- **Menús** (conjunto de elementos, agrupados y clasificados cuya interacción dispara una acción en el sistema).

- **Listas** (conjunto de elementos, agrupados y clasificados cuya interacción no provoca una acción en el sistema).

- **Formulario** (conjunto de entradas de información).

- **Elementos de acción** (elementos aislados que disparan una o varias acciones del sistema).

- **Entradas de información con un casting básico de tipo** (por ejemplo la entrada de una fecha).

- **Entradas de información de tipo selector** (selección de una opción o varias opciones entre un conjunto de opciones ofertadas).

- **Pantalla o vista** (como agrupación de elementos de distinto tipo).

- **Display** (visualizador de un tipo de dato específico, ya sea simple o complejo, formado por varios tipos de datos, y que puede permitir la interacción para modificar el tipo de dato, su visualización o acceder a datos específicos de forma más detallada).

3.4. Los problemas de la usabilidad en el mundo de las aplicaciones masivas

En (Rusu y col., 2008) encontrábamos la definición de interfaz del SIGCHI (*Special Interest Group on Computer-Human Interaction*) de la ACM (*Association of Computer Machinery*) y quedaba definida como el diseño, implementación y evolución de sistemas interactivos para el uso humano y el estudio del fenómeno que le rodea. Por su parte (Kawahara, Morikawa & Aoyama, 2007), continuando en esa línea y profundizando en aquello que rodea a la interfaz de comunicación, define el concepto de interfaz en función del rol que juega ésta en la comunicación: bien como poseedora de puntos de acceso (1), bien como intérprete entre el usuario y sus metas (2) o bien como encargada de dar feedback al usuario en función de sus expectativas y sus modelos mentales (3). De cara al presente estudio es esta última acepción (3) la que más interesa, puesto que es la que hace referencia una automatización más directa, partiendo de los datos recogidos de la interacción del usuario para la modificación de la interfaz en función de sus expectativas.

Como se puede observar, algunas de estas expectativas tienen que ver directamente con la necesidad del usuario de readaptar la interfaz a sus propias inquietudes, que no necesidades (a este tipo de usuario lo podríamos denominar **usuario inquieto**), pero también tienen que ver que con el diseño inicial de la interfaz y su capacidad de adaptación a la actividad habitual de un usuario (al que podríamos denominar **usuario conformista**) que tiene patrones de actividad propios y a los cuales la interfaz puede no dar una solución directa.

De acuerdo con (Ehlert, 2003), (Zuffi, Brambilla, Beretta & Scala, 2007) y (Carroll, 1988) los problemas más graves de cara dar solución al cumplimiento de expectativas del usuario en las interfaces son:

- El uso de opciones de menú con un orden erróneo y una separación confusa, inapropiada terminología o iconografía que no se ajusta al significado esperado.

- Inapropiado e inconsistente uso de colores, sonidos, iconos, y demás contenido multimedia que no se ajusta a la cognición del usuario

- Ilegibilidad de texto debido a problemas de apariencia o contenido erróneo.
- Carencia de la propiedad de adaptación de la interfaz a los gustos del usuario y ausencia de un sistema de ayuda.
- Comportamiento no deseado.
- Desaprovechamiento de los recursos computacionales asociados a la creación de interfaces.

Estas carencias, a pesar de haber sido detectadas en interfaces visuales, son tan genéricas que pueden ser extensibles a interfaces de cualquier otro tipo.

Posiblemente la más preocupante y genérica de todas ellas es la de comportamiento no deseado, que no ha de confundirse con un *bug* o un error de código, sino una funcionalidad que no ha sido correctamente indicada al usuario. En (Avdiienko y col., 2017) podemos encontrar un ejemplo bastante claro de este tipo de problemas:

La pantalla de registro de TRIPWOLF, una popular aplicación de guía de viaje de Google Play Store, contiene una caja de texto para la introducción del correo electrónico y tres botones con las etiquetas:

- *Login con facebook*
- *Login con Google*
- *Unirse a TRIPWOLF*

La inscripción en el servicio se realiza ingresando una dirección de correo electrónico y haciendo clic en "Unirse a TRIPWOLF". El problema reside en que este botón de registro no solamente envía la dirección de correo electrónico, sino también la ubicación del usuario a los servidores TRIPWOLF, utilizando la API LocationManager.

Se entiende que el comportamiento del botón no es esperado porque, a pesar no producirse un error, no tiene el mismo comportamiento que elementos de IU similares en otras aplicaciones. Como solución para este tipo de problemas en (Avdiienko y col., 2017) y (Kuznetsov, Avdiienko, Gorla & Zeller, 2018) proponen

utilizar BACKSTAGE, un software que detecta anomalías particulares en el nivel de la interfaz de usuario en lugar del nivel de la aplicación con una precisión del 73 %.

Proyectos como JANUS -ver (Balzert, 1995)- dan una solución parcial al problema de comportamientos inesperados. JANUS es un sistema de gestión de interfaz de usuario (UIMS) que se utiliza para construir los componentes de la interfaz de usuario, pensado para solucionar los problemas existentes entre las interfaces del usuario autogeneradas a partir del dominio del problema. El sistema JANUS resuelve este problema mediante la introducción de un modelo que utiliza el análisis orientado a objetos (ODA).

El hecho de generar la UI partiendo del dominio restringe mucho la aparición de comportamientos no deseados, dejando al desarrollador como responsable final de que estos se hayan añadido, lo cual significa que deja de ser un problema de diseño.

Existen otros problemas graves a solucionar, como por ejemplo el que tienen en las interfaces de lenguaje natural (Siri, Cortana, etc.) con la ambigüedad -ver (Tong Gao, Dontcheva, Adar, Liu & Karahalios, 2015)-, o los reconocimientos de olores o sabores, pero consideramos estos problemas como problemas en la recepción de datos. En este documento se tratarán problemas de presentación, no de recepción de información.

3.5. Las leyes de la Gestalt, su mujer y sus amantes

Espero se permita la pequeña licencia del homenaje en el título de la sección a la película de Peter Greenaway de 1989 -ver (Grant, 2001). Lo cierto es que esta sección tratará acerca de un tema de índole más psicológico que computacional: La ley de la Gestalt -ver (Florio, 2014)-. Este es un concepto clave que ayudará a comprender los Entornos de Interfaces Adaptativos propuestos, y la pregnancia (su mujer) será la esencia para entender cómo se interrelacionan los elementos de una interfaz (sus amantes).

3.5.1. La Gestalt

La Gestalt es un movimiento de la psicología nacido en Alemania a comienzos del siglo XX, bajo la autoría de los investigadores Köhler, Wertheimer, Koffka y Lewin, que tenía el campo de la percepción como su principal objetivo de estudio -ver (Florio, 2014)-. La percepción, considerada como un proceso fundamental de la actividad mental, supone el apoyo para otras actividades psicológicas como el aprendizaje, la memoria o el cálculo.

En los inicios del siglo pasado la fisiología era la base de la explicación psicológica. Suponía que todo hecho psíquico se encontraba precedido y acompañado por un determinado tipo de actividad orgánica. La percepción era entendida como *el resultado de procesos corporales como la actividad sensorial* -ver (Leonardo, 2004)-. Estos procesos corporales fueron rápidamente asociados a los sentidos y entonces la psicofisiología modificó un poco su definición de percepción, quedando esta definida como *una actividad cerebral de complejidad creciente impulsada por la transformación de un órgano sensorial específico, como la visión o el tacto*. El principal avance de la Gestalt en ese momento fue modificar el concepto de percepción definiéndolo como *el proceso inicial de la actividad mental y no un derivado cerebral de estados sensoriales* a través de la cual se realiza una abstracción del mundo externo o de hechos relevantes.

"La percepción visual no opera con la fidelidad mecánica de una cámara, que lo registra todo imparcialmente: todo el conglomerado de diminutos pedacitos de forma y color que constituyen los ojos y la boca de la persona que posa para la fotografía, lo mismo que la esquina del teléfono que asoma accidentalmente por encima de su cabeza. ¿Qué es lo que vemos?... Ver significa aprehender algunos rasgos salientes de los objetos: el azul del cielo, la curva del cuello del cisne, la rectangularidad del libro, el lustre de un pedazo de metal, la rectitud del cigarrillo".

Extraído de (Arheim, 2005)

Bajo estas premisas la percepción no está sometida a la información proveniente de los órganos sensoriales, sino que es la encargada de modelar y abstraer lo recogido por ellos, calificando y cuantificando la innumerable cantidad de datos provenientes de la realidad para evitar una inútil sobrecarga de estímulos.

La Gestalt supuso una ruptura radical con la *Tabula Rasa* (Tabla rasa), la teoría de la percepción dominante desde el siglo XVII planteada por el filósofo John Locke -ver (Locke, 1690)-, según la cual la mente es una hoja en blanco sobre la cual escribe la experiencia y donde la mente es una blanda masa sistemáticamente moldeada por la influencia de las sensaciones. Y como consecuencia una ruptura también con toda la línea de pensamiento asociada a Locke y el empirismo británico de Berkeley, Hume, James y Stuart Mill.

3.5.2. La forma y las leyes de la percepción

Para establecer las leyes que rigen la percepción definidas por la Gestalt resulta ineludible aclarar el concepto de forma y fondo, como se verán en los ejemplos de la sección siguiente. Siguiendo su tradición filosófica, la Gestalt plantea que en la relación sujeto-objeto, el sujeto es aquel encargado de extraer información relevante del objeto. Esto implica que el objeto por tanto debe ser un elemento discriminable del entorno, es decir reconocible. A la barrera que separa el elemento de otro elemento o del entorno -el **fondo**- se le denomina contorno, borde o **forma**, y constituye *todo aquel conjunto de información relevante y oportuna que permite al objeto representarse* -ver (Leonardo, 2004).

De acuerdo con esta definición tenemos que las leyes de la Gestalt son las siguientes:

- Ley de la pregnancia, enunciada por Kurt Koffka, -recogida por (Katz, 1967)-: *"La organización psicológica será siempre tan excelente como las condiciones dominantes lo permitan. El término excelente abarca propiedades como la regularidad, simetría, armonía de conjunto, homogeneidad, equilibrio, máxima sencillez, concisión. . . Una buena forma (buena desde el punto de vista de la Gestalt) es la que está bien articulada. Tiende a dejar su huella en el observador, a persistir, a recurrir".*

- El principio de proximidad, enunciado por Wertheimer -citado por (Kanizza, 1987):

  "Los elementos próximos tienden a ser vistos como constituyentes de una unidad antes que los elementos alejados".

- Ley de la semejanza, recogida de (Katz, 1967):

  "Si son varios los elementos activos de diferente clase, entonces hay, en idénticas condiciones, una tendencia a reunir en grupos los elementos de igual clase".

- Tendencia al cierre, recogida por (Leonardo, 2004):

  "Toda información que contribuya a la conformación del concepto de contorno es privilegiada sobre aquella que no contribuye".

- Ley de la relación figura-fondo, recogido en (Paul, 2012):

"Todo objeto sensible existe en relación con un cierto fondo; esta expresión no solo se ajusta a las cosas visibles, sino también a toda clase de objeto sensible; un sonido se destaca sobre un fondo constituido por otros ruidos o sobre un fondo de silencio".

- Ley de dirección, recogida por (Leonardo, 2004):

  "Los detalles que mantienen un patrón o dirección tienden a agruparse juntos como parte de un modelo. Eso implica que nuestro cerebro percibe elementos continuos aunque éstos estén interrumpidos entre sí".

  Este principio posee elementos de cierre, puesto que las partículas independientes tratan de formar figuras, partiendo del principio de cierre. La ley de la dirección implica romper la dirección de lectura habitual del usuario; en el caso de no existir elementos que se agrupen será la dirección del orden de lectura la que prime en las imágenes.

  Algunos autores añaden también ley del contraste (muy similar a la ley de relación de figura y fondo, la cual afirma que un elemento puede dejar huella en el espectador rompiendo alguna de las características de forma que tengan el resto de elementos en el entorno -por ejemplo un círculo rojo que entre en conjunto de círculos negros-) y la ley de la experiencia (tendencia a acabar figuras incompletas a partir de datos de la memoria).

  En la imagen que se muestra a continuación -Fig. 3.8- la Gestalt vaticina que aunque el perfil del perro no está claramente definido, nuestro cerebro identificará al perro como una entidad completa dentro de una imagen poco clara.

Figura 3.8: Tendencia al cierre

3.5.3. La pregnancia

La pregnancia (una latinización del término *pregnanz*) resultará un concepto fundamental en este estudio, pues se tomará como medida del impacto visual (importancia que tiene un objeto sobre el espectador). La pregnancia es un concepto que puede agrupar distintas magnitudes físicas dependiendo del entorno en que sea considerada, lo cual, teniendo en cuenta la diversidad de factores que afectan a la percepción, nos da una perspectiva más fiable -ver (Lappin, Seiffert & Bell, 2020)-.

Por lo general, la pregnancia tiene una fuerte vinculación al sentido de la vista (como la mayoría de los términos utilizados por la Gestalt), haciendo referencia a al color, la textura y otras características que hacen que un observador pueda captar una forma de manera más rápida y simple (Behrens, 1998). Pero el concepto puede hacerse fácilmente extensible al resto de sentidos, puesto que la pregnancia en sí no es una propiedad física sino una característica abstracta.

Pese a que en este estudio se toma como unidad de medida, resulta muy difícil establecer un valor exacto de referencia, puesto que no existe una relación física que otorgue la referencia (como la velocidad por ejemplo que sabemos que es la

distancia entre el tiempo).

Podemos determinar que, para un contexto en concreto, un elemento puede tener más pregnancia que otro; pero solo para ese contexto, no es un valor absoluto como la velocidad y que su valor dependerá no sólo de sí mismo, sino del entorno.

Para calcular su valor primero es necesario definir qué propiedades de los objetos son importantes en ese entorno. Por ejemplo, en un entorno visual la posición relativa al centro de la composición puede ser una medida a tener en cuenta. Otra opción es recoger las primeras impresiones del espectador y recalcular atendiendo a variaciones de los mismos elementos dentro del entorno.

3.5.4. La psicología de la Gestalt y las interfaces gráficas

Los principios de la Gestalt son unos principios que, a pesar de tener casi un siglo, son todavía totalmente válidos, se encuentran vigentes para diseño visual y diseño de interfaces, y se siguen estudiando dentro de los entornos creativos.

Como hemos dicho antes las leyes de las Gestalt están muy orientadas al mundo visual, pero consideremos la siguiente adaptación para que puedan aplicarse a cualquier tipo de entorno (perceptible por los sentidos):

Tipo de interfaz	Factores
Visual	Tamaño, color, colocación, movimiento (para interfaces multimedia)
Aromática	Tipo de olor, intensidad
Gustiva	Tipo de sabor, intensidad, textura
Auditiva	Duración, velocidad, longitud de onda, periodo, amplitud
Táctil	Duración, intensidad, viscosidad, puntos de acción

Cuadro 3.1: Tipos de medidas de pregnancia en función de la interfaz

- Ley de la pregnancia. Los factores descritos en la tabla anterior ayudarán a determinar la pregnancia de forma comparativa entre los elementos del entorno. Si entre los elementos del entorno solo varía un factor, el cálculo de la diferencia de pregnancia es directo y puede ser medido por la diferencia de la intensidad de ese factor entre los dos elementos. En caso contrario, debe existir un sistema de calibrado que pondere los factores en función del entorno.

- Ley de proximidad. Los usuarios tenderán a agrupar los elementos que tengan similares factores que afecte a la pregnancia.

- Región común: en este caso podemos usar la forma o el color para dividir la interfaz en regiones, las cuales usaremos para construir las diferentes partes de una interfaz: menú de navegación, barra de herramientas, contenido, barras laterales, etc.

- Tendencia al cierre. El usuario tenderá a asociar elementos aislados si la suma de los factores de estos elementos se aproxima a la composición de un elemento mayor. Por ejemplo en una interfaz gustativa, si un usuario percibe un sabor que el 70 % dulce, 15 % ácido y 15 % amargo es posible que lo asocie con tarta de limón.

- Ley de la relación figura-fondo. En un entorno donde permanecen constantes los factores que definen la pregnancia, una variación en estos factores permite la diferenciación de una forma frente al fondo. Por ejemplo un sonido agudo entre un ruido constante de sonidos graves.

- Ley de dirección. Será únicamente aplicable a interfaces que fuercen a crear una distribución espacial del contenido, no tiene sentido para otra distinta a la visual, salvo que estemos hablando de entornos multimedia; en ese caso es posible hacer un mapa espacial de cualquier sensación.

3.6. Soluciones para problemas de usabilidad

La clave para adecuar una interfaz a un usuario es conocer cómo la percibe, lo cual implica conocer bien las leyes de la Gestalt. Una vez tenemos claras las reglas de la percepción, es el momento en que podemos comenzar a diseñar una interfaz de forma correcta y adaptada. La adaptabilidad es un concepto muy cercano al de la usabilidad, tanto que ambos han de ir siempre de la mano. Si bien la adaptabilidad es *la capacidad de una herramienta (en este contexto aplicación informática) de adecuarse a las características físicas y mentales de un usuario y su entorno,* la usabilidad es *la facilidad con que el usuario puede utilizar la herramienta* -ver (NaumannIna, Wechsung & Schleicher, 2009)-.

A día de hoy parece imposible que se pueda encontrar una solución global para los problemas de usabilidad, dada la gran diversidad de tipos de interfaces que nos encontramos como consecuencia de la ubicuidad de los sistemas. La versatilidad en la implementación de interfaces en distintos formatos nos lleva a reconsiderar los patrones de usabilidad -ver (Rusu y col., 2008)- y construcción de interfaces que a día de hoy se aplican, y considerar si se pueden establecer de una forma global.

La mayoría de metodologías más utilizadas en el desarrollo de interfaces, como (Kawahara y col., 2007), (Ehlert, 2003) o (Zuffi y col., 2007), están basadas en la resolución de problemas de GUI. Por ejemplo, Bentley -ver (Bentley, 1999)- presenta una guía de catorce pasos para interfaces en sistemas de tipo SAS (*Statistical Analysis System*), cuya implementación la lleva a cabo un equipo de diseño en el cual puede participar el usuario final. Los pasos de esta guía, que se realizan de forma iterativa, son: el análisis de las actividades y los usuarios, diseño de la interfaz, selección de los componentes, prototipado y evaluación de resultados.

Desde el punto de vista del diseño gráfico y la UX, vimos en al comienzo del capítulo cómo se han diseñado un conjunto de metodologías para tratar de solucionar un problema que podemos definir como "el diseño de la interfaz ideal". Estamos hablando del Design Sprint, Lean Spring o Design Thinking. Todas ellas, independientemente de su grado de efectividad, son procesos secuenciales, iterativos o incluso testeables, pero por desgracia no automatizables (por ahora). Requieren de una maquinaria humana que realice los proceso de inducción y deducción necesarios.

Desde el punto de vista de la ingeniería informática las soluciones que se pueden aportar pasan por dejar al usuario la tarea de diseñar su interfaz ideal. El proceso de esta forma se volvería secuencial, iterativo y testable, pero también automatizable, basado inicialmente en una propuesta de ensayo y error: es decir, propone una interfaz inicial basada en la experiencia del sistema, y se itera sobre ella hasta encontrar un resultado satisfactorio.

3.6.1. Los sistemas adaptativos

La idea de iterar sobre un sistema para que este finalice cumpliendo unos objetivos, no es nueva. Al final un sistema adaptativo no es más que un sistema que

deduce normalmente por fuerza bruta -ensayo y error- (realiza pruebas y algo le indica que va en la dirección correcta o no) si se acerca al fin que se espera de él. Es decir se transforma o aprende de una manera supervisada.

El concepto de aprendizaje supervisado se lleva estudiando muchas décadas -ver (Matthias, 2006)- y su aplicación al mundo de las interfaces (gráficas, habitualmente) también ha sido recurrente.

Dentro de los métodos de aprendizaje supervisado podemos encontrar: estadística bayesiana, razonamiento basado en casos, árboles de decisión, lógica difusa, retropropagación, métodos estadísticos de regresión, aprendizaje simbólico, máquinas de vectores de soporte, conjuntos de clasificadores, etc. En resumen, una amplia variedad de algoritmos que cumplen con la premisa de recibir una entrada y una salida deseada para reacondicionar su mecanismo interno en el futuro y realizar una predicción.

3.6.2. La lógica borrosa como sistema inductivo

Los sistemas basados en lógica difusa están muy extendidos en la tecnología cotidiana, por ejemplo en reconocimiento facial, controladores de sistema biológicos, sistemas de vehículos, etc. Se inventaron en 1965 en la Universidad de Berkeley. Se caracterizan por imitan la forma en que toman decisiones los humanos, con la ventaja de ser mucho más rápidos. Estos sistemas son generalmente robustos, así como tolerantes a imprecisiones y ruidos en los datos de entrada. En lógica difusa se usan modelos matemáticos para representar conjuntos de valores discretos que puedan ser manipulados por los ordenadores -ver (Elkan, 2001).

Las bases de una interfaz adaptativa inteligente basada en lógica borrosa recursiva fueron propuestas en 1994 en (Fukuda, Arai, Yamamoto, Naito & Matsui, 1994), que proponía cambios en la interfaz a partir del estado mental -grado de estrés- y la capacidad para realizar tareas del usuario. En este caso las ecuaciones de la lógica borrosa se utilizaban para calcular el nivel de asistencia que se le debería proporcionar al usuario en sus tareas. Como ejemplo se utilizaba un juego de hockey de realidad virtual. El análisis estadístico de los resultados experimentales reveló que la interfaz adaptativa propuesta era muy efectiva. En este caso la concesión de metas durante la adaptación de la interfaz era primordial. En la propuesta

de (Fukuda y col., 1994) se asignaban tres grados de asistencia (débil, medio o fuerte) en función del grado de experiencia del usuario (Experto, medio, principiante). En el comienzo del juego el grado de estrés era máximo y luego se iba adaptando al usuario en función de la asistencia y la experiencia que iba adquiriendo.

3.6.3. La interfaz adaptativa con sistemas multiagentes

Un sistema multiagente (SMA) es un sistema compuesto por múltiples agentes inteligentes que interactúan entre ellos. Los sistemas multiagente son utilizados para resolver problemas de carácter monolítico, dividiendo la problemática entre agentes con distinta funcionalidad -para obtener una definición más completa de agente consultar el glosario-. La cooperación entre los agentes es fundamental y conlleva aplicar protocolos de comunicación y agrupación, e interaccionar para realizar óptimamente labores de administración como reparto de tareas, negociación o asignación de trabajos especializados. La cooperación requiere a su vez que el SMA disponga de una estructura social que restringe la labor de los agentes dentro la organización.

Los sistemas multiagente constan de un middleware para soportar la comunicación (a través de estándares de comunicación entre agentes como FIPA ACL -ver (German & Sheremetov, 2008)- o KQML -ver (Labrou & Finin, 1998)-, o a través de interfaces específicas.

Los agentes, que habitualmente cooperan para proporcionar servicios inteligentes a los usuarios, pueden desempeñar dos tipos de funciones en el caso de las interfaces -ver (Shamma, 2007)-:

- Realizar tareas de control y catalogación de eventos para la construcción de sistemas de ayuda o asistencia.
- Integrarse en la capa más externa de interfaz humano-computadora. En la cual cada agente se hará cargo del control o bien de un elemento o bien de una funcionalidad, promoviendo a los agentes como mejores unidades de desarrollo y administración.

El primero de los casos es el proyecto AESOPWORLD. Según se explica en (Okada, Inui & Tokuhisa, 1999), utilizaron agentes inteligentes integrados dentro

de la interfaz, que se encargaban de recoger información y evaluar siete facultades mentales del usuario y dos físicas: reconocimiento, planificación, acción, deseo, emoción, memoria, lenguaje (mentales) y sensores, actuadores (físicas). Todo ello para confeccionar un subsistema de diálogo emocional entre el usuario y el sistema y así poder crear una línea de diálogo afectiva entre ambos.

Como ejemplo del segundo de los casos, a la postre como veremos en el siguiente capítulo más inspirador en lo que se refiere a este estudio, tenemos el proyecto AgentSpace -ver (A. R. da Silva, da Silva & Romão, 2000)-. AgentSpace es un framework de agentes móviles sobre Voyager -ver (Kiniry & Zimmerman, 1997)- y la Máquina Virtual de Java (JVM), que se desarrolló en la Universidad Técnica de Lisboa (IST) y el INESC en Lisboa, Portugal. AgentSpace provee de una serie de applets gráficos, AgentSpace-based Applets, preparados para interaccionar con los agentes de su plataforma, que tienen la capacidad de acceder a los servidores AgentSpace así como a sus recursos respectivos. Aunque el sistema de comunicación resuelve la mayoría de las necesidades de interacción entre los applets y los agentes, existen aplicaciones en las que el agente debería tomar la iniciativa (este tipo de interacción iniciada por el agente es útil cuando los agentes ejecutan tareas que requieren mucho tiempo y no se pueden completar en tiempo real). En la siguiente imagen vemos a qué niveles trabajan los agentes de (A. R. da Silva y col., 2000) -los Applets de tipo AgentSpace son los más externos.

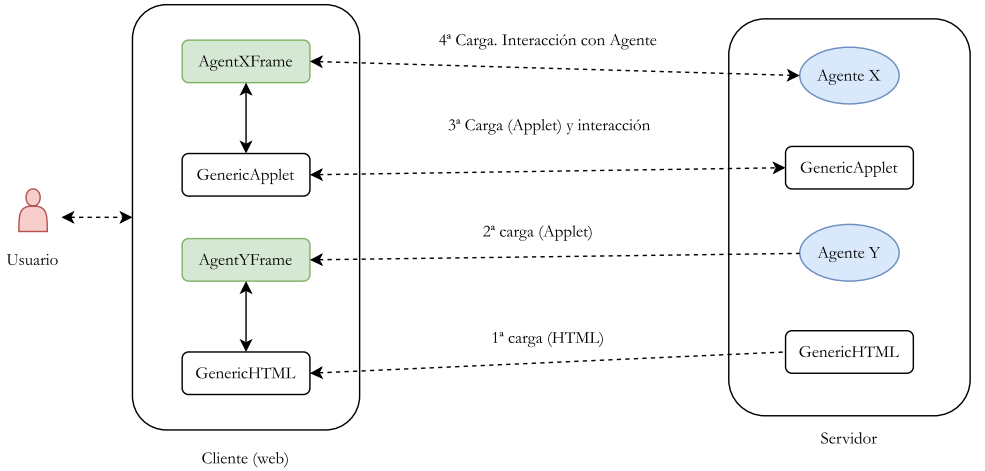


Figura 3.9: Modelo de integración de los agentes en la interfaz (AgentSpace)

Otros ejemplos de un agente de interfaz como punto de conexión entre el usuario y el diseño de la interfaz pueden encontrarse en BROA (Rodriguez, Tabares, Duque, Ovalle & Vicari, 2013) o en entornos TIC como (Ospina, Marín, Carranza & Méndez, 2017).

3.6.4. Redes de neuronas y patrones de conducta

Al igual que los SMA, las redes de neuronas artificiales (RNA) se han utilizado para desempeñar distintas funciones dentro de las interfaces. Quizás su uso más extendido es el de reconocimiento de patrones de entrada aunque también pueden ser utilizadas para reconocer y clasificar patrones de conducta.

La diversidad de topologías de redes, hace que su uso sea muy extenso en diversas áreas. En el caso de las interfaces gráficas las más utilizadas son las redes retroalimentadas (feedforward), como el Perceptrón Multicapa (MLP).

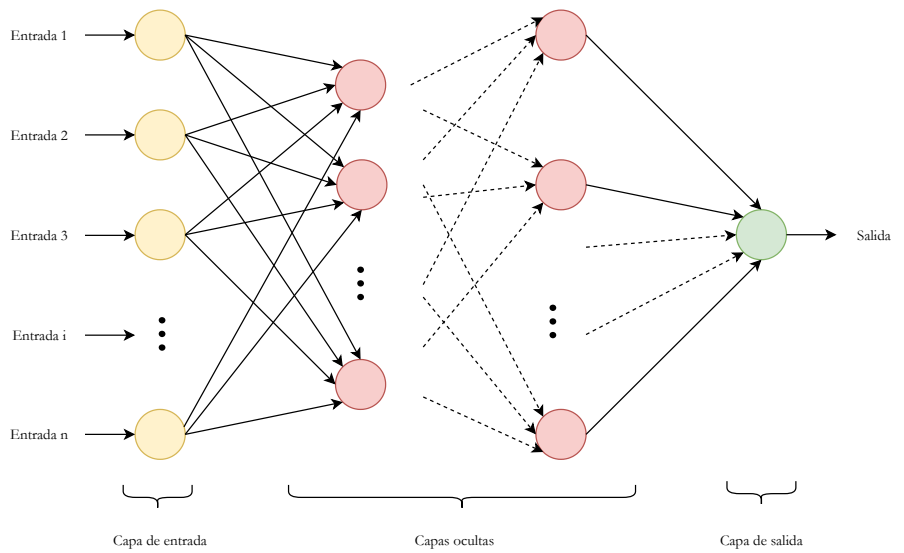


Figura 3.10: Topología de un Perceptrón Multicapa

El análisis mediante una red neuronal suele implicar una gran carga computacional (procesamiento en paralelo) para poder reducir tiempos de ejecución y acceso a los datos en su memoria local -ver (Chong, 2013)- pero posee ciertas ventajas en comparación con los métodos estadísticos tradicionales -ver (Sharma, Govindaluri & Balushi, 2015)-: puede ser lineal o no lineal, lo que permite el examen

de los procesos de decisión no compensatorios, el mapeo de entrada y salida también se puede lograr sin asumir ninguna distribución particular, la adaptabilidad de una red permite responder a cambios estructurales en el proceso de generación de datos o se puede volver a reestructurar para abordar los cambios ambientales.

Las prestaciones de las redes son muy buenas a la hora de reconocer patrones de entrada, y tiene resultados similares al de otras técnicas estadísticas como Análisis de Componentes Principales (PCA). En el estudio de (Kharat & Dudul, 2008), para reconocer la emotividad con la que el usuario se enfrenta a la resolución de tareas, se comparan redes neuronales de tipo MLP, *Generalized Feed Forward* (GFFNN) y PCA. Se utilizaron las tres técnicas para reconocer patrones en la cara de los usuarios para capturar sus emociones y como resultado se concluyó que una aproximación con Transformada de Coseno Discreta junto con un Perceptrón Multicapa (DCT-MLP) es la solución más afectiva.

La utilización de las redes para contrastar patrones de conducta tiene más que ver con la experiencia de usuario. En el estudio de (Shin & Biocca, 2017), se analizó el comportamiento del usuario en distintos servicios multipantalla, empleando redes neuronales para predecir la satisfacción general del cliente y priorizar los factores que influyen en las intenciones finales de la adquisición de un servicio que provea de un ecosistema de aplicaciones que de servicios a distintos dispositivos inteligentes. La red tomaba como entrada distintos parámetros de la experiencia de usuario (la calidad del servicio, la portabilidad, la confiabilidad, la utilidad o la disponibilidad) para determinar qué subconjunto de ellos resultaba realmente útil al usuario.

3.6.5. La gamificación y los objetivos del usuario

La gamificación -ver (Deterding, Dixon, Khaled & Nacke, 2011)-, en rasgos generales, es una estrategia que aplica técnicas de diseño de juegos a otros contextos para orientar el comportamiento de los usuarios, motivándolos a lograr sus objetivos (premiando, creando tablas de clasificación, etc.). La utilización de metodologías clásicas en el ámbito de los juegos es un excelente modo de incrementar la concentración, el esfuerzo y la motivación fundamentada en el reconocimiento.

En (Leichtenstern & Andre, 2008) proponen una variante a (Kharat & Dudul,

2008) y (A. R. da Silva y col., 2000), partiendo del análisis de las características propias del usuario para categorizarlo y agruparlo. La categorización está basada en dos aspectos: los modelos mentales (que recogen la información necesaria para identificar y construir la estructura mental del usuario junto con su actitud positiva para utilizar el sistema) y la consecución de objetivos (es decir, las especificaciones personales de objetivos primarios y secundarios que los usuarios pueden tener al usar el sistema -ver (Dillon & Watson, 1996).

El modelado a partir de objetivos de (Leichtenstern & Andre, 2008) nos lleva directamente a los estudios de Seaborn y Fels -ver (Seaborn & Fels, 2015)- que mencionan que se puede recurrir al término gamificación, comúnmente utilizado en juegos y que tiene relación directa con la consecución de objetivos, en contextos que no son juegos (es necesario aclarar que existe una diferencia entre ambientes de gamificación y los ITS retroalimentados *-Intelligent Tutorial Systems-*, ya que los ITS utilizan mensajes motivacionales, y confirman las acciones del usuario).

En la gamificación el componente adaptativo es esencial, los resultados en un ambiente gamificado dependen única y exclusivamente del usuario. Una interfaz gamificada es por tanto una interfaz en mayor o menor medida adaptada a las circunstancias de cada usuario.

La desventaja de la gamificación es que puede sufrir un rechazo temprano por parte del usuario:

> *"Un sistema gamificado puede fracasar por la reacción inicial que pueda provocar. Por esta razón, consideramos que esta pregunta -¿Cuáles son los objetivos que se quieren conseguir?- es esencial a la hora de iniciar el proceso".*
>
> (Teixes, 2014)

El proceso de gamificación de una interfaz puede resumirse en dos pasos, siguiendo los pasos de (González, 2014) :

- **Análisis del problema.** El motivo suele estar ubicado en una dimensión bipolar entre la dificultad al acceso de información (desconocimiento de los elementos de la interfaz) y los problemas normativos (desconocimiento de las "reglas de uso").

- **Método de resolución.** Los dos tipos de problemas tienen formas de solución bien distintas. Los problemas de dificultad al acceso de información se suele solventar utilizando la gamificación como interface para la estructura conceptual subyacente: ayudando al usuario a localizar las funcionalidades que le sirven. Respecto a los problemas normativos, la gamificación se utiliza como una herramienta para ayudar, también al usuario, a realizar más rápida o grata la funcionalidad que busca. Dicho con otras palabras tutorizar y asistir al usuario.

Para poder aplicar estos pasos urge saber por tanto cuáles son las expectativas del usuario, qué es lo que demanda de la interfaz, y eso si no se tiene claro desde el principio ya que el comportamiento del usuario es muy voluble, la interfaz debe inferir puntuando positivamente aquellas funcionalidades que el usuario más solicita.

3.7. Describiendo interfaces

A día de hoy la autogeneración de interfaces es un reto que está llegando a completarse gracias al desarrollo de dos áreas. Una es la del procesamiento del lenguaje natural y otra la de los lenguajes de definición de interfaces. En cuanto a la primera cabe señalar que si bien no es indispensable, su trabajo como tecnología "traductora" ayuda en muchas áreas a integrar tecnologías. De la segunda, más interesante en este estudio, es el uso de estándares lo que más ha favorecido su implantación.

A finales de siglo XX la creación de interfaces iba ligada al código, las primeras librerías que permitían acceso a los componentes gráficos del sistema lo hacían desde llamadas a funciones (primero) o métodos de objetos (después).

La irrupción de los "editores de ventanas", entre los que quizás el más exitoso fue el Visual Studio (Microsoft Corporation, 1991) marcaría el desarrollo en las siguientes décadas, utilizando un ambiente de desarrollo completamente gráfico que facilitara la creación de interfaces gráficas. El resto de librerías gráficas incluirían desde entonces hasta el día de hoy editores de ventanas similares.

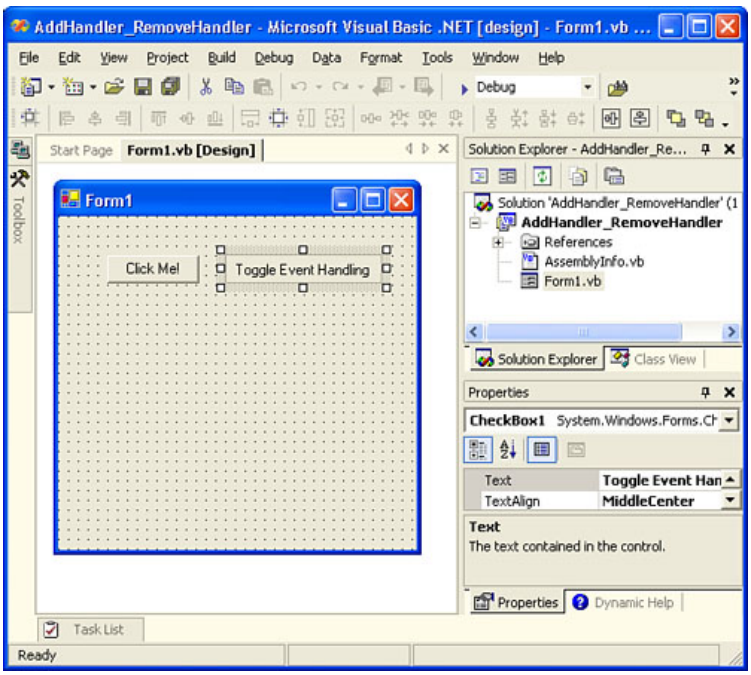


Figura 3.11: Editor de ventanas de Visual Basic .NET 2003

El uso de los editores y el impulso de arquitectura MVC (*Modelo, Vista, Controlador*) hizo que poco a poco el diseño de la interfaz se fuera alejando de la funcionalidad y permitió la inclusión de lenguajes con el que describirlas.

A partir del 1998 el XML pasa a ser aceptado por la W3C y librerías como Glade (The GNOME Foundation, 1998), que define GladeXML, o GTK (GNU Project, 1998) lo toman como base para sus editores de interfaces. El XML resultó ser el modelo perfecto dado su capacidad de extensión. Así cada librería podía definir sus propias especificaciones pero sin salirse del estándar. Ahora en todos los entornos de desarrollo más utilizados (xCode, Eclipse, JetBrains, Visual Studio, etc.) existe un editor de interfaces que guarda la información en XML.

Una vez definido el estándar, y pudiendo el diseñador obrar con más libertad, comenzaron a establecerse las metodologías definidas a continuación, que permiten producir interfaces más rápida y eficazmente.

La metodología de Bentley -ver (Bentley, 1999)- es uno entre los muchos ejemplos de metodologías iterativas que derivan de los ciclos de desarrollo propuestos por el RUP (*Rational Unified Process*) -ver (Jacobson, Booch & Rumbaugh, 1999)- o el DSDM (*Dynamic Systems Development Method*) -ver (Stapleton, 2003)- don-

de la usabilidad gira entorno al diseño de interfaces, arquetipado de usuario -ver (Junior & Filgueiras, 2005)-, y crítica para identificar las necesidades y habilidades del usuario.

Podemos encontrar también metodologías como TRIDENT (*Tools foR an Interactive Development ENvironmenT*) -ver (Bodart y col., 1995)- para la generación de interfaces de usuario partiendo de una definición de requisitos basada en un modelo E/R enriquecido y Grafos de Encadenamiento de actividades (ACG) que dan solución desde cero a los problemas de generación de interfaces de forma semiautomática.

Otras metodologías como OVID (*Object, View and Interaction Design*) -ver (Isensee, Roberts, Berry & Mullaly, 1998)- partieron de los principios de la división de la interfaz, de los elementos Objeto-Vista-Interacción (no confundir con MVC) para diseñar la interfaz tomando como entrada los requisitos y el análisis de las tareas del usuario. OVID genera una salida estructurada que puede ser integrada fácilmente en las metodologías para el diseño.

TERESA (*Transformation Environment for inteRactivE Systems representAtions*) es también una solución semiautomática diseñada para la generación de interfaces de usuario para distintas plataformas a partir de modelos de tareas creados utilizando la notación de ConcurTaskTrees (CTT) definida en (Paternò, 2000) y (Mori, Paternò & Santoro, 2002). CTT es una notación basada en el lenguaje LOTOS para el análisis y generación de interfaces de usuario a partir de modelos de tareas.

La generación de una interfaz partiendo de un modelado de UML o UML con notación expandida como es el caso de UWE -ver (Koch, 2001)- y WISDOM (*Whitewater Interactive System Development with Object Models)*, descrito en (Jardim & Falcão, 2001) y (Nunes, 2001), o herramientas de modelado procedimental (PMT) -como el proceso descrito en (Kirisci & Thoben, 2018)- sigue también generando una interfaz de forma automática desde cero, pero en ningún caso se tiene en cuenta la adaptabilidad al usuario una vez resuelto el diseño.

En entornos de TIC (*Tecnologías de la Información y la Comunicación*), podemos encontrar multitud de trabajos en los que se han generado interfaces adaptativas para ajustar los repositorios de objetos de aprendizaje a las capacidades del usuario. Véanse por ejemplo los trabajos de (Smyth, McGinty, Reilly & McCarthy,

2004), (Steichen, Ashman & Wade, 2012), (Uruchrutu, MacKinnon & Rist, 2005), o (Nivethika, Vithiya, Anntharshika & Deegalla, 2013).

3.8. Conclusiones

Realizando una pequeña comparativa entre los problemas más comunes de las interfaces -de acuerdo con (Ehlert, 2003), (Zuffi y col., 2007) y (Carroll, 1988)- y las soluciones aportadas por las distintas metodologías analizadas, tenemos que para los siguientes casos:

- (1) El uso de opciones de menú con un orden erróneo y una separación confusa.
- (2) Inapropiado e inconsistente uso de colores, sonidos, iconos, y demás contenido multimedia que no se ajusta a la cognición del usuario.
- (3) Ilegibilidad de texto debido a problemas de apariencia.
- (4) Carencia de la propiedad de adaptación de la interfaz a los gustos del usuario.
- (5) Desaprovechamiento de los recursos computacionales asociados a la creación de interfaces.

Nota: Quedan exentos los siguientes problemas al salirse fuera del ámbito de este estudio:

- Inapropiada terminología o iconografía.
- Contenido erróneo de los textos.
- Ausencia de sistema de ayuda.

Los resultados en cuanto a solución de los problemas planteados son los siguientes:

	TRIDENT	PMT	AOP
(1)	No	Sí	No
(2)	No	Sí	Sí
(3)	No	Sí	No
(4)	Partial	No	No
(5)	No	No	Sí

Cuadro 3.2: Comparativa de solución a problemas de interfaz

Como se puede apreciar, las soluciones actuales quedan lejos de ser óptimas. Las tendencias más modernas giran entorno, como bien apunta (Lauren, Arnaud, Paul, Hervé & Olivier, 2018), a la utilización de machine learning a la hora de configurar las interfaces o bien a la recolección de datos del entorno mediante sensores, como en el ejemplo de AOP de (Sebek, Trnka & Cerny, 2015), para adaptarlas.

En (Martin-Hammond y col., 2018) se señala que la retroalimentación negativa obtenida (por ejemplo, botones demasiado pequeños o un volumen muy bajo en una interfaz auditiva) y comportamientos no deseados (por ejemplo, un elemento del menú que no nos lleva a donde queremos), se obtuvo de la experiencia del usuario, también debe tenerse en cuenta al planificar la interfaz, luego todos estos datos deben cuantificarse y analizarse.

Como se puede observar, todas las líneas de investigación apuntan a la recopilación y análisis de los elementos parametrizables de los patrones de comportamiento del usuario frente a las aplicaciones (tiempos, repeticiones, etc.) para construir interfaces intuitivas y efectivas.

Por un lado, las técnicas de gamificación son muy útiles para calibrar las propuestas generadas como resultado del análisis de estos datos, y por el otro, todas las técnicas utilizadas en la última década para la definición de interfaces -que se resumen en (Kirisci & Thoben, 2018)- indican que es necesario crear una taxonomía que permita clasificar los elementos de una interfaz si se desea trabajar con ellos para adaptarlos al comportamiento del usuario.

Poner estos elementos juntos en un sistema que proporciona a las interfaces un mecanismo para adaptarse al comportamiento del usuario (siempre basado en las limitaciones funcionales de la interfaz en sí) puede ser la solución técnica automatizada y desatendida para proyectos donde no es posible tener un equipo con personal dedicado a UX.

Por otro lado, el diseño semiautomático agregado al aumento de los procesos

de aprendizaje automático -ver (Lauren y col., 2018)- al configurar las interfaces o la recolección de datos ambientales utilizando sensores -en (Sebek y col., 2015)- son los avances más significativos hasta la fecha. La planificación y el tratamiento de las experiencias de los usuarios, tanto positivas como negativas -ver (Kirisci & Thoben, 2018)-, se deben tener en cuenta a la hora de planificar la interfaz. Parece asumido que sin los recursos necesarios (como equipos multidisciplinarios) es imposible idear una interfaz adecuada para grupos de usuarios no homogéneos y que las experiencias de los usuarios varían mucho dependiendo de cada arquetipo. Por esta razón, en (Kirisci & Thoben, 2018) señala como acciones fundamentales, el aprovechamiento y el análisis de los elementos parametrizables de los patrones de comportamiento del usuario para construir interfaces intuitivas y efectivas. A esto debemos agregar la ubicuidad de las interfaces, por lo que el procedimiento de descripción se convierte en un punto crítico para analizarlas y modificarlas, resulta imprescindible mejorar el servicios de cara a los usuarios, y conviene por tanto también diseñar mecanismos que ayuden a definir la interfaz de una forma más completa aprovechando los estándares expansibles de los que ahora disponemos.

Los Ecosistemas de Interfaz Adaptativos

4.1. ¿Qué es un Ecosistemas de Interfaz Adaptativo?

Básicamente una interfaz es un punto de contacto entre dos zonas distintas de un mecanismo que requieren comunicarse, son captadores y proveedores de datos, traductores, al fin y al cabo. Pero, por ello mismo, si la mutación y expansión de los lenguajes es algo usual ¿por qué por naturaleza las interfaces son rígidas, y no son ellas mismas las que modifican sus procesos de captación y proyección de datos con el tiempo? En la naturaleza las interfaces se modifican. Un ser humano es capaz de generar lenguaje corporal, los gestos o el tono de su conversación dependiendo de quien tenga delante, ya sea niño o adulto.

Pensemos que una interfaz de usuario es como un hábitat natural, donde sobrevive sólo el más fuerte, el más adaptado. Pensemos también, que pese a que el diseñador de cualquier aplicación crea que es lo mejor para el usuario, bien basado en su experiencia o bien basándose en las pruebas empíricas y estadísticas que avalan su teoría, el usuario es un ser autónomo con una forma de proceder, actuar y pensar propia. Añadamos otro pensamiento a nuestra teoría: el usuario es parte de una interfaz de usuario, pero en ella no es parte integrante como un sujeto vivo dentro de un ecosistema, sino que es más un elemento atmosférico, una fuerza natural a la que los elementos del sistema deben adaptarse para fortalecer el sistema mismo y hacerse fuerte con el paso del tiempo. Bien, ya estamos en el punto de partida, bienvenido a los Ecosistemas de Interfaz Adaptativos (AIE).

Si cada elemento de una interfaz es considerado como un individuo que se está adaptando a un sistema, está claro que en su mecanismo debe llevar implementado la funcionalidad necesaria para reconocer los eventos del sistema y, con un objetivo

claro, cambiar su forma de proceder para hacer que el sistema en su conjunto sea más capaz de afrontar la próxima eventualidad con mayor diligencia.

La evolución natural es lenta, normalmente hacen falta generaciones de individuos para que una especie altere su estructura genómica y no siempre la mutación desarrollada por los individuos implica que ésta sea un progreso para su especie. A menudo es todo parte de una estrategia de ensayo y error. Si cada especie pudiera acelerar su proceso de adaptación, tanto en tiempo como en acierto de sus mutaciones, obviamente aseguraría más rápidamente su supervivencia y optimizaría el uso de los recursos que tiene a su alcance. Por otro lado, si una especie no está indicada para al entorno (¿qué va a hacer una foca en el desierto?, prolongar su vida allí sería alargar su sufrimiento) lo mejor es que minimice el uso de recursos o que, si no hubiera más remedio, desaparezca.

La teoría de Darwin y Wallace es aplicable no solo a entornos naturales sino también a artificiales. Es difícil quitarle la razón a un mecanismo que lleva aplicándose con éxito en la tierra más de 4.500 millones de años.

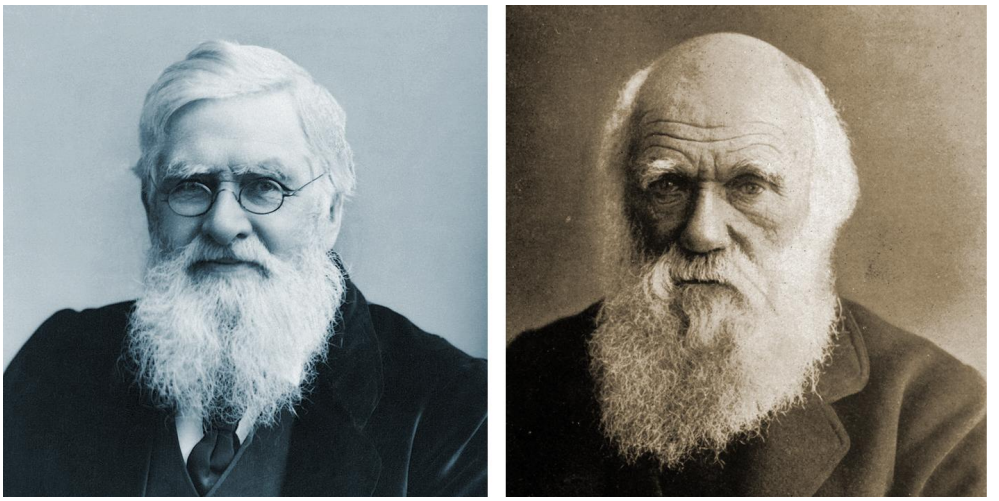


Figura 4.1: Alfred Russel Wallace (1823-1913), izquierda y Charles Robert Darwin (1809-1882), derecha

Para llevar la teoría natural a las interfaces gráficas debemos hacer el siguiente símil: cada elemento de una interfaz es un ser vivo independiente tratando de adaptarse al medio. El medio en este caso proveerá de servicios al individuo como propiedades físicas para su representación y cada individuo deberá realizar un servicio al medio, es decir su funcionalidad.

El usuario por su parte representa el medio ambiente, y sus necesidades serán los requisitos del sistema. Desde el punto de vista más poético será el *deux ex machina* al que todos los sujetos del ambiente deben someterse y adaptarse.

La supervivencia en el ambiente depende por tanto de dos factores: la buena utilización de los recursos y la utilidad de la funcionalidad al usuario.

4.1.1. Un botón en la selva

Ahora que ha quedado claro qué relación tiene el ambiente con la interfaz, veamos qué propiedades deben tener los individuos que la pueblan. Pongamos por ejemplo un botón de una interfaz gráfica. El botón tiene ciertas propiedades físicas (color, tamaño, tipo de tipografía, tamaño de la tipografía, *padding*, *margin*, tipo del borde, animación, tamaño del borde, etc.) que se definen en la creación del botón y que implican un consumo de recursos de interfaz del usuario en varios sentidos: el botón consumirá espacio en memoria (o en disco cuando la aplicación pase a segundo plano) y tiempo de procesador (de la CPU o de la tarjeta gráfica) cada vez que se pinte en pantalla. Además el botón tiene asociada una cierta funcionalidad que también se carga en memoria.

La "supervivencia" del botón dentro de la interfaz depende de lo importante que sea su funcionalidad para el usuario. Si su funcionalidad es más importante podrá ir cogiendo más recursos visuales (color, tamaño, posición, etc.) para tener más importancia dentro de la interfaz, eso es ir aumentando su **pregnancia**.

La modificación de la pregnancia del botón significa la modificación de las propiedades visuales que hemos visto antes para que al usuario le sea más sencillo acceder a él. La modificación de algunas de las propiedades, como por ejemplo el tamaño del texto, tienen una "solución universal" (aumentar el tamaño del texto significa ganar pregnancia y disminuirlo perderla), pero para otros es necesario conocer características extra del usuario: cultura, geografía, etc. La posición de los elementos no tiene la misma importancia para los usuarios del hemisferio oriental que para los occidentales, donde derecha e izquierda (debido a su orden de lectura) tienen significados opuesto -véase el apartado de la Gestalt-.

Figura 4.2: Elemento con mayor pregnancia en Europa occidental. Botón con mayor pregnancia en Japón

Para que cada elemento o componente de la interfaz modifique su pregnancia, (concepto que más tarde quedará definido como **mutación**), se han definido ciertos conceptos a él asociados:

- **Interacciones**. Cada evento en que el usuario proporciona algún tipo de información al sistema, ya sea de forma directa o indirecta es una interacción. De tal forma que se puede utilizar un modelo de dominio similar al de (Feng & Liu, 2015), para almacenar la información.

- **Memoria**. Cada componente debe evolucionar con el tiempo, ello implica que debe tener consciencia de su estado y anotar si los cambios producidos tienen consecuencias positivas o negativas para él.

- **Comunicación**. Cada elemento debe poder comunicarse con el resto de componentes, puesto que las medidas que tome sobre su propia capacidad de respuesta no tienen sentido si puede no establecer una escala. Por ejemplo, un botón no podría determinar si una pulsación sobre él es poco o mucho sin saber si sobre el resto de componentes se hace alguna acción.

- **Límites**. De alguna forma cada componente debe saber en qué rangos debe poder modificarse (no queremos que una caja de texto inunde toda la pantalla o que un enlace sobre política de privacidad desaparezca de una web porque nadie acceda a su contenido).

Existe una gran diferencia entre los componentes de una interfaz y los sujetos de un determinado ecosistema: mientras que los segundos tienen como objetivo

primario la supervivencia propia y como segundo el de su especie -ver (Freeman, 1977)-, los primeros deben encargarse de que la interacción que finaliza en la obtención de datos, se produzca de la manera más rápida y cómoda para el usuario.

4.2. El ritmo evolutivo

Un ecosistema es, después de todo y en su conjunto, un gran ser vivo que evoluciona con el tiempo. Su evolución, como ocurre con los elementos de la naturaleza, proviene de la mutación de los elementos que la conforman.

En los sistemas naturales, el ritmo evolutivo puede ser a corto o largo plazo, pero normalmente imparable -ver (de Oliveira, Medeiros, de Bragança Terra & Quelhas, 2011)-, sin embargo, en un sistema digital, la evolución constante se puede hacer de manera cronometrada, es decir, podemos decidir cuándo congelar o cuándo estimular el ritmo evolutivo del sistema.

Debe quedar claro que los elementos AIE no son elementos de la interfaz, sino -volviendo al símil natural- parásitos de estos. Existirá un elemento AIE por cada elemento de la interfaz que pueda mutar (podrán existir elementos de la interfaz que por distintas razones no deban modificarse). En términos de relaciones entre entidades será 1:1.

El elemento AIE será el que induzca al cambio al elemento de la interfaz, debiendo proporcionarle este una interfaz para que esto sea posible.

Al igual que la evolución en especies naturales -ver (Tingting Gao, Liu, Liu & Li, 2018)-, la evolución de los componentes de un AIE tiene tres fases bien definidas, con una obvia correspondencia asociativa:

- **Desarrollo / Asociación**. Durante la fase de desarrollo, el componente AIE se asocia con un componente de la interfaz. Durante esta fase, los componentes recopilarán los datos que consideren pertinentes de la interfaz.

- **Aprendizaje / Maduración**. En la maduración cada elemento AIE modificará sus propiedades internas a partir de los datos recopilados.

- **Integración / Mutación**: El componente AIE muta, modifica su pregnancia, y fuerza la transformación de su elemento asociado de la interfaz de acuerdo a ese incremento o disminución de pregnancia.

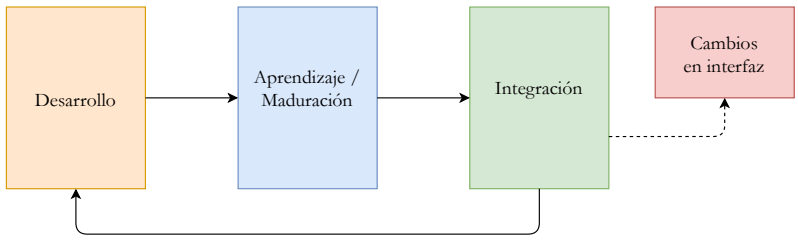


Figura 4.3: Proceso de evolución de un AIE

El proceso es circular e iterativo, pero la duración de las etapas puede no ser igual entre ellas mismas ni entre cada iteración. Lo más normal es que la etapa más larga sea la maduración, pues depende del número de interacciones del usuario.

4.2.1. La configuración de los límites

Como se ha mencionado anteriormente la formalización de límites es una tarea prioritaria en el diseño de una AIE. Antes de la entrada en funcionamiento, previo a la etapa de **desarrollo**, de un AIE se ha de definir la pregnancia de cada elemento y de los recursos disponibles, la velocidad de mutación y la función de mutación.

El hecho de que los elementos de la interfaz cambien libremente puede entrar en conflicto con las consideraciones estéticas (estéticas en su más amplio rango, no solo visual que es el más comúnmente tratado). Por ello el establecimiento de los recursos disponibles de una AIE no tiene porqué ser el del total del sistema, esto debe determinarlo un diseñador de UX. Los límites de crecimiento y decrecimiento de la pregnancia de los elementos del sistema deberán ser establecidos por el diseñador de la interfaz de este, de tal forma que el aspecto estético no se vea afectado ni se invada el espacio de otras funcionalidades del sistema a los que no aplique un IEA. Este tipo de decisiones, que son del tipo: tamaño máximo de las cajas de entradas del sistema, tamaño mínimo de los textos o de las imágenes, volúmenes máximos y mínimos de los sonidos del sistema, etc. son las cotas que debe establecer el diseñador de UX a la hora de implantar un sistema IEA.

4.2.2. Generaciones digitales

Una mutación en la naturaleza es "*un cambio en la secuencia genética, y es la causa principal de la diversidad entre los organismos*" -ver (Orr, 2005)-. ¿Cómo trasladar ese concepto al mundo digital? El concepto de mutación digital no es nuevo -ver (Adami, 2006)-, lo encontramos, por ejemplo, en los virus. La diferencia entre las mutaciones de los virus y las de los elementos de una interfaz, es que mientras los virus pueden cambiar la estructura de su código (para que al replicarse sea más difícil de ser localizado por los antivirus) los elementos de una interfaz deberán modificar su apariencia externa y, cuando lo requieran, también su funcionalidad.

Las mutaciones biológicas de una especie causan efectos secundarios: si unas especies se adaptan mejor al medio ambiente que otras especies y logran prosperar, a menudo es habitual que lo hagan unas a costa de otras. En la naturaleza, este proceso se autorregula a lo largo del tiempo y es predecible gracias a las teorías de la dinámica del sistema, como las descritas por las ecuaciones de Lotka-Volterra -ver (JA & JA, 2011, 1)-. Desafortunadamente, en el entorno digital no existe la autorregulación, y los límites de las mutaciones deben moderarse antes de que se inicie el sistema -véase la sección "Límites".

¿Qué parámetros (visuales y funcionales) de un elemento de interfaz son susceptibles a mutaciones? Teniendo en cuenta la categorización más simple posible entre los elementos de una interfaz (elementos de entrada, elementos de salida, elementos de entrada-salida y elementos compuestos), los factores de mutación pueden reducir a la **pregnancia** y las **mutaciones de comportamiento**. La mutación de comportamiento no debe confundirse con la funcionalidad asociada con la interfaz del sistema, sino con la forma de actuar de la interfaz en sí (es la misma diferencia que la forma y la sustancia).

En una interfaz bien construida -según (Sulaiman & Sohaimi, 2010)- los usuarios avanzan paso a paso, desde el clasificado de la información más abstracta hasta la sección cuyo contenido es más específico, están estableciendo caminos o vías de acción, que posteriormente a medida que vayan conociendo los mecanismos internos de la interfaz, transformarán en atajos.

Para explicar un poco mejor lo referente a mutaciones de funcionalidad abor-

daremos el siguiente ejemplo: un usuario accede frecuentemente a la funcionalidad asociada a un segundo nivel de menú de una aplicación, de hecho lo hace más frecuentemente que algunas de las opciones dispuestas en el primer nivel de dicho menú (es decir, determina un camino de acción). La mutación normal en este caso sería la inclusión del elemento del segundo nivel de menú dentro del primer nivel, en detrimento de alguno de los elementos de primer nivel de menú menos utilizados. Pueden considerarse también como mutaciones de funcionalidad la preselección de elementos o los cambios en las sugerencias de datos de entrada. Con la mutación, el sistema ha construido automáticamente el atajo sin necesidad de que éste fuera definido por el usuario.

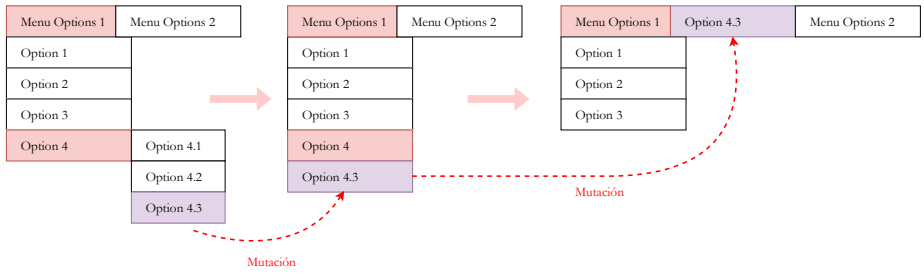


Figura 4.4: Fases de la evolución de un elemento de menú

Como se ha comentado previamente, las mutaciones de un elemento de una interfaz difieren en su intención última frente a las mutaciones de la naturaleza. Las mutaciones biológicas tienen como objetivo mejorar la adaptación del individuo al medio llevándole a una posición más favorable, las mutaciones digitales en este caso no favorecerían al individuo sino al usuario final de la interfaz. Esto nos lleva a considerar que las mutaciones no serán "favorecedoras" en todos los casos, y que podemos encontrar elementos que pierden pregnancia, para favorecer a otros.

Para que un elemento pueda saber exactamente cuándo mejorar sus condiciones o cuándo retraerse frente al resto de elementos, debe tener consciencia del estado de éstos, por ello el mecanismo de comunicación entre los elementos es muy importante -véase la sección de "Comunicación"-; pero de igual forma deben tener consciencia de las condiciones del entorno, es decir los recursos disponibles.

4.2.2.1. Pregnancia y energía

Puede parecer evidente considerar que si un elemento gana pregnancia dentro de la interfaz a través de una mutación, disminuya la del resto de elementos en un grado proporcional para compensar, de tal forma que la suma total de las pregnancias (P) de elementos antes y después de la mutación sea igual, para que el sistema continúe estable.

Considerando esto último tenemos la primera constante que rige los Sistemas de interfaces adaptativos (al igual que la energía en el universo, la pregnancia de una interfaz no varía entre un estado y otro):

$$\sum P_{t-1} = \sum P_t \tag{4.1}$$

en la ecuación anterior no se debe entender por P únicamente la pregnancia de los elementos del sistema. Si bien el crecimiento de la pregnancia de los elementos de un sistema no es infinita, pueden existir estados en los que los recursos de un sistema no se hayan explotado por completo y la suma total de las pregnancias de los elementos sea distinta antes y después de la mutación. Eso nos lleva a la plantear la siguiente pregunta: ¿Existe pregnancia que no se utiliza?

Un ejemplo sencillo de este tipo de estados puede ser el siguiente: consideremos una aplicación software con menús, en la que un elemento del menú de segundo nivel es nuevamente más utilizado que uno de primer nivel y por tanto debe 'promocionarse' hacia el primer nivel. El comportamiento esperado en este caso es que uno de los componentes del primer nivel pase a ser del segundo para contrarrestar la mutación, pero esto sólo sería necesario si el número total de elementos de primer nivel hubiera llegado a su tope, en caso contrario únicamente se verían incrementado los elementos de menú de primer nivel.

Utilizar los menús como ejemplo nos puede llevar a pensar incluso que la pregnancia total de un sistema (considerada como la suma de la pregnancia de sus elementos) puede llegar a ser infinita: podemos llegar a tener un número infinito de niveles de menú, pero ello claramente no es lo deseable para una interfaz, ni tampoco es un caso real que una interfaz tenga infinitas funcionalidades[1]. Por ello,

[1]Existen además una serie de consejos/buenas prácticas para el diseño de interfaces que deben tenerse en cuenta incluso antes de aplicar los Ecosistemas de Interfaz Adaptativos, una de ella es la

antes del diseño de la implementación de un AIE resulta inevitable la definición de límites.

Como consecuencia de la evidencia de recursos no utilizados de un sistema podemos redefinir la ecuación anterior de la siguiente forma (donde *Pu* es la pregnancia asociada a los recursos no utilizados):

$$Pu_{t-1} + \sum P_{t-1} = Pu_t + \sum P_t \tag{4.2}$$

4.3. Ecosistemas y ambientes

Los ecosistemas naturales son mundos complejos, no solo conviven distintos tipos de individuos, sino que podemos encontrar distintos tipos de hábitats o ambientes. En un río podemos observar tanto concentraciones de hormigas viviendo en en pequeños hormigueros en un remanso, como *Austropotamobius italicus* (cangrejos de río) morando en comunidad en la grava del fondo del río. Cada ambiente que conforma un ecosistema tiene sus propias normas e idiosincrasia. De igual forma en el entorno digital puede convivir en una misma pantalla un menú, un formulario y una gráfica con controles, que al fin y al cabo son subambientes.

Parece evidente, pero es importante también señalar que por su naturaleza no todos los elementos de una interfaz tienen prestaciones similares y por tanto los cambios en las pregnancias de los elementos deben afectar solamente a sus competidores directos; es decir no tendría sentido que, por ejemplo, en una aplicación web el intercambio de elementos de menú entre diferentes niveles afecte al tamaño del texto de las cabeceras o al color de los elementos del pie de página. Dentro de una interfaz podemos definir:

limitación de elementos de menú y de subelementos a 7+/-2. Para más información consultar los estudios de George Miller en 1956 en (Knight, 2019).

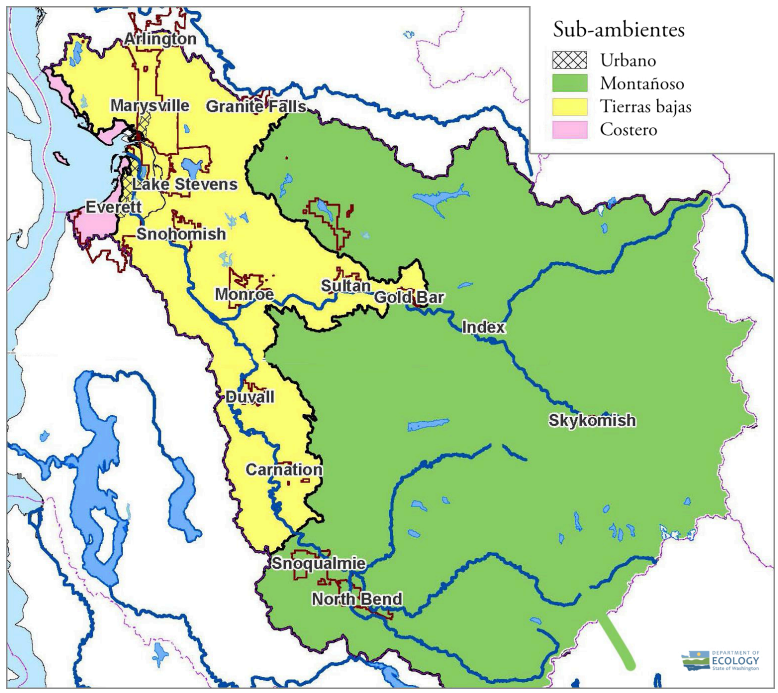


Figura 4.5: Sub-ambientes naturales del condado de King, Washington (EE.UU), extraido de (Hume y col., 15)

Un ambiente es el conjunto de elementos de una AIE, cuyos elementos referenciados en la interfaz forman parte de un conjunto de elementos iguales y cuya pregnancia se ve afectada directamente por la mutación de cualquiera de los elementos de ese conjunto (elemento que puede ser a su vez otro ambiente).

Cabe también señalar que cada elemento de un AIE, debe pertenecer a uno y solo a un ambiente (relación 1:N) y que, por necesidad, un AIE debe contener al menos un ambiente.

Los ambientes pueden ser considerados también como elementos de un AIE (es lo que podríamos denominar *subambientes*) de tal forma que pueden mutar y competir entre sí en caso de ser necesario. No tiene sentido que la mutación de un elemento de menú afecte a una caja de entrada de texto de un formulario, pero sí que el tamaño del espacio asignado al menú se vea afectado si el tamaño del menú que se encuentra encima se incrementa.

$$P_{ambient} = \sum_{i=1}^{n} P_i + Pu_{ambient} \tag{4.3}$$

donde:

P_i = pregnancia del elemento *i* del ambiente

n = número de elementos del ambiente

Considerado como un elemento de un sistema, los ambientes pueden también agruparse entre sí dando lugar a nuevos ambientes. Por ello la pregnancia de un sistema viene determinada como la suma de las pregnancias de cada uno de los ambientes de primer nivel (aquellos que cuelgan del nodo raíz del árbol de pregnancias) más la suma de la pregnancia de los recursos no utilizados del ambiente más externo:

$$P_{system} = \sum_{k=1}^{r} P_k + Pu_{system} \tag{4.4}$$

donde:

P_k = pregnancia del ambient *k (de primer nivel)* del sistema

r = número de elemento de primer orden del sistema

Si un elemento de una interfaz es más utilizado que otro, lo ideal es que este gane pregnancia frente al resto -ver (Gould & Lewis, 1985)-. Bajo esta base y para estar de acuerdo con lo descrito anteriormente, la variación de la pregnancia de un elemento (P_i) debe ser igual a la suma de los decrementos de las pregnancias del resto de los elementos más el decremento de la pregnancia no utilizada (asociada a los recursos no consumidos), de tal forma que la pregnancia del total del sistema no varíe, aunque sí pueda variar la composición de sus ambientes. Es decir P_{system} es constante, pero las $P_{ambient}$ no, varían en función del tiempo:

$$P_{system} = \sum_{k=1}^{r} P_k(t) + Pu_{system}(t) = \sum_{k=1}^{r} P_k(t-1) + Pu_{system}(t-1) \tag{4.5}$$

$$P_{ambient}(t) = \sum_{k=1}^{n} P_i(t) + Pu_{ambient}(t) \tag{4.6}$$

$$P_{ambient}(t) = \sum_{k=1}^{n} P_i(t-1) + Pu_{ambient}(t) + \sum_{k=1}^{n} \Delta P_i(t-1) \qquad (4.7)$$

Si se representara gráficamente la pregnancia, puede utilizarse un TreeMap puesto que la pregnancia global del sistema es invariable -para más información sobre TreeMaps consultar (N, J & M, 2010, 6)-. A continuación podemos ver un ejemplo, donde los elementos AIE están marcados en rojo y la pregnancia libre en gris:

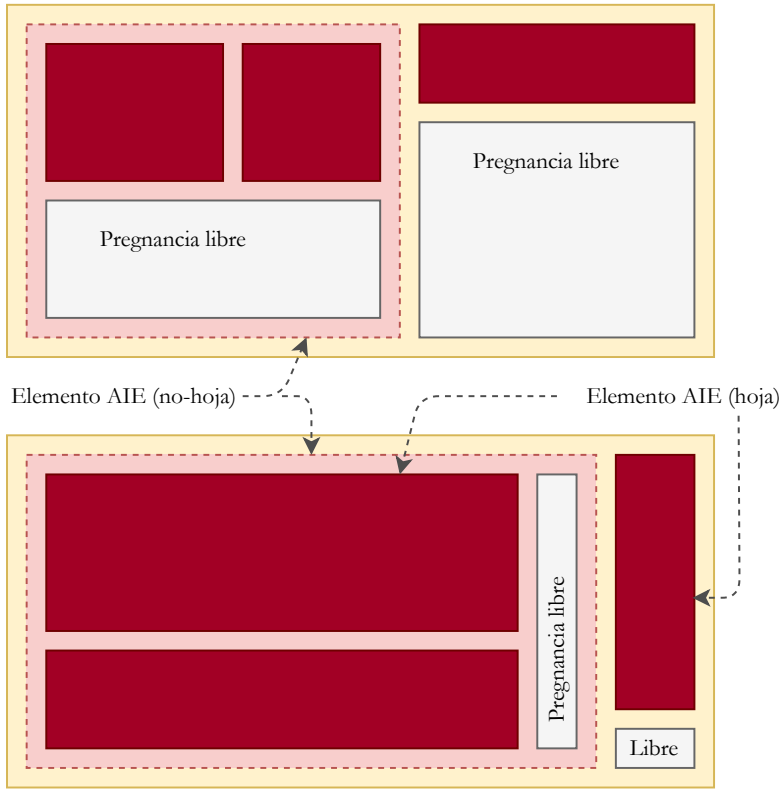


Figura 4.6: Representación de la pregnancia como TreeMap

4.3.1. Los recursos

Como hemos visto antes pueden existir dos tipos de recursos en sistema: los recurso **computacionales**, relacionados con la funcionalidad, y los recursos **re-**

presentativos, relacionados con la representación física de los elementos.

Dentro de un AIE, solamente tiene sentido definir como "recurso" a un recurso o propiedad de representación de un elemento, pues será aquello afectado por los cambios de pregnancia y a lo que repercutan las mutaciones de los elementos del ambiente.

Los recursos **representativos** pueden ser bien propiedades estéticas (tamaño, brillo, contraste, tono, saturación, posición, etc.) si estamos tratando con una interfaz visual, bien propiedades sonoras (tono, volumen, etc.) si se trata de una interfaz sonora, etc. -véase las tablas 3.1 y 3.1-. En resumen, un recurso es una propiedad física que proporciona pregnancia a un elemento de la interfaz, que pueda ser detectada por los sentidos y mensurable (por tanto no se deben considerar solamente propiedades visuales, la intensidad de un umami[2] puede ser considerado también un recurso si consideramos elementos de una interfaz gustativa).

Los límites de los recursos los definen generalmente los espacios de representación: el tamaño, resolución o profundidad de color de las pantallas para las interfaces visuales (o el tamaño de las áreas de ejecución de las aplicaciones), la profundidad y calidad de sonido de los altavoces en las auditivas, etc.

Un ambiente deberá dar rienda suelta a las necesidades de sus elementos hasta el momento en que haya conflicto y no queden más recursos disponibles, en ese caso, como no puede ser de otra manera, el reparto se hará en función de la pregnancia de cada elemento, es lo que representará las necesidades del usuario. A más pregnancia mayor consumo de recursos.

Debe existir un mínimo de recursos que un elemento debe consumir. Pero puede llegar un determinado momento en que la pregnancia del elemento es tan baja que forzará a ese elemento a desaparecer de la interfaz porque desde el punto de vista físico no tiene sentido. Imaginemos por ejemplo una caja de entrada de texto para la introducción de un apellido, ¿que sentido tendría tener esa caja en pantalla con un tamaño de $10px$x$20px$? Sería completamente inutilizable.

[2] La palabra umami proviene del idioma japonés y no tiene una traducción directa, lo más similar sería "sabroso". El umami es uno de los cinco sabores básicos junto con el dulce, ácido, amargo y salado.

4.3.2. El reparto de los recursos

En esta sección se presenta el algoritmo de reparto de recursos que seguirá un AIE. Suponemos que existe un recurso W, finito y divisible en porciones. Podríamos llegar considerar que W es un subconjunto de N, suponiendo también que w es la unidad de ese subconjunto.

$$W = \{1w, 2w, ...(n-1)w, (n)w\} \tag{4.8}$$

Si suponemos que *P* es la pregnancia inicial de un ambiente, *Pu(t)* la pregnancia libre del mismo ambiente en el momento *t*, $P_i(t)$ la pregnancia del elemento *i* tras la mutación *t*. R los recursos totales del ambiente, *Ru(t)* son los recursos libres del ambiente tras la mutación *t*, $R_i(t)$ los recursos utilizados por un elemento tras la mutación *t*, con ello determinamos que:

$$R = \sum R_i(t) + Ru(t) \tag{4.9}$$

y por tanto, considerando que los recursos son finitos y no varían:

$$\sum R_i(t-1) + Ru(t-1) = \sum R_i(t) + Ru(t) \tag{4.10}$$

Si a esto le añadimos que cada elemento AIE puede (y en muchos casos debe) tener establecidos límites de representación -ver apartado "Límites"-:

$$R_i(t) = \kappa(\frac{P_i(i)}{P_i(0)} R_i(0), Rmax_i(t), Rmin_i) \tag{4.11}$$

Siendo:

$Rmax_i$ = máximo número de recursos consumidos por el componente *i*.

$Rmin_i$ = mínimo número de recursos consumidos por el componente *i* para ser representable.

La función *kappa*(κ) será una función no lineal que vendrá dada por:

$$\kappa(x, Rmin, Rmax) = \begin{cases} 0 & x < Rmin \\ x & x \leq Rmax, x \geqslant Rmin \\ 0 & x > Rmax \\ 0 & Rmax < Rmin \end{cases} \tag{4.12}$$

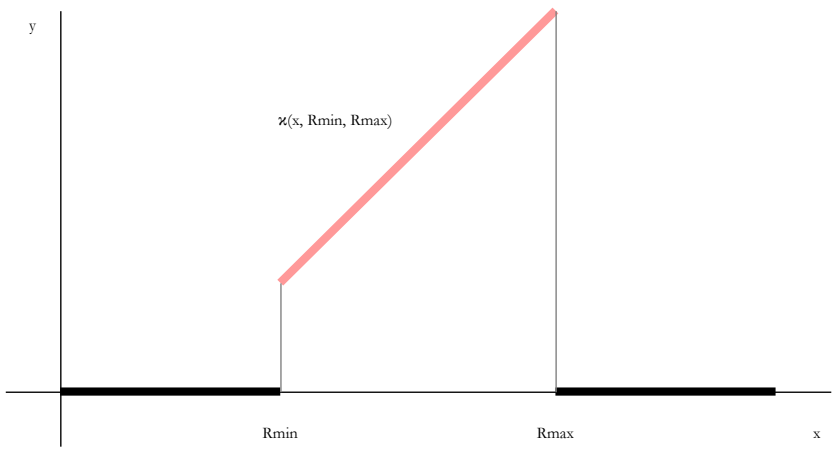


Figura 4.7: Representación de la función kappa de cálculo de recursos

Para cumplir la invariabilidad de los recursos disponibles entre una mutación y otra, el valor de los $Rmax_i$ debe ser previamente calculado, de tal forma que se cumpla:

$$R \leq \sum_{i=1}^{n} Rmax_i \tag{4.13}$$

para ello si hay conflicto (si se superan los recursos disponibles) entonces

$$Rmax'_i(t) = \frac{P_i(t)}{P} * R \tag{4.14}$$

y así:

$$Rmax_i(t) = f(Rmax_i(0), Rmax'_i(t)) \tag{4.15}$$

$$f(x,y) = \begin{cases} x & x < y \\ y & x \geq y \end{cases} \tag{4.16}$$

4.3.3. Memoria y almacenamiento

Como todo componente cualquier ser vivo dentro de un ecosistema natural, un elemento de EIA debe poseer una memoria (aunque sea tan muy pequeña) que le permita conocer su propio estado y el del ambiente para decidir qué acciones tomar cada vez que el usuario interaccione con la interfaz. La memoria debe almacenar información sobre los datos de los recursos consumidos por los elementos del entorno. No es necesario que cada elemento almacene los datos del resto de compañeros del ambiente al detalle, pero como se verá más adelante, le será muy útil conocer estado general de su entorno para para agilizar los cálculos.

Dentro de la memoria cada componente deberá almacenar también los límites que afecten a sus capacidades de mutación.

4.3.4. Las mutaciones

De inicio, el diseñador de la interfaz considerará que existen elementos de menor o mayor pregnancia, puesto que es imposible diseñar un sistema donde todos los elementos sean percibidos por el usuario con la misma consideración. En una interfaz visual, por ejemplo, su posición hace decantar al usuario hacia una determinada categorización.

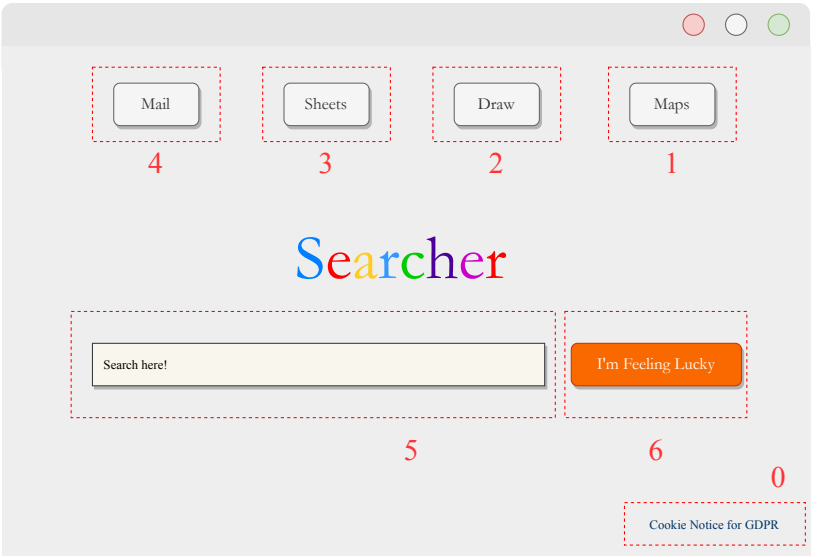


Figura 4.8: Asignación inicial de pregnancias de la interfaz (en color rojo)

¿Cuándo ha de mutar un elemento? ¿Cuándo ha de crecer o disminuir la pregnancia de un elemento? La mutación de la pregnancia es mucho más fácil de calcular que la de la funcionalidad al tener a disposición cantidades mensurables, pero es necesario determinar cuál es la cantidad mínima de pregnancia que puede ser alterada en una mutación -ver (Bundesen, Vangkilde & Petersen, 2015).

4.3.4.1. La unidad mínima de pregnancia

Al no ser la pregnancia una magnitud física concreta, sino que es la suma de cualidades de un objeto, la identificación de una unidad mínima no es trivial. Una unidad mínima de pregnancia podría ser un pixel si esta determina el tamaño dentro de una pantalla de ordenador o un decibelio si se trata de comparar sonidos; en resumen, el rango de propiedades que pueden afectar a la pregnancia de un objeto puede ser muy grande.

Un factor clave para determinar esta unidad mínima es que el incremento o decremento de pregnancia entre dos elementos solo puede ser determinado dentro de un ámbito donde ambos elementos puedan ser considerados iguales. Ese ámbito dentro de una EIA será el ambiente. Lo cual determina que todos los elementos dentro de un ambiente utilicen la misma unidad mínima de pregnancia.

La forma o fórmula de determinar la unidad mínima dentro de un ambiente, se realizará de forma manual, es decir, el diseñador/programador de la interfaz identificará las propiedades que modifican la pregnancia de los elementos del ambiente.

Pueden llegar a existir casos en la pregnancia sea una unidad compuesta, como por ejemplo la dupla *{altura en píxeles, intensidad de color rojo (*$0, 255$*)}* lo cual puede llegar a complicar los cálculos. Aunque en estos casos es posible también establecer una unidad mínima, como por ejemplo *{*$1px$*, una unidad de color rojo}*, no es lo deseable, aunque sí hay estudios que permiten este tipo de cálculos -como (Bundesen y col., 2015).

4.3.4.2. La velocidad de los cambios

La mutación no es por necesidad algo que se produzca cada vez que ocurre una interacción con el usuario. En los ciclos biológicos hacen falta generaciones para que una especie mute (aunque esto está condicionado fundamentalmente porque las mutaciones se producen durante la recombinación genética) y aunque esto no es necesario en un entorno digital, lo cierto es que un análisis más contrastado de las condiciones de uso de la interfaz llevará posiblemente a una comprensión mucho mayor de los procesos de interacción del usuario. Por ello, generar una mutación instantánea cada vez que se produzca una interacción con la interfaz no es lo más indicado.

Lo ideal es que elementos y ambientes utilicen su memoria para almacenar durante un periodo de tiempo los resultados de las interacciones del usuario y luego realizar estadísticas de utilización antes de tomar una decisión.

El proceso es exactamente el mismo que seguiría un experto en UX, hacer cambios en la interfaz en cada momento generaría un gran rechazo del usuario.

La velocidad de adaptación del sistema será algo que puede variar a lo largo del tiempo. Lo más habitual es que la interfaz tenga cambios más drásticos y frecuentes al principio y que luego pase a largos periodos de estabilidad, de igual forma que pasa con los sistemas biológicos. El método de aprendizaje de un perceptrón simple -ver (Malsburg, 1986)-, donde los pesos varían muy rápidamente al principio de su aprendizaje, puede ser resultar un modelo muy similar al esperado.

El concepto velocidad de mutación inicial, que denominaremos v_m será muy

similar al de momentum de una red multicapa y será una constante en la función de mutación.

Cabe aclarar que pese el sistema no tiene solamente una velocidad inicial de mutación, sino también una aceleración, que será la que condicione el estancamiento del sistema (para ver cómo se produce consultar la sección "La maduración: el fin de la mutación").

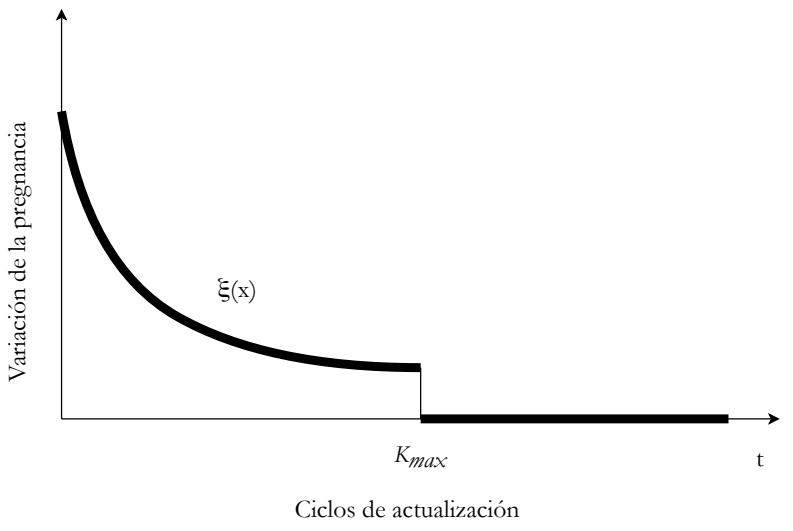


Figura 4.9: Maduración del sistema (ξ)

4.3.5. Las interacciones

Podríamos definir una interacción en un EIA como el evento que provoca una variación de pregnancia de un elemento. La naturaleza de ésta no depende de la propia interfaz, sino que está directamente ligada a cada elemento, es decir cada elemento del sistema podría tener asociada una interacción distinta.

En una interfaz visual de una aplicación software, podrían darse varios tipos de eventos sobre una caja de entrada: un click sobre ella, la posición de una tecla mientras el cursor se encuentra sobre ella, etc. Estos eventos, sin embargo, sobre un mapa, como por ejemplo un *plug-in* de Google Maps (que podemos ver con asiduidad en muchas páginas web), no tienen sentido. Sobre un mapa el usuario interacciona de forma muy distinta: puede hacer zoom, arrastrar un marcador,

hacer scroll, etc. Y si consideramos una interfaz de tipo sonoro el usuario puede interaccionar con los elementos mediante silencios o variaciones tonales.

La diversidad de eventos en los distintos tipos de interfaces es inconmensurable aunque no todos ellos tienen una importancia para un AIE. Sobre los elementos gráficos el evento de *MouseOver* (pasar el ratón por encima del componente) es muy usual pero normalmente poco relevante. La figura del desarrollador de componentes es muy importante para determinar qué eventos sobre un componente son valiosos y cuáles no.

Cada elemento debe, por tanto, saber lo que representa una interacción para sí mismo (un clic, la finalización de un campo, la aceptación de un orden de sonido, etc. dependerá de la naturaleza del elemento y del tipo de interfaz).

Por otro lado, una interacción no significa siempre una variación positiva de la pregnancia, sino que depende de la reacción, positiva o negativa, por parte del usuario. Por ejemplo, que un usuario haga *click* repetidamente sobre una opción de menú no quiere decir que esa opción sea la que quiere pulsar, también puede significar que la interfaz está de inicio mal configurada y genera por parte del usuario un fallo repetido. Por ello el *feedback* que devuelve el usuario cuando interacciona con un elemento es determinante a la hora de saber si debe producirse un incremento o decremento de la pregnancia.

El feedback puede ser recogido de forma **directa** (si es el usuario quien indica al sistema que su interacción ha sido involuntaria) o **indirecta** (si es el sistema quien recoge la respuesta no verbal del usuario).

Existen multitud de formas de recoger este feedback. En los trabajos de (Kharat & Dudul, 2008) y (Halder y col., 2012) se estudian las reacciones del usuario a través del reconocimiento de expresiones faciales gracias a redes de neuronas, PCA o SIFT.

La recogida de datos de forma indirecta no debe basarse solamente en características visuales propias del usuario, sino también de la propia interacción, pensemos por ejemplo en la desafección que en el usuario que pueden provocar recesos en las respuestas, variaciones de tono o variación de la frecuencia cardiaca.

4.3.6. La función de mutación (τ)

El momento álgido de la fase de maduración en AIE viene determinado por el **desarrollo**. En el proceso de mutación se recalculan las pregnancias de los elementos para la siguiente generación de acuerdo a:

- **La velocidad de mutación** (v_m): Determinará la velocidad de evolución del sistema. Será una constante definida por el usuario.
- **La función de mutación** (τ): Calcula la variación de la pregnancia partiendo del número de interacciones del usuario sobre el elemento en función de número de interacciones del usuario sobre el ambiente.

La pregnancia de un elemento *i* será determinada por la siguiente ecuación:

$$P_i(t) = \tau(v_m, \frac{in_i(t)}{in_{ambient}(t)}, P_{ambient}(t)) \tag{4.17}$$

donde:

$in_i(t)$ = interacciones del usuario sobre el elemento *i* en el momento *t*

$in_{ambient}(t)$ = interacciones del usuario sobre el ambiente en el momento *t*

La función de mutación la define el desarrollador en función del entorno al que se aplique un AIE, considerando que debe respetarse la preservación de la pregnancia entre estados.

En el caso más sencillo, donde $v_m = 1$ y τ fuera una función lineal tal que:

$$\tau\,(v, x, y) = v \cdot x \cdot y$$

tendríamos que:

$$P_i(t) = \frac{in_i(t)}{in_{ambient}(t)} \cdot P_{ambient}(t) \tag{4.18}$$

4.3.7. El inicio de los tiempos y el fin de la mutación

Cada vez que se produce una mutación y en el momento de puesta en marcha del sistema surge la necesidad de hacer un reajuste de las pregnancias de los elementos. Esto es debido a la asignación de recursos disponibles en función de los máximos y mínimos de estos. Si tras una asignación de recursos quedan recursos no utilizados, esto afecta significativamente puesto que en el sistema debe haber una cantidad equivalente de *Pu*.

Para poder realizar la asignación se debe seguir el siguiente procedimiento:

- Cálculo de la pregnancia de la forma indicada arriba.
- Asignación del recurso partiendo de los valores de la pregnancia calculados.
- Aplicación de restricciones a los valores del recurso.
- Cálculo de los recursos disponibles.
- Reasignación de la pregnancia de acuerdo con:

$$P_i'(t) = \sum_{j=1}^{k} P_i(t) \cdot \sum_{j=1}^{y} \frac{R_i^j(t)}{R_{ambient}^j(t)} \tag{4.19}$$

$$Pu'(t) = \sum_{j=1}^{k} P_i(t) \cdot \sum_{j=1}^{y} \frac{Ru^j(t)}{R_{ambient}^j(t)} \tag{4.20}$$

donde:

P'$_i$(t) = Pregnancia del elemento i en el ciclo t

$$R_{ambient}^j(t) = \sum_{i=1}^{k} R_i^j(t) \tag{4.21}$$

k = número de elementos del ambiente

y = número de tipo de recursos asociados al ambiente j

$$P'_i(t) = \textit{Pregnancia del elemento i en el ciclo t}$$

$$R^j{}_i(t) = \textit{Valor del recurso de tipo j asignado al elemento i en el ciclo t}$$

$$Rw^j(t) = \textit{Valor de los recursos libro de tipo j asignado al elemento i en el ciclo t}$$

La asignación inicial de las pregnancias, previa a la puesta en marcha del sistema, se puede hacer de igual forma, es decir, asignado los valores en función de los recursos iniciales asignados a cada elemento.

A continuación se muestra un ejemplo del cálculo de recursos y pregnancias en un sistema muy sencillo, en un ambiente con seis elementos (*E1 . . . E6*) y recursos libres (*Free*).

El sistema tiene 20 unidades de recursos para asignarse, asigna de forma inicial una unidad a cada recurso, dejando como recursos libres 14 unidades. Cada uno de los elemento tiene un máximo (R_{max}) y un mínimo (R_{min}) de recurso que puede utilizar (por ejemplo, en el caso de *E2* el mínimo es 1 y máximo 6).

Tras un proceso de aprendizaje del sistema, donde el elemento que más interacciones ha sido el elemento *E1* (con el 57,1 % de las interacciones, con lo cual le corresponden 11,429 unidades de recursos *-Res-*, sin tener en cuenta los máximos y mínimos), se hace el cálculo para la reasignación de recursos. En la asignación final de recursos los elemento copan un total de 14 unidades de recursos, y quedan 6 unidades como recursos libres del sistema.

	Estado inicial							Asinación	Estado final
Elem.	Res	R_{min}	R_{max}	Preg. (p)	Inter.	Preg (%)	Res	Res	Preg. (p)
Total	20	6	20	1	7	1,000	20,000	20	1
E1	1	1	4	0,05	4	0,571	11,429	4	0,2
E2	1	1	6	0,05	3	0,429	8,571	6	0,3
E3	1	1	3	0,05	0	0,000	0,000	1	0,05
E4	1	1	2	0,05	0	0,000	0,000	1	0,05
E5	1	1	4	0,05	0	0,000	0,000	1	0,05
E6	1	1	1	0,05	0	0,000	0,000	1	0,05
Libre	14			0,7				6	0,3

Cuadro 4.1: Cálculo de la pregnancia después de mutación. Caso simple, un solo recurso

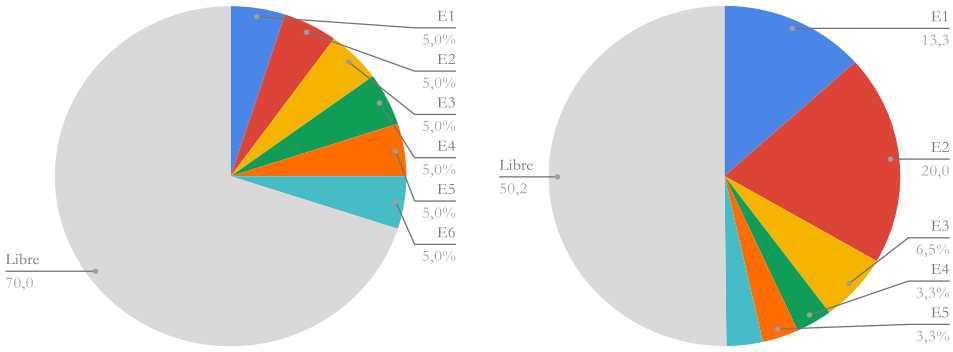


Figura 4.10: Representación la pregnancia antes (izquierda) y después de la mutación (derecha). Caso simple, un solo recurso

En el caso de tener varios recursos se deben tener en cuenta los máximos y mínimos de los dos:

	Estado inicial											Asignación		Estado final
	Recurso 1			Recurso 2										
	Res	R_{mir}	R_{max}	Res	R_{mir}	R_{max}	Preg. (p)	Int.	Preg. (%)	Res_1 (%)	Res_2 (%)	Res_1	Res_2	Preg. (p)
Total	20,00	6,00	20,00	12,00	6,00	12,00	1,00	7,00	1,00	20,00	12,00	20,00	12,00	1,00
E1	1,00	1,00	4,00	1,00	1,00	2,00	0,06	4,00	0,57	11,43	6,85	4,00	2,00	0,18
E2	1,00	1,00	6,00	1,00	1,00	2,00	0,07	3,00	0,43	8,57	5,14	6,00	2,00	0,23
E3	1,00	1,00	3,00	1,00	1,00	2,00	0,07	0,00	0,00	0,00	0,00	1,00	1,00	0,07
E4	1,00	1,00	2,00	1,00	1,00	2,00	0,07	0,00	0,00	0,00	0,00	1,00	1,00	0,07
E5	1,00	1,00	4,00	1,00	1,00	2,00	0,07	0,00	0,00	0,00	0,00	1,00	1,00	0,07
E6	1,00	1,00	1,00	2,00	1,00	2,00	0,11	0,00	0,00	0,00	0,00	1,00	1,00	0,07
Libre	14,00			5,00			0,56					6,00	4,00	0,31

Cuadro 4.2: Cálculo de la pregnancia después de mutación. Caso con varios recursos

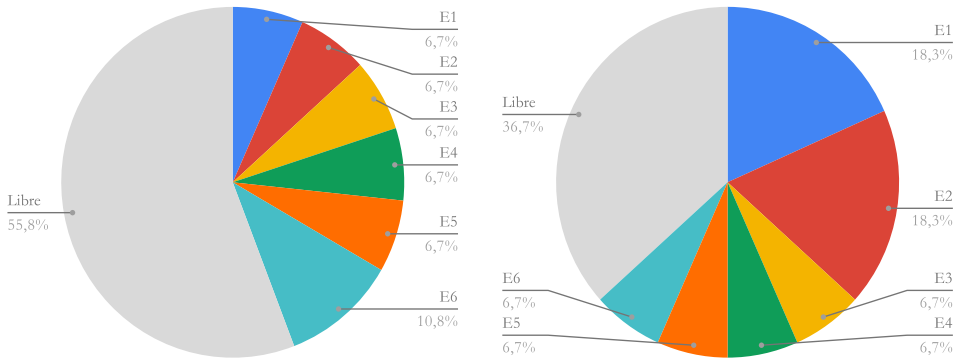


Figura 4.11: Representación la pregnancia antes (izquierda) y después de la mutación (derecha). Caso con varios recursos

4.3.8. La mutación de funcionalidad

Una mutación de funcionalidad es un cambio en la funcionalidad misma de un elemento, no radical, sino asociada a su funcionalidad inicial.

Las mutaciones de funcionalidad son fácilmente detectables por los humanos pero es muy difícil encontrar una solución global que resuelva todos los casos. La mayoría de las veces, las mutaciones funcionales se pueden enfocar como mutaciones de pregnancia, reevaluando las pautas que determinan las funciones de mutación de cada entorno.

Un claro ejemplo de mutación de funcionalidad es la que agrupará el resultado de varios procesos de cálculo. Este caso podría resolverse si, al momento de planificar la funcionalidad asociada con la interfaz, se decidiera incluir un elemento que realizaría el cálculo completo, y así poco a poco a través de mutaciones de pregnancia, llegaría a adquirir una posición relevante en relación a su uso.

Las mutaciones de funcionalidad por su complicación técnica quedarán fuera del ámbito de estudio de esta tesis.

4.3.9. Comunicación entre elementos de un AIE

4.3.9.1. Procesado de eventos

El procesado de eventos se produce en la fase de **aprendizaje/maduración.** En una AIE, la comunicación entre los elementos es providencial a la hora de plantear

el momento y la intensidad de las mutaciones. Cada elemento debe tener constancia de forma pormenorizada de su propia interacción con el usuario y forma generalizada de la relación del usuario con el resto de elementos de su ambiente, además debe ocuparse de la recoger dichas interacciones.

En cada interacción el elemento debe modificar los datos acerca de las interacciones, y por su parte el ambiente en que se encuentra comunicar al resto de elementos que se ha producido una interacción. Si el ambiente al que pertenece el elemento es parte a su vez de otro ambiente, debe realizar la misma operación. De esta forma cada interacción del usuario supondrá una sucesión de notificaciones en cascada desde el elemento con el que se interaccione hasta los elementos que componen el ambiente de 'primer nivel' del sistema. Notificación de actualización de pregnancia se produce de forma ascendente: desde el elemento en que se ha producido el evento hasta el Ambiente "raíz" del sistema. Mientras que la actualización de los valores se produce en sentido inverso, desde la "raíz" hasta las las "ramas".

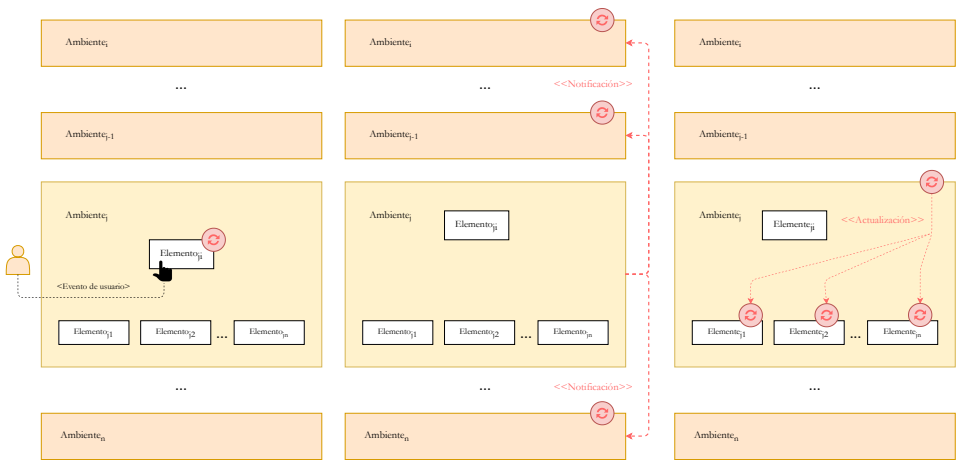


Figura 4.12: Notificación de interacciones

Una vez que todos los elementos están actualizados, se podría evaluar si han de producirse mutaciones. Eso implica que el proceso de actualización debe notificar a quien lo inicia que ha acabado.

Se ha de puntualizar que una mutación de pregnancia no se produce en el momento en el que haya un evento sobre la interfaz, se puede producir en cualquier

momento del ciclo de vida, de igual forma que los cambios en la interfaz, son ambos procesos completamente independientes del procesado y actualización de los eventos. El proceso de actualización de pregnancia es iniciado por usuario, mientras que la mutación no.

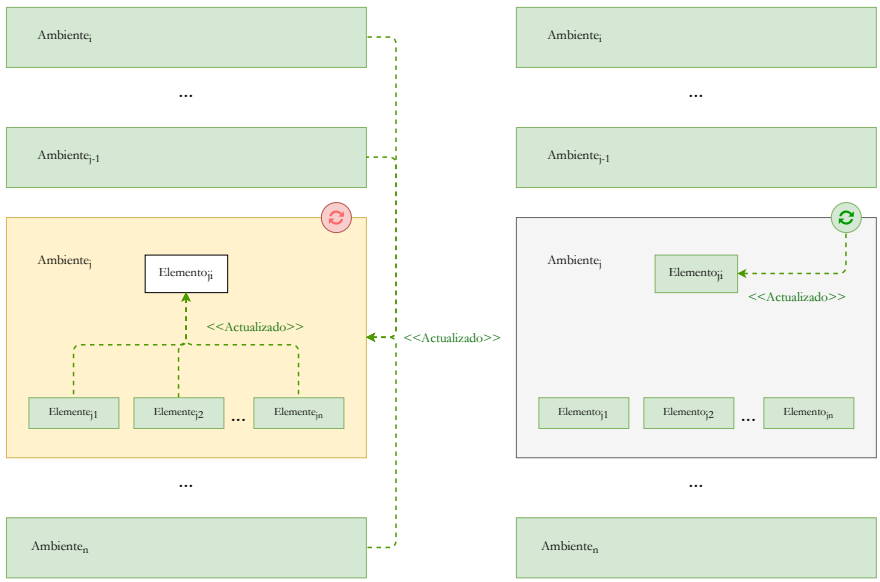


Figura 4.13: Difusión de las interacciones

Los cambios provocados por una mutación no deben afectar a ningún elemento fuera del ambiente del que se produzcan de forma directa, tanto si los recursos del sistema están siendo completamente aprovechados como si no, puesto que serán los elementos de cada subambiente los que compitan entre sí y será el elemeto subambiente que los contiene el que transmita el resultado de estos cambios al nivel superior.

Para tener control de los recursos no utilizados dentro de cada nivel, deberá existir un componente que los controle. Este componente puede ser o bien el propio elemento ambiente o bien otro componente aislado, dependerá de la implementación.

Ya que es posible tener interacciones de forma concurrente en el sistema, lo ideal es que exista un mecanismo de comunicación en el que se puedan encolar los mensajes sobre las interacciones y los mensajes sobre los procesos de mutación que

se lleven a cabo, sin que unos interrumpan a otros ni los cálculos se basen en datos desactualizados.

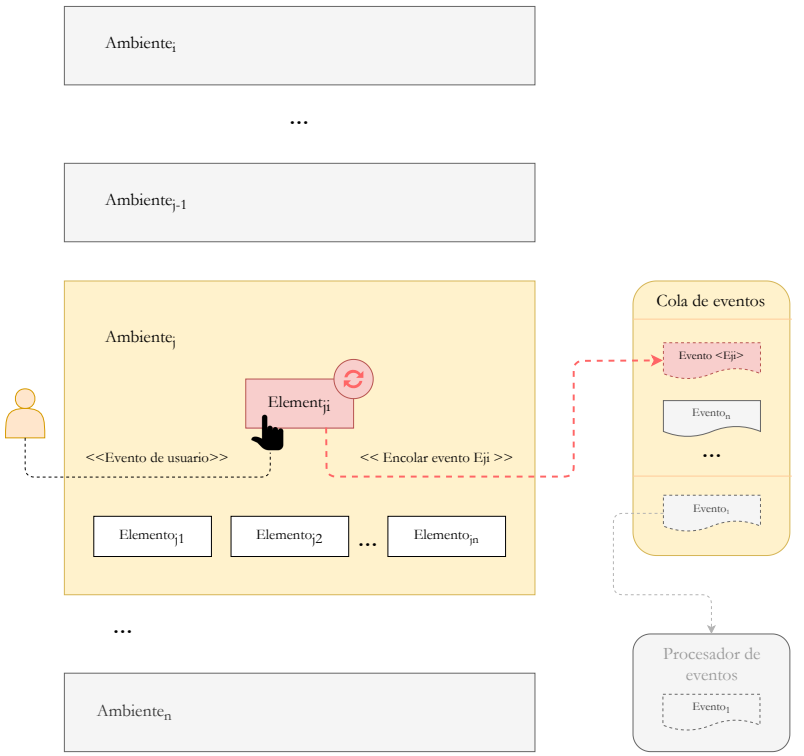


Figura 4.14: Interacciones, cola y procesador de eventos

Este mecanismo consiste en la inserción al proceso de dos componentes nuevos: una **Cola de eventos** y un **Procesador de eventos**. La actualización así se vuelve más lenta pero se garantiza que los eventos se procesen en el mismo orden en el que se producen y que sea consistente como consecuencia la actualización de las pregnancias.

Cada evento producido sobre el elemento *i* de un AIE será insertado en la cola, el procesador irá sacando de la cola los eventos en orden de entrada y actualizando los elementos del ambientes desde el nodo raíz hasta las ramas.

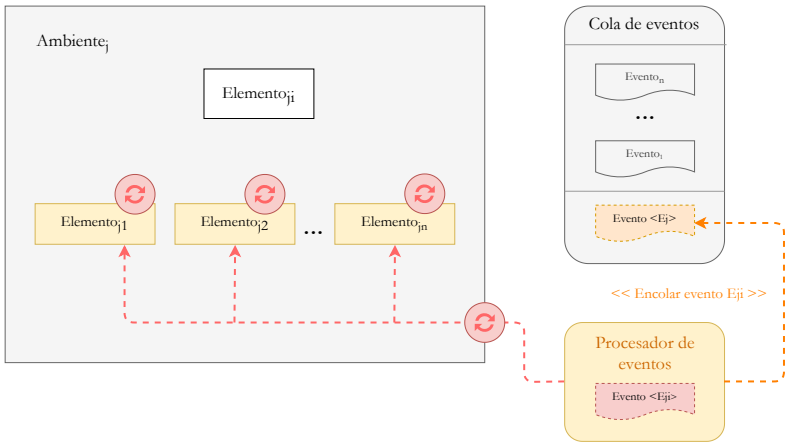


Figura 4.15: Procesando evento de la cola de eventos

Una vez finalizado, el Procesador de Eventos comprobará la cola y continuará con la actualización del siguiente evento.

4.3.9.2. Expansión de las mutaciones

El modo en que se aplican las mutaciones en el sistema es diametralmente opuesto al de la recogida de eventos. Como hemos visto en la sección anterior, la función de mutación necesita saber el valor actualizado de la pregnancia del ambiente del elemento que desea mutar.

La mutación comienza por los niveles superiores del árbol que configura un Ecosistema de Interfaz Adaptativo. No es posible por tanto mutar un elemento aislado del sistema, es todo el sistema el que cambia.

El sistema puede mutar en cualquier momento, siempre que se haya producido una interacción, puesto que si no no tiene información para hacer los cálculos de la nuevas pregnancias. Para asegurar la integridad de los datos no se debe interrumpir el procesado de eventos, se deberá esperar a que la cola de eventos esté vacía.

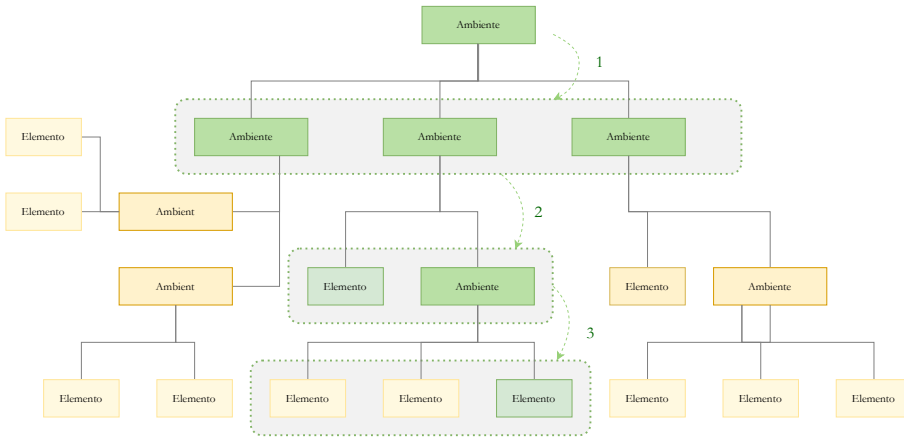


Figura 4.16: Proceso de las mutaciones

4.3.10. Propiedades grupales

A la hora de definir una interfaz nos damos cuenta que las propiedades físicas a las que hacemos referencia no son del elemento en las que comúnmente las referenciamos sino de su contenido. Es decir que si un elemento tiene una pregnancia X esa será la pregnancia a dividir entre su ambiente, teniendo en cuenta los máximos y los mínimos establecidos.

La definición que en principio es un poco confusa puede clarificarse en el siguiente ejemplo:

En un listado gráfico -un ambiente- de cadenas de texto -donde todas ellas son elementos AIE-, definimos que como representación de la pregnancia propiedad física a tener en cuenta es el tamaño de la fuente con el que están escritas.

Se establecen un máximo de 20pt y un mínimo de 10pt como límites para esa propiedad. Como resultado tendríamos que los elementos de la lista tendrían tamaños entre 10pt y 20pt atendiendo a su relevancia para el usuario.

El lector se estará preguntado qué pasaría entonces con otros elementos que se encuentran dentro del ambiente que contienen texto pero que no elementos AIE, ¿les afectan las mutaciones? ¿y las restricciones? La respuesta es que no le afectan las restricciones ni las mutaciones de la lista, pero sí las mutaciones o restricciones definidas en el nivel superior. El siguiente gráfico intentará clarificar algo más esta cuestión:

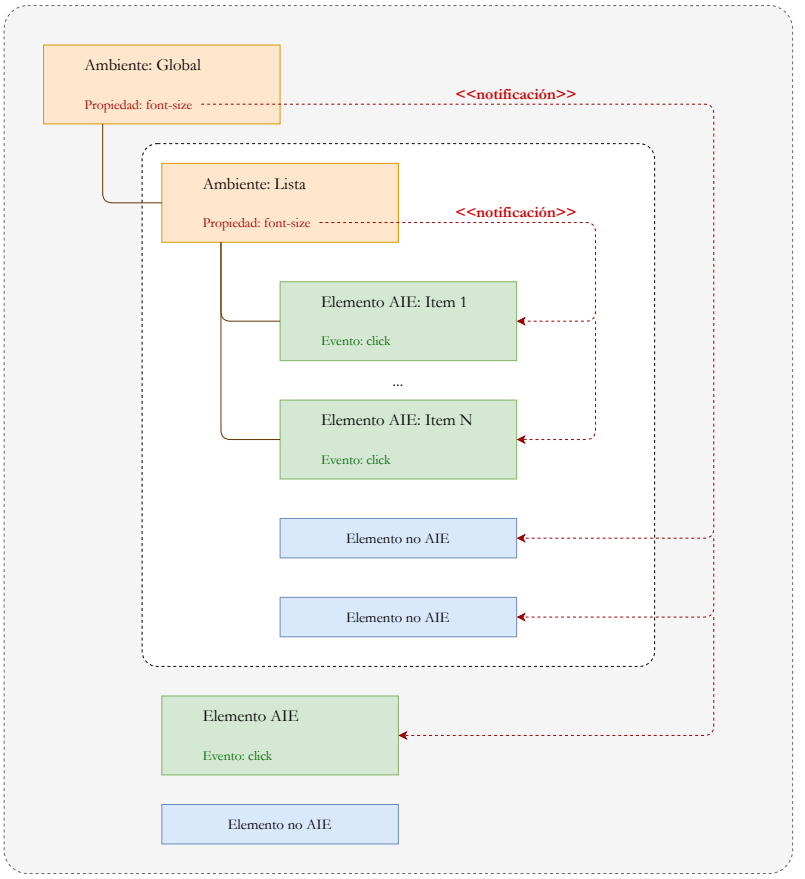


Figura 4.17: Propiedades grupales

4.3.11. Propiedades complejas

Podemos encontrarnos con algunas propiedades cuya modificación sea más compleja al no tener su representación una relación directa con una medida, sino con varias al mismo tiempo.

Un ejemplo de una propiedad simple es el ancho o el alto de un elemento de la interfaz. Ambas propiedades, con independencia de la unidad en la que se se midan en múltiplos de un único valor unitario, ya sea píxeles, centímetros o pulgadas.

Una propiedad compleja es por ejemplo en color, puesto que depende para ser representada de al menos 3 valores, puesto que se forma con la composición del color en términos de la intensidad de los colores primarios de la luz (rojo, azul y verde). Una representación del color, en este caso, expresada en hexadecimal, sería

FFFFAA (amarillo claro).

Las propiedades complejas pueden presentar dos problemas:

- Pueden tener un valor máximo representable, bien determinado por la propia naturaleza de la propiedad (en el caso de la las mejores pantallas actuales podemos representar colores con una profundidad de 32-bits, lo que equivale a una cantidad de colores mayor de 4 billones, un cifra alta pero finita, lo cual indica que no todos los colores de la naturaleza son representables, sino sólo un subconjunto de intervalos) o bien por la capacidades representativas del hardware (en el caso del sonido podemos encontrar dificultades para reproducir ciertas frecuencias graves o agudas al no tener equipos de la calidad suficiente).

- La comparación entre estas propiedades puede no llegar a tener sentido, puesto que habitualmente no las comparamos como tal sino que comparamos características de estas propiedades. Siguiendo con el ejemplo del color, no tiene sentido comparar, por ejemplo, un color con otro con respecto a su valor absoluto, pero sí tiene sentido comparar la saturación de sus componentes o su luminosidad.

Para trabajar con este tipo de propiedades resulta más efectivo considerar su grado de contraste con el entorno, determinado por su grado de saturación o intensidad de sus componentes con respecto al de los componentes cercanos o al promedio de la interfaz.

4.3.12. Actualizaciones del sistema

Una duda que surge al ver un AIE es si este debe reiniciar sus cálculos si un elemento nuevo aparece en el sistema. Por ejemplo, supongamos que en una aplicación de móvil aparece un nuevo elemento asociado a una nueva funcionalidad, ¿que pasaría entonces con ese botón? ¿se incluiría en el sistema con el valor mínimo para su pregnancia? ¿si es una funcionalidad nueva podría considerarse que al usuario le sería de mucho interés conocerla y por tanto debe incluirse en el sistema con un valor máximo para su pregnancia?

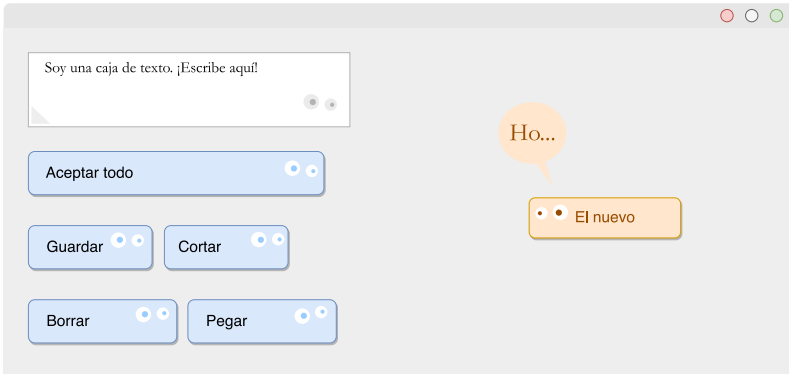


Figura 4.18: Actualizaciones en el sistema

Está claro que la importancia inicial de este nuevo elemento no viene definida ni por l mismo, ni por el sistema, porque el elemento no ha tenido aún contacto con el usuario, es el diseñador de UX quien debe decidir qué interés tiene.

Podemos encontrarnos dos casos diferenciados en lo que refiere a la inclusión de nuevos elementos en el sistema:

- El primer caso es que el elemento sea totalmente nuevo y no tenga relación con el resto de elementos del sistema. Supongamos, ppor ejemplo, un elemento de menú asociado a una nueva funcionalidad que no tiene ninguna relación primaria ni asociada con el resto de funcionalidades del sistema -imaginamos una app para dispositivos móviles de gestión de correo electrónico que permitiera como novedad gestionar el envío postal tradicional y que tuviera como consecuencia un elemento nuevo en el menú principal que diera acceso a esa funcionalidad-, en este caso el diseñador de la UI debe precisar el lugar exacto en el que debe colocar el elemento de menú y su pregnancia inicial.

- El segundo de los casos es que el nuevo elemento esté directamente relacionado con uno de los elementos ya presentes en el sistema y por ello puede asumirse que la pregnancia de ese elemento será similar que la del elemento ya presente.

En ambos casos el diseñador de UX debe decidir qué pregnancia asignarle al nuevo elemento (P_{new}), y esto implica que si el equilibro del sistema debe variar, la

asignación de la pregnancia del nuevo elemento debe ser parte bien de la pregnancia asignada a los recursos no consumidos (*Pu*), o bien de pregnancia del resto de elementos del sistema. Lo cual nos lleva a considerar tres casos (consideramos t+1 como el momento después de hacer la inserción del nuevo elemento con pregnancia P_{new}):

1) Que hubiera suficiente pregnancia no consumida. Entonces:

$$Pu(t+1) = Pu(t) - P_{new} \tag{4.22}$$

2) Que hubiera cierta cantidad de consumida (Pu >0) pero no la suficiente. Entonces:

$$Pu(t+1) = 0 \tag{4.23}$$

y para todo elemento del ambiente en que se insertará en el nuevo elemento (salvo, obviamente, para el nuevo elemento):

$$\Delta P_i(t+1) = -(P_{new} - Pu(t)) \cdot \frac{P_i(t)}{P_{ambient}(t)} \tag{4.24}$$

3) Que toda la pregnancia del sistema estuviera repartida entre sus elementos. Entonces, para todo elemento del ambiente en que se insertará en el nuevo elemento:

$$\Delta P_i(t+1) = -P_{new} \cdot \frac{P_i(t)}{P_{ambient}(t)} \tag{4.25}$$

La asignación directa de una pregnancia puede solucionar esta problemática, pero al hacerse debe considerarse que en el sistema no todos los elementos han sufrido los mismos *ciclos de actualización*, no tienen el mismo tiempo de vida *t.*

El concepto de ciclo de actualización tiene una importancia clave si se desea que el sistema evolucione a diferentes velocidades en función de sus actualizaciones. En el siguiente apartado veremos cómo incluirlo dentro de la función de mutación.

4.3.13. La maduración: el fin de la mutación

Antes de definir este último apartado en la fase de maduración, formalizaremos el concepto de ciclo de actualización:

***Ciclo de actualización**: Número de veces que un elemento o ambiente del sistema sufre una actualización. La memoria de elemento debe llevar contabilizados estos ciclos, que se incrementan cada vez que se produce un evento dentro del sistema que conlleve una mutación de la UI.*

Es importante definir cuándo el ecosistema se ha adaptado al usuario. No es práctico que el ecosistema esté mutando a lo largo de todo su ciclo de vida, ya que en determinado momento se pueden producir patrones de comportamiento que se alejen del comportamiento habitual por parte del usuario y eso lleve al sistema a evolucionar hasta un estado no deseado, sería algo similar al *overfitting* de las redes de neuronas -ver (Schittenkopf, Deco & Brauer, 1997)-. En este caso, aunque sea más difícil llegar el sobreajuste, el procesado interno del AIE conlleva un consumo de recursos que debe eliminarse cuando deje de ser necesario.

Con el concepto de ciclos ya introducido, una función de control puede hacerse cargo de limitar el periodo de mutación, así como de acrecentar o disminuir su efecto. A esta función la denominaremos función de maduración (ϱ).

Un sistema se mantendrá estable mientras haya finalizado la mutación y no hayan aparecido elementos nuevos que puedan provocar al usuario un comportamiento diverso. Por ello, el parámetro que debe recibir esta función no es el número de ciclos del ambiente, sino el número de ciclos desde la última vez que se introdujo un elemento en él (o en su defecto desde el inicio del sistema). Este número de ciclos que será una variable del sistema lo denominaremos k se entiende volverá a cero cada vez que sea introducido un nuevo elemento. Así:

$$P_i(t) = \xi(\tau(v_m, \frac{in_i(t)}{in_{ambient}(t)}, P_{ambient}(t)), P_i(t-1), k, K_{max}) \qquad (4.26)$$

siendo K_{max} el número máximo de ciclos de actualización y

$$\xi(x, y, k, K) = \begin{cases} x & k \leqslant K_{max} \\ y & k > K_{max} \end{cases} \tag{4.27}$$

4.4. Modelando un sistema AIE

La implementación de un AIE, sea cual sea el entorno en el que se aplique, debe dividirse en dos capas: por un lado, el **modelo analítico**, que constará de una serie de componentes que permiten realizar cálculos y recopilar y almacenar la información (recogida de eventos, cálculo de pregnancia, etc.) proporcionada por la capa de conexión, y por otro lado, el **modelo estructural** que permite que los elementos estén conectados a la interfaz (podemos llamarlo la capa de conexión). El modelo con componentes de una interfaz -**Elementos DOM**- son los elementos propios de la plataforma sobre la que trabaja el AIE (botones de una aplicación de escritorio de windows, elementos DOM de un navegador, cajas de entrada de una aplicación Android, etc.).

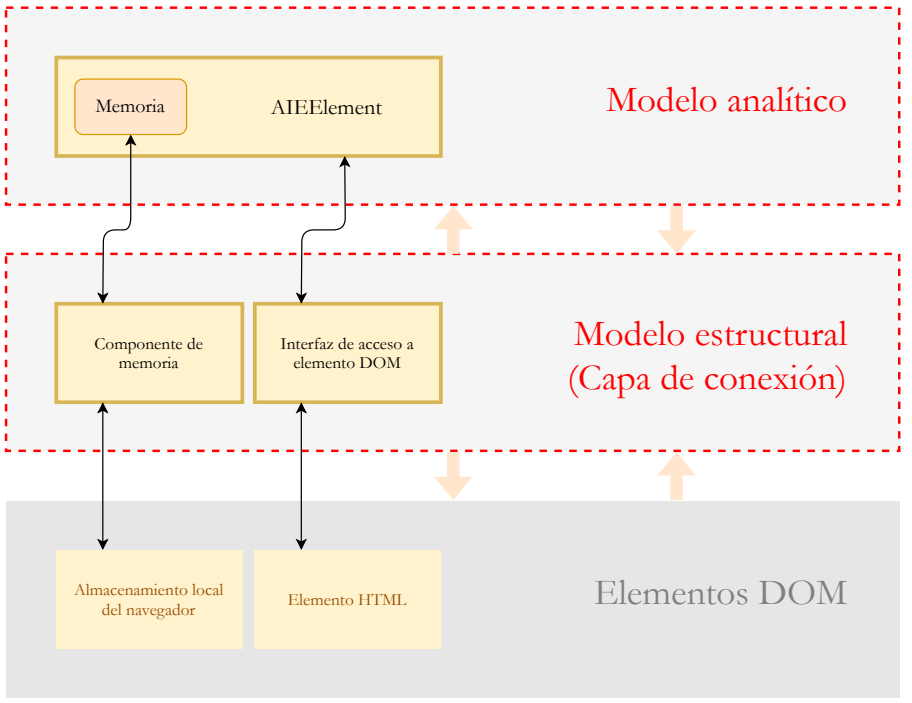


Figura 4.19: Ejemplo de modelado un sistema AIE para una aplicación web

El modelo analítico de la AIE, estará formado por los componentes esenciales que se han definido en los capítulos anteriores, es decir los *Ambientes*, formados a su vez por *Elementos* del ambiente.

Para dividir las tareas que ha de realizar el sistema, consideramos que cada elemento tiene asociado un (i) **Calculador de pregnancia**, un (ii) componente ocupado de gestionar los datos que puede guardar (**Memoria**) y un (iii) **Procesador de eventos** (que se comunicará con el sistema de eventos de la interfaz para capturar los eventos que considere oportunos).

4.4.1. El modelo analítico

La implementación del modelo analítico puede cambiar dependiendo de los mecanismos provistos por el lenguaje utilizado, pero su funcionalidad debe ser la misma para cualquier entorno. Sin embargo, la capa de conexión variará en gran medida según el entorno para el que se decida adaptar un AIE.

En la etapa de desarrollo se configurará el entorno: de definirá cuál es la fun-

ción de maduración (τ) y la velocidad de maduración(v_m) y el número máximo de ciclos de actualización (K_{max}).

También antes de poner en funcionamiento el sistema es muy importante definir qué eventos resultan importantes y cuáles son las propiedades físicas a las que afecta la pregnancia de cada componente.

4.4.1.1. El calculador de pregnancia

El **Calculador de pregnancia** será responsable de implementar el modelo matemático que representa las funciones de mutación y maduración. Para ello, utilizará las constantes v_m y K_{max} definidas en el entorno.

Haciendo un análisis de cómo representar un AIE, partimos que este tiene dos tipos de componentes: Ambientes y Elementos. La única diferencia entre ambos es su composición, puesto que un Ambiente puede estar compuesto de cero o más Ambientes o de cero o más Elementos, con la condición de estar compuesto de al menos un componente. El Elemento sin embargo no está compuesto por nada.

4.4.1.2. El procesador de eventos

En cuanto a la recogida de eventos, un Ambiente puede recoger eventos de su entorno de forma independiente a la de sus componentes, de igual forma que las características físicas a las que afecta su pregnancia no tienen porqué ser las mismas ni una suma de sus componentes.

Esto nos lleva a que Ambientes y Elementos son en realidad el mismo tipo de componente, al que denominaremos en su modelado *AIEElement* y que puede estar compuesto de cero o más elementos similares a él.

Por otro lado, este componente *AIEElement* tendrá asociadas cero o más propiedades a las que afecta la pregnancia, pero estas propiedades son una representación abstracta de las propiedades físicas de los elementos, y será necesario que un elemento que comunique sus cambios de estado a los elementos de la interfaz (la capa de conexión es responsable de ello).

Se ha realizado un diagrama de las clases de UML para el modelo analítico (independiente del entorno), del cual se deriva una posible implementación, que se muestra a continuación, donde se modelan estos elementos y las relaciones entre

ellos. En el diagrama, hemos compuesto una estructura de clase donde el componente *AIE* representa el entorno de nivel superior, *AIEMemory* es el componente asociado con la gestión de la memoria, *AIEPregnancyCalculator* realizaría las funciones de cálculo de la pregnancia y *AIEventProcesor* sería responsable de gestionar el acciones resultantes de los usuarios sobre los componentes físicos.

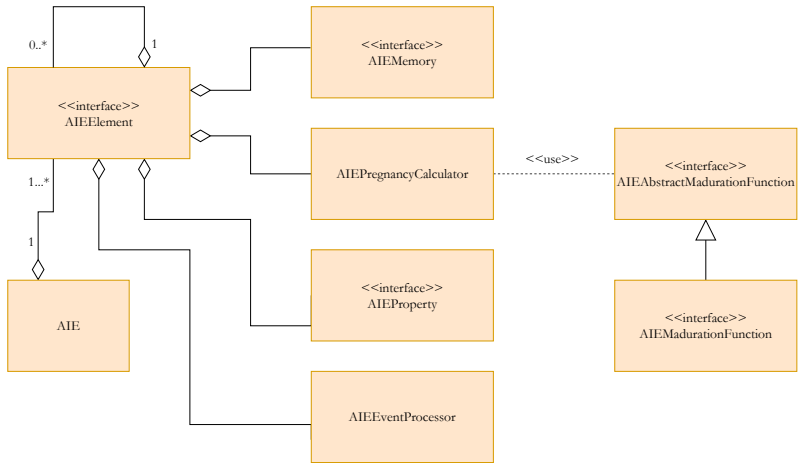


Figura 4.20: UML. Modelo analítico

4.4.2. El modelo estructural

Al adaptar un AIE a cualquier interfaz, debemos tener en cuenta que debe existir un mecanismo de comunicación entre los componente AIE y el DOM (*Document Object Model*) -DOM es un concepto que se utiliza a menudo en programación web, pero que poco a poco se ha ido incorporando en distintos entornos para hacer referencia al modelo con componentes de una interfaz-.

Como ya hemos comentado no hace falta que todos los objetos de la interfaz tengan asociado un elemento AIE (por ejemplo, un pie de página o una notificación puede que deban permanecer en la web y su apariencia no deberían verse afectada por motivos como una restricción legal) pero los que sí lo tengan asociado deben exponer una API que permita realizar modificaciones en su apariencia.

Los dos puntos críticos que debe manejar el sistema estructural son: cómo abordar las representaciones y cómo almacenar los datos de los cálculos. Para ello

debe haber tanto componentes que puedan trabajar con el sistema de almacenamiento de cada dispositivo (es decir un componente de *Memoria* adaptado), como elementos que permitan manejar la complejidad de las dimensiones de representación (*Propiedades*) traduciendo las variaciones a las medidas asociadas correspondientes a cada dimensión.

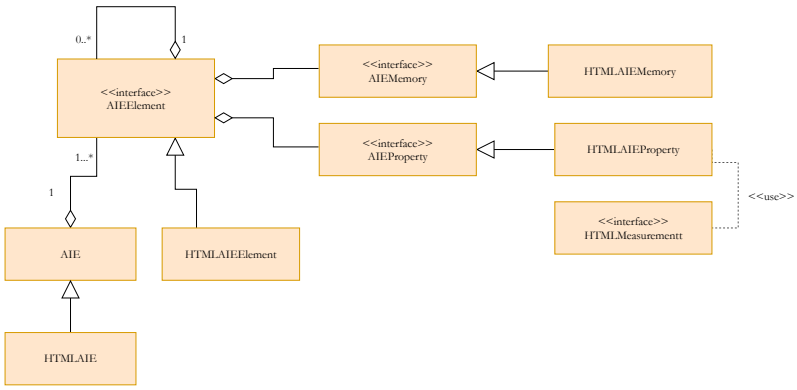


Figura 4.21: UML. Modelo estructural (parcial)

A circle of life

5.1. Los Ecosistemas de Interfaz Adaptativos en los ciclos de vida clásicos

Lo más habitual es que Ecosistemas de Interfaz Adaptativos sean incluidos en un proyecto en la fase de mantenimiento, puesto que normalmente las aplicaciones que más necesitan este tipo de soluciones son aplicaciones pequeñas con pocos recursos, cuyo mantenimiento es muy básico y han seguido ciclos de desarrollo en cascada.

Para aplicaciones medianas y grandes en cuanto a recursos y financiación, la inserción de los AIE no dejaría de ser una *feature* más en el ciclo de desarrollo. Las empresas que desarrollan este tipo de aplicaciones suelen llevar ciclos de vida iterativos. Se planificará por tanto como una nueva necesidad, se añadiría al pliego de requisitos y seguiría el flujo normal de desarrollo de software como la integración de cualquier otra funcionalidad.

El ciclo de vida iterativo es posiblemente el más utilizado en estos días, y unido a las metodologías ágiles, conforman el *sancta sanctorum* del desarrollo en la mediana/gran empresa.

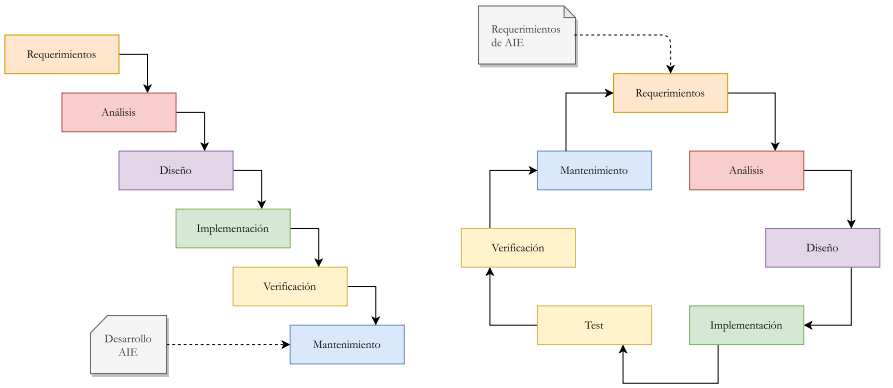


Figura 5.1: Los AIE en los ciclos de vida clásicos

Dentro de las metodologías ágiles, el desarrollo orientado a funcionalidades (FDD, *Feature-Driven Development*) es posiblemente el más utilizado, ya sea en conjunto con SCRUM -ver (Schwaber, 1997)-, X-treme -ver (Fraser y col., 2003)- o KanBam -ver (Matharu, Mishra, Singh & Upadhyay, 2015)-. La idea de todas ellas es acortar los ciclos de desarrollo a pocos días (15 habitualmente) y centrarse en pequeños desarrollos que pasan por el ciclo completo de vida. Para ello en cada ciclo (un desarrollo completo consta de varios) se centra toda la atención en unas pocas funcionalidades, a estas funcionalidades se las denomina *features*.

Otras metodologías ágiles se centran en la adaptación y desarrollo de proyectos que tienen un crecimiento rápido, como *Adaptive Software Development* (ASD) -ver (Highsmith & Cockburn, 2001)-. Estas metodologías se realizan más iteraciones cortas, solventando problemas muy pequeños en cada iteración.

5.1.1. Un pequeño vistazo atrás: el desarrollo orientado a features y los flags

El desarrollo orientado a *features* no es precisamente una idea nueva, la primera vez que se plantea es en 1997. El diseño final de este desarrollo partió del método Coad, elaborado por Peter Coad, Eric Lefebvre y Jeff De Luca, que puede encontrarse en (Palmer & Felsing, 2001).

El FDD está basado en la división del desarrollo en dos etapas, una para la identificación de características y otra para la implementación de cada una de ellas de forma individual. Todo se basa en la división del trabajo en paquetes, validados por

desarrolladores y clientes. Cada paquete de trabajo es un grupo de características que están relacionadas entre ellas que se completan por un método iterativo.

En 2007 es propuesto el MDD (*Model Driven Development*), -ver (Trujillo, Batory & Diaz, 2007)-, un paradigma orientado a la creación de software partiendo del modelado y transformación de sus datos. En realidad el MDD da con una solución más que válida para la creación de software al inicio de siglo, un software muy ligado al tratamiento y transformación de datos, continuando así el trabajo de De Luca, Coad y Lefebvre.

En 2009 Sven Aplen y Christian Kästner proponían un modelo de desarrollo software también basado en *features* -véase (Kästner & Apel, 2013)-, como una formalización de la teoría de varios autores -véanse los estudios (Schermann, 2017), (Czarnecki, 2002), (Aleixo, Freire, Alencar, Campos & Kulesza, 2012), (Don Batory, 2005) y (Apel, Lengauer, Möller & Kästner, 2008)-. Su idea principal es que en el modelo de desarrollo orientado a *features* se definiera una línea de productos software cuyas partes estuvieran interconectadas a través de un canal de comunicación estandarizado pero que fueran independientes entre sí, lo que suponía una propuesta muy cercana a la definición aportada por (Fabijan, Olsson & Bosch, 2016), que la define así:

"In software development, a feature is a component of additional functionality, in addition to the core software. Typically, features are added incrementally, at various stages of the life cycle, usually by different developer groups"

Por otro lado, con este tipo de desarrollo la *feature* se convierte en un elemento determinante a la hora de organizar el código, pues implica un gran esfuerzo para la obtención de un diseño e implementación especialmente modulares. En este sentido Prohofer -ver (Prehofer, 1997)-, ya apuntaba a que la implementación de una funcionalidad debe quedar patente y bien definida en el código fuente.

Algunos lenguajes como Jak (un superconjunto extensible de Java que admite la meta-programación) -ver (Don Batory, 2006)-, proveen de utilidades para hacer posible una implementación orientada a *features*, como por ejemplo la creación de jerarquías de contención -ver (D. Batory, Sarvela & Rauschmayer, 2003).

Una de las características más importantes de un desarrollo orientado a *features* es la comunicación entre las distintas capas del sistema. Las interacciones entre las capas requieren que las *features* se adapten cuando estas se componen a su vez

de otras *features*. Para ello se definieron paradigmas computacionales como GenVoca -ver (D. Batory y col., 2003)-, AHEAD -ver (D. Batory, Sarvela & Rauschmayer, 2004)-, o más recientemente un Sistema de Diseño *Feature-Oriented Model-Driven* como FOMDD -ver (Cavarlé, Plantec, Costiou & Ribaud, 2016).

Como sustento para el análisis de sistemas MDD se utiliza FOP (*Feature Oriented Programing*), que es un paradigma para crear líneas de productos de software que establece un modelo matemático donde la unidad básica es una *feature* sobre la que operar con un conjunto de operaciones de álgebra lineal. Por ejemplo en FOP, $M=\{ i,j,k\}$ representa un programa M que está compuesto por *i*, *j* y *k* *features*.

Muy cercano a la FOSD se encuentra la CBSE (*Component-based software engineering*), una metodología de construcción de sistemas de software utilizando componentes independientes -en (Trujillo y col., 2007)-. Si consideramos que un componente desarrolla una y solo una característica software, entonces FOSD y CBSE son muy similares, aunque no iguales, puesto que siempre existirá la diferencia entre ellos de que un componente en un sistema CBSE funciona como una caja negra, es decir, únicamente debe tener una interfaz bien definida para poder comunicarse con él, lo que conlleva que su nivel de encapsulamiento es mayor: no puede tener acceso libre a otras partes del sistema, en FOSD los componentes sin embargo son más abiertos y puede llegar existir cierto nivel de acoplamiento.

FOSD y CBSE son soluciones muy válidas, pero si bien el uso de componentes facilita el desarrollo y el despliegue independiente, dificulta la encapsulación de *features* transversales de un sistema de software. Para solucionar el tema de la transversalidad se han venido utilizando durante los últimos años los *flags*, como una técnica de desarrollo habitual en pequeñas y grandes aplicaciones con miles de usuarios (generalmente aplicaciones móviles y web). Más allá de la transversalidad, el uso de *flags* reporta grandes beneficios al desarrollo, pero tiene también puntos negativos como su mantenimiento y la deuda técnica que genera.

Ahora que han quedado claras las bases de las metodologías más extendidas, la pregunta clave es ¿pueden los AIE ayudar a solucionar los problemas de mantenimiento y deuda técnica? La respuesta es sí. Pero para ello habrá que hacer un planteamiento más amplio y considerar los *flags* como la base de una nueva arquitectura y metodología: a *feature flags-Driven Development (FFDD)*.

5.2. Feature Flag-Driven Development

5.2.1. Usando flags

Comencemos clarificando conceptos, veamos que es un *flag* y cómo afecta al desarrollo. Podemos definir los flags desde dos puntos de vista, como usuario o como desarrollador. Desde el punto de vista del desarrollador es un mecanismo para controlar el ciclo de vida completo de una *feature*, permitiendo gestionar componentes y compartimentar riesgos. Puede también utilizarse para hacer test A/B -ver (Lennarz, 2017)- activando ciertas funcionalidades para unos usuarios o discriminando a otros, o hacer despliegues progresivos en función de los riesgos que conlleva la distribución de nuevas funcionalidades. En este último caso de uso, que se puede ver frecuentemente en aplicaciones web y móviles, se requiere que las aplicaciones tengan acceso a la red, porque será desde el servidor de estas desde donde se realice la configuración que determine qué *flags* estén activos y cuáles no.

Para Peter Hodgson, ver (Schermann, 2017) y (Hodgson, 2016), existe una pequeña variedad de utilizaciones de flags; atendiendo a su duración en el tiempo y su dinamismo se consideran: los *feature-flags*, los *ops-flags*, los *permission-flags* y los *experiment-flags* (que normalmente se asignan a funcionalidades experimentales). Los *feature-flags* son utilizados en desarrollo para poder integrar las *features* en desarrollo, en proyectos basados en *trunk development* -ver (Sole, 2018) y (Hammant, 2013)-, dentro de una rama de integración compartida. Los *experiment-flags* son *flags* de un tiempo de vida muy corto, utilizados para realizar test sobre el comportamiento del usuario, normalmente test A/B -ver (Vitharana & Jain, 2000)-. Los *ops-flags* son flags de un tiempo de vida muy variable que se aplican generalmente sobre el comportamiento del sistema. Desde el punto de vista del usuario, los flags determinan qué características de la aplicación desea él que estén activas y cuáles no. Los flags utilizados en este caso son los que Hodgson denomina *permission-flags*. El uso de estos puede llegar a ser más variado, porque pueden ser también utilizados para otorgarle al usuario permisos de utilización de ciertas funcionalidades de forma extraordinaria desde el área de negocio que gobierne una aplicación (por ejemplo convirtiendo a un usuario en «usuario premium»).

El usuario normalmente dispone de un panel de configuración con un listado

de *checkbox* que activa o desactiva a su antojo (depende de sus propios requisitos en cuanto a espacio o asignación de recursos) y que puede implicar la descarga de paquetes adicionales.

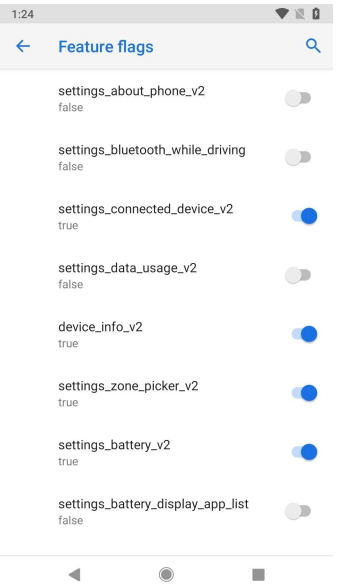


Figura 5.2: Menú de feature-flags en la configuración de desarrollador de Android 9 Pie

Puede considerarse también una tercera vía: que los *flags* se activen o desactiven de forma autónoma en función de comportamiento del usuario, es decir, a través de una evaluación del uso de la aplicación. De esta forma el usuario no se tendría que preocupar de la gestión de sus recursos. A estos *flags* los podemos denominar *ambient-flags*. Si representamos los distintos *flags* en un eje temporal, de igual forma que (Hodgson, 2016), obtendremos el siguiente resultado:

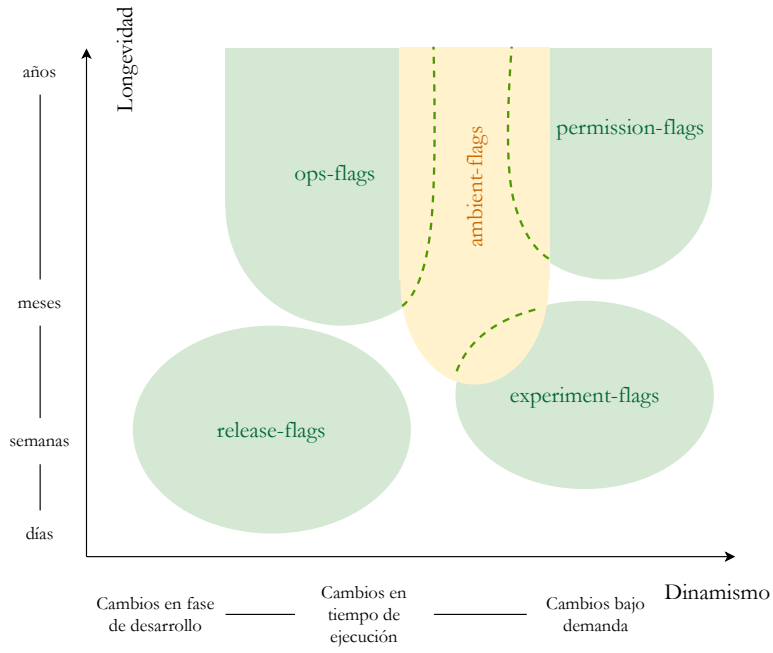


Figura 5.3: Grupos de flags en función de su dinamismo y longevidad

5.2.1.1. Beneficios del uso de flags

La utilización de *flags* dentro de un sistema grande reporta muchos beneficios tanto desde el punto de vista de desarrollo como desde el de usuario.

Para el usuario:

- Permite un mayor control de la aplicación, ayuda a comprender mejor su funcionamiento, a adecuar el uso de ésta a sus recursos y por ende, a mejorar el rendimiento de los dispositivos.
- Aumenta el grado de personalización y adaptación a cada tipo de usuario.

Para el desarrollador:

- **Reducción de riesgos e invisibilidad**. El uso de *flags* permite realizar pruebas en entornos de producción que sean invisibles para el usuario de tal forma que las versiones finales lleguen más estables.

- **Facilidad de integración**. La velocidad de integración de versiones beta dentro del producto final hace que la integración de la versión final de la *feature* no sea un proceso crítico.

- **Flexibilidad**. Una vez implementado el sistema de comunicación el desarrollo de nuevas *features* se vuelve más sencillo y el sistema puede expandirse más rápido, tanto horizontal como verticalmente.

- **Depuración**. Mejora las labores de reparación, aislando el código y evitando dependencias -ver (Jones, 2012)-.

5.2.1.2. . . . y daños colaterales

La utilización de *flags*, genera también una serie de perjuicios que requieren de la atención del desarrollador:

- **Deuda técnica**. De acuerdo con (Rahman, Querel, Rigby & Adams, 2016), el mantenimiento de los *flags* genera automáticamente una deuda técnica. A excepción de los *permission-flags* y los *ambient-flags*, el resto de *flags* requieren de un mantenimiento por parte del desarrollador: deben eliminarse en el momento que no se estén usando.

- **Seguridad**. El mantenimiento de un sistema de almacenamiento donde se consulten los *flags* supone también un problema de seguridad. Si los *flags* pueden activarse manipulando el código fuente de la aplicación, utilizando por ejemplo ingeniería inversa, entonces el almacenamiento de datos requerirá de un sistema de seguridad adicional para evitar conflictos.

5.2.2. Formalizaciones

El salto desde el FDD al FOSD y finalmente al FFDD (*Feature Flag Develop-Driven*) se basa en que la concepción de que la estructura software no es un ente estático que se actualiza solamente con cada versión del sistema, sino que es completamente dinámico y puede ser activado bien por el equipo de producción del software o bien por el usuario.

En una arquitectura FFDD el sistema software estaría constituido por una *feature* base, que sería la encargada de controlar el inicio del sistema y manejar el sistema de control que maneje la activación y desactivación de *features* (aunque esto puede ser considerado también como una *feature*).

Por otro lado la estructura transversal del código creada para los FFDD debe operar también en el flujo de datos del sistema, asegurando que cada *feature* respete un sistema de permisos para no invadir el espacio de datos ni de funcionalidad perteneciente a otra *feature*.

También debe existir un mecanismo dentro del FFSS que determine la compatibilidad o incompatibilidad de *features* coetáneas, y como ya apunta en el FOP, de un sistema de resolución de dependencias -ver (Breivold, Crnkovic, Land & Larsson, 2008).

Si representamos esta arquitectura en el álgebra para FST (*Feature Structure Tree*), utilizada en FOP, obtenemos que:

Si

$$z_i = x \cdot y, z_j = m \cdot \neg y \tag{5.1}$$

entonces,

$$App = \{z_1 \cdot z_2 \cdot ... \cdot (z_i \vee z_j) ... \cdot z_n\} \tag{5.2}$$

donde *x, y, z* serían features, *App* una aplicación y *n* el número de *features*.

En la actualidad el mecanismo de control de dependencias lo podemos encontrar ya integrado en grandes sistemas software (como por ejemplo los sistemas operativos), donde ayuda a manejar la complejidad en instalación de paquetes y bibliotecas con relativa facilidad.

Sin embargo, el proceso de control para la disyunción de *features* se implementa muy poco (no confundir con el control de dependencias en el versionado), puesto que para ello cada *feature* debe exponer su funcionalidad, los requisitos del sistema que utiliza y cuáles de ellos no puede compartir y debe utilizar de forma

exclusiva; el sistema por su parte debe indicar a las *features* de qué recursos dispone y cuál es su grado de disponibilidad (es decir el número de *features* que tienen acceso a ello).

Podría ampliarse con esto el álgebra de definición de FOP, incluyendo los recursos y definiendo todos los elementos como funciones que actúan sobre los recursos:

$$App(r_1...r_k) = \{z_1 \cdot z_2 \cdot ... \cdot (z_i \vee z_j)... \cdot z_n\}(r_1...r_k) \tag{5.3}$$

Donde *r* son los recursos del sistema.

Como es de esperar, puesto que estamos hablando de *feature-flags*, las *features* a pesar es encontrarse en el sistema no estarán siempre consumiendo recursos, dependen de los *flag* activos, esto se puede definir mediante una función que lo compruebe *A*, tal que

A(z, f_Z) = { }, si el flag de z (f_Z) no está activo

A(z, f_Z) = z, si el flag de z (f_Z) está activo

y por tanto:

$$\begin{aligned} App(r_1...r_k, f_{z1}...f_{zk}) = &\{A(z_1, f_{z1}) \cdot A(z_2, f_{z2}) \cdot ... \\ &\cdot (A(z_i, f_{zi}) \vee A(z_j, f_{zj}))... \\ &\cdot A(z_n, f_{zn})\}(r_1...r_k) \end{aligned} \tag{5.4}$$

Añadiendo la *feature* base del sistema (z_{base}) y el control de activación ($z_{control}$) tendríamos algo tal que así:

$$\begin{aligned} App(r_1...r_k, f_{z1}...f_{zk}) = &\{z_{base}, z_{control}, A(z_1, f_{z1}) \cdot A(z_2, f_{z2}) \cdot ... \\ &\cdot (A(z_i, f_{zi}) \vee A(z_j, f_{zj}))... \\ &\cdot A(z_n, f_{zn})\}(r_1...r_k) \end{aligned} \tag{5.5}$$

En muchos casos (como en el del Android 9 Pie de Fig. 5.2) el consumo de recursos y la disyunción entre *features* no son elementos importantes o no implican una degradación crítica del sistema puesto que las funcionalidades a las que se aplican los *flags* no implican un uso excesivo de recursos ni son excluyentes entre sí.

El proceso de desarrollo FFDD es por tanto incremental y está basado, como el FOSD y el CBSE en la creación de pequeñas funcionalidades que deben suscribirse a un sistema control que se haga cargo de mantenerlas o no activarlas de acuerdo a los requisitos de usuario, a la degradación del sistema o a un control remoto impuesto desde el modelo de negocio de la aplicación.

5.2.3. Una arquitectura para feature-flags

Para la implementación de un FFDD, de acuerdo con lo visto en el punto 3, la aplicación está constituida al menos por los siguientes módulos:

- Flag storage. Repositorio de almacenamiento con los valores de los *flags* y la meta-información que estos requieran para la activación o desactivación de *features*. Entre los datos extra que puede contener sobre la activación de *flags* están los siguientes:
    - Duración de la activación.
    - Fecha de la activación.
    - Intervalos de activación.
    - Bloqueo (para impedir su cambio de estado).
    - Permisos de modificación (para el sistema, el usuario, el área de negocio, etc.)
- Base Feature. Será la encargada de realizar el inicio de la aplicación, capturar los recursos disponibles y entablar un diálogo con el sistema operativo para obtener los permisos que se requieran.
- Control feature. Será la encargada de gestionar la carga y activación de las *features*. Mantendrá el sistema de control de versiones y dependencias. Consultará el *flag storage* para saber qué *features* activar y establecerá un canal

con la *Base Feature* con el que controlar el estado del uso de recursos. Será habitual que algunas *features* estén activadas siempre desde el primer momento por considerarse indispensables para el uso interno de la aplicación (funcionalidad básica).

- Features. Cada *feature* deberá mantener información sobre sus dependencias, recursos y servicios actualizados y proporcionarlos a la *Base Feature* previamente a su activación.

- User flag control feature (opcional). Si el usuario tuviera posibilidad de habilitar los *flags* por su cuenta, la *feature* podría modificar el valor de aquellos *flags* del *Flag storage* que tuvieran permitido ser modificados por el usuario. La activación de los *flags* por parte del usuario puede no implicar la activación de una *feature*. El encargado de decidir en última instancia si una *feature* se activa o no es *Control feature*.

- Business flag control feature (opcional). La funcionalidad de esta *feature* es la misma que *User flag control feature*, pero a nivel de negocio que controla la aplicación. Estará conectada por la red al control de negocio desde donde le llegarán las órdenes. Será a través de la cual se activarán por ejemplo los test A/B o se dará acceso a nuevas *features* en procesos complejos de despliegue o actualización.

- Feature communication channel. Canal de comunicación que pone en contacto a las *features*. Por motivos de seguridad pueden existir uno o varios canales de comunicación dentro de la App.

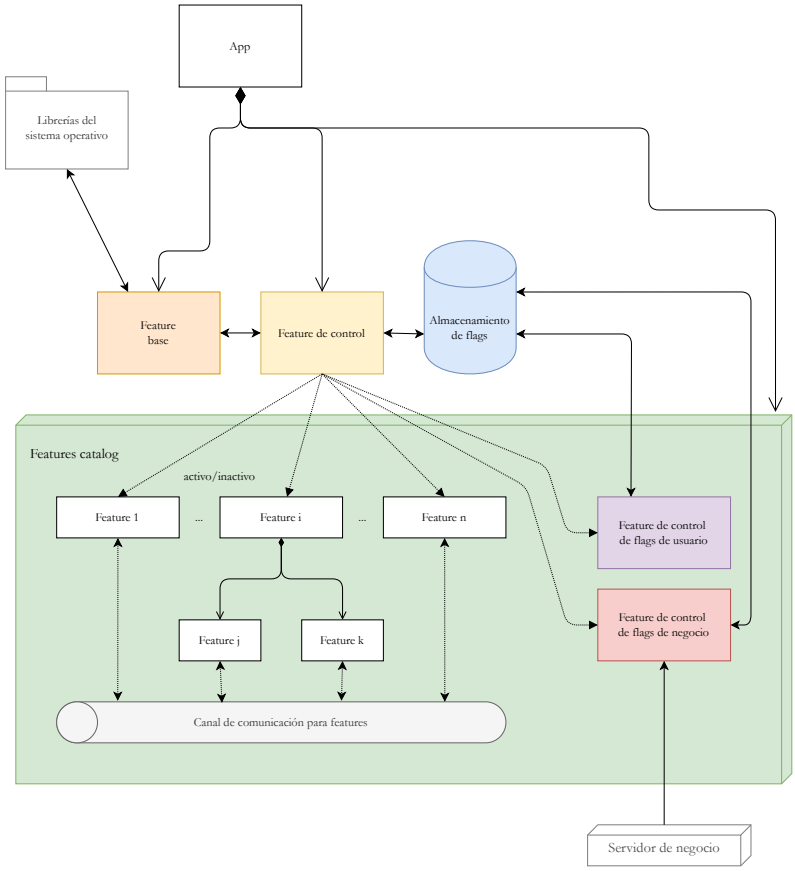


Figura 5.4: Modelado de un sistema FFDD

Existe la posibilidad de que el sistema sea muy grande y podamos tener un repositorio de *features* donde se encuentren todas las *features* del sistema (lo cual sería extremadamente útil para aplicaciones multisistema y multidispositivo con versionados distintos, como por ejemplo las app de teléfonos móviles). Si es así, es posible que la *feature* de control que tenga un poco más de trabajo. Además de revisar las dependencias, sería también la encargada de instalar y desinstalar las *features* desde el repositorio. Como es una tarea muy grande para una sola *feature*, podemos considerar *Control Feature* como un agregado de dos features distintas (ver imagen): una encargada del control de dependencias y la otra encargada de la instalación y un recolector de basura para limpiar el sistema de aquello que no se utiliza.

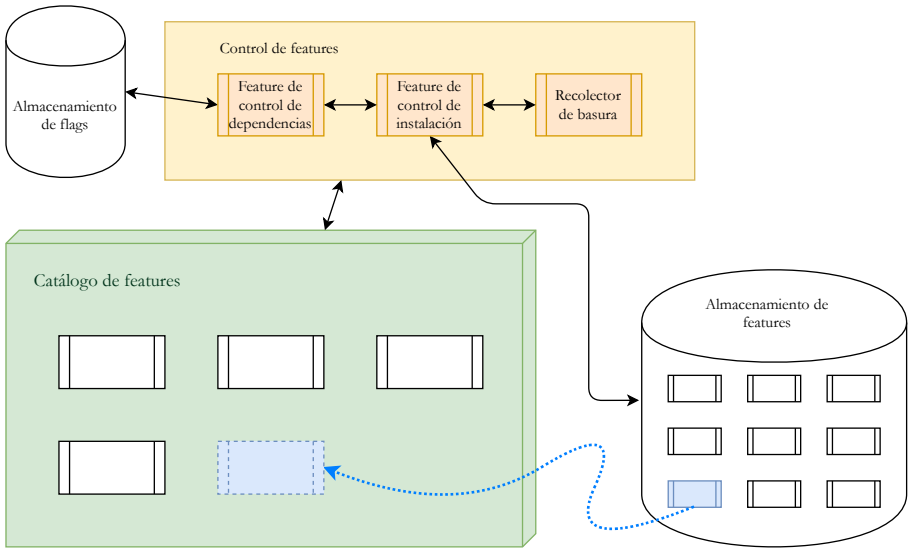


Figura 5.5: Arquitectura de Control Feature

La activación y desactivación de *flags* provee grandes posibilidades a la hora de ofrecer al usuario un mejor aprovechamiento de los recursos a nivel de hardware y una mejor experiencia dando la posibilidad de tener acceso a *features* experimentales, añadiendo o quitando características en función de sus preferencias, o mejorando las interfaces y procesos internos a través de test A/B.

Como ya se adelantaba en la sección "¿Qué es un *flag*?", los *flags* también pueden ser activados y desactivados a partir del comportamiento del usuario, es decir si un usuario no utiliza durante mucho tiempo una funcionalidad de un sistema, podrá esperar que el peso de esa funcionalidad en la interfaz disminuya poco a poco, y si finalmente no la utiliza en absoluto esperará a que esta se desactive hasta que él mismo desee reactivarla. Por el contrario si la funcionalidad gana peso en cuanto a utilización también es asumible que esté siempre activa y además incremente su presencia en la interfaz. La no utilización de funcionalidades propias de una aplicación no es algo extraño, haciéndonos eco del estudio del Pew Research Center sobre el uso de twitter (Barthel & Shearer, 2015) y de (N. Y. Lee, Kim & Sang, 2017) encontramos que el 28 % de los usuarios de Twitter (varios millones) no han enviado nunca un tweet, es decir, para ellos el botón de escribir tweet (y su funcionalidad asociada, lo que implica a toda la *feature* de publicación) no les importa, porque utilizan la red social solamente para leer y/o hacer retweets de no-

ticias. ¿Qué sentido tiene por tanto para ellos mantener ese botón en la pantalla?

Dentro de un sistema FFDD, el módulo de AIE debe ser introducido como una *feature* más y requiere de una pequeña adaptación por parte del resto de *features*, puesto que debe nutrirse de la información generada por éstas de forma constante para poder realizar los cálculos de la pregnancia.

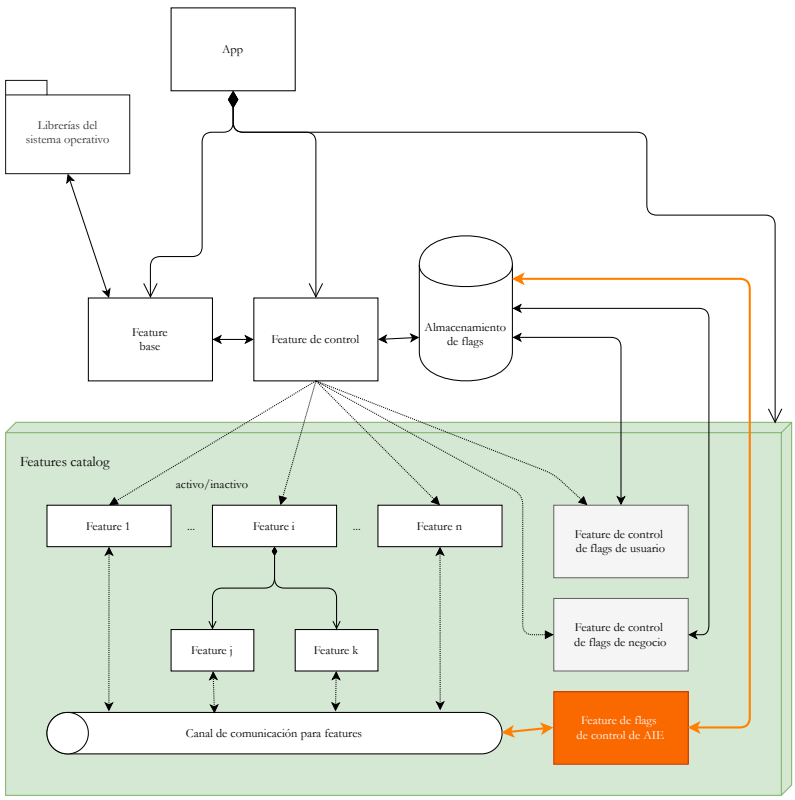


Figura 5.6: Modelado de un sistema FFDD con AIE

Puesto que el AIE necesita tener información de cómo interactúa el usuario y cómo disponer de competencias para modificar la UI, ha de tener contacto de forma directa o indirecta con la UI. Así pues nos podemos encontrar dos casos:

- En sistemas pequeños es posible que haya una única *feature* que pueda hacerse cargo de gestionar la GUI, y por tanto bastaría adaptar esta *feature*. Además es posible abrir un canal de comunicación exclusivo entre esta *feature* y la *feature* AIE.

- En sistemas más grandes el funcionamiento se vuelve más complejo, aunque en esencia respondería al mismo patrón. Cada *feature* que tuviera interfaz gráfica debería estar compuesta al menos de dos *features*: una encargada de la funcionalidad y otra encargada del UI, de tal forma que fuera esta última la que proveyera de datos al AIE y tuviera competencias para modificar la UI de acuerdo a las órdenes indicadas por el AIE.

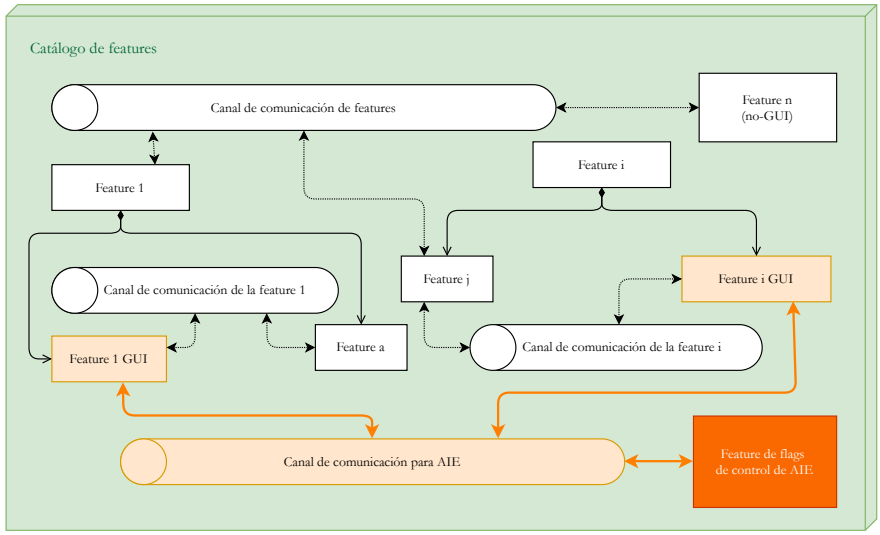


Figura 5.7: Conexión de features y AIE

Las modificaciones que han de realizarse en las *features* de interfaz para poder adaptar el AIE, tienen que ver con el marcado y definición de elementos de la interfaz, así como de los eventos que se producen:

- En primera instancia, la *feature* debe comunicar al AIE la estructura inicial de la interfaz convenientemente etiquetada con la nomenclatura AIE. Se han de añadir marcas en la definición de los elementos, enviar por medio del canal de comunicación todos los eventos que resulten significativos.

- Cada evento de interfaz que se produzca sobre los elementos marcados debe ser comunicado al AIE mediante el canal de comunicación interno.

- En el caso de no tener permisos del AIE para modificar la interfaz, será cada *feature* la que asuma esa competencia, interpretando los mandatos que realice el AIE a través del canal.

El caso más drástico que puede darse es la supresión completa de una *feature*. Este caso debe ser tratado bajo ciertos niveles de seguridad de cara a que el usuario perciba que esa *feature* está desactivada por falta de uso, y que sepa en todo momento donde pueda reactivarla si se fuera necesario. En caso más común de *flag* utilizado en un sistema AIE pertenecería a la intersección de los grupos *ambient-flag* y *permission-flag*. Para determinar si una *feature* debe o no desactivarse, al AIE le basta con comprobar si en su árbol de componentes el nodo asignado a dicha *feature* tiene una pregnancia igual a 0.

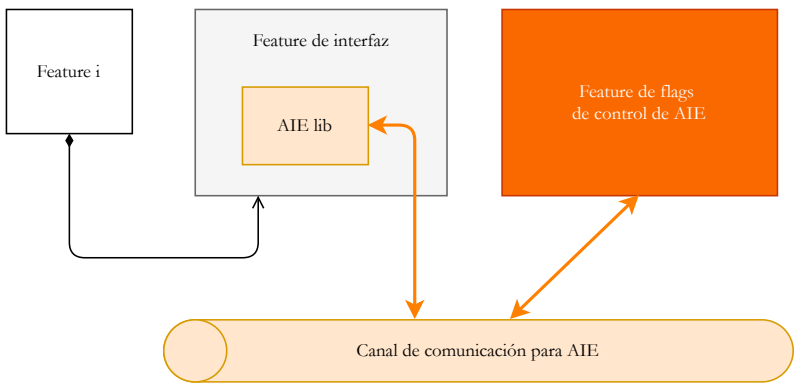


Figura 5.8: Modelado de la conexión entre una Feature GUI y la feature AIE

Otra opción menos drástica de cara al usuario, pero que también implica la desactivación de la *feature*, es su reducción dentro de la interfaz de tal forma que tenga un impacto mínimo. Para ello cada *feature* debe estar compuesta necesariamente por al menos otras dos: una con la funcionalidad completa y otra que solo contenga una mínima representación en la interfaz y cuya funcionalidad fuera similar a la de una *feature* de control de *flag* del usuario (ver apartado 3). Esta última feature, llamémosla *feature* fantasma, tendrá como única funcionalidad activar en el repositorio de *flags* un conmutador que indique cuál de las dos *features* está activa, si ella misma o la feature original con toda la funcionalidad.

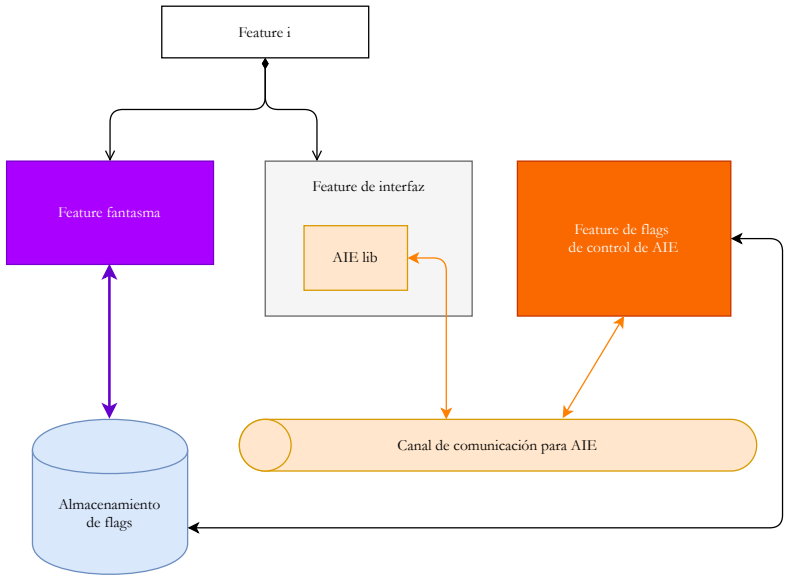


Figura 5.9: La feature fantasma

Tanto si los Ecosistemas de Interfaz Adaptativos llegan a una aplicación en la etapa de mantenimiento de un ciclo clásico como si se van a añadir como una *feature*, previamente a su puesta en funcionamiento es lógico plantearse ciertas cuestiones como:

- ¿Cuál debe ser el número de ciclos de actualización (K_{max})?
- ¿Puede llegarse a configurar un producto mejor para los usuarios desde un entorno de pre-producción?
- Si todo va mal, ¿cómo hacer rollback?

En los siguientes apartados vamos a tratar de solucionar estas dudas, viendo también cómo crear una arquitectura que nos permita recoger información de varios usuarios de pruebas y fusionar sus datos para conseguir que la primera versión del producto que salga a producción se encuentre medianamente adaptada.

5.3. Selección de los ciclos de actualización

Resulta un poco difícil predecir sin ningún tipo de análisis de datos previo cuál va a ser el momento en el que el usuario ha realizado un número suficiente de

interacciones para determinar cuáles son sus costumbres o actividades recurrentes en una aplicación. Y mucho más difícil hacer esta predicción extendida a cualquier tipo de aplicación.

La solución más óptima pasa por recurrir a la estadística a la hora de predecir el momento de estabilidad de un sistema. Podemos convertir por tanto un valor variable K_{max} en una función dependiente de la variabilidad de la pregnancia del sistema, es decir:

$$P_i(t) = \xi(\tau(v_m, \frac{in_i(t)}{in_{ambient}(t)}, P_{ambient}(t)), P_i(t-1), f_k(k, ...)) \tag{5.6}$$

$$\xi(x, y, K) = \begin{cases} x & K = 0 \\ y & K = 1 \end{cases} \tag{5.7}$$

*siendo f_k una función binaria (*0, 1*), que devolverá* 0 *siempre que el sistema esté estabilizado*

En la bibliografía pueden encontrarse multitud de funciones para predecir la estabilidad de sistemas sobre sistemas de control -véanse por ejemplo (Oldenburger, 1963), (DiStefano & DiStefano, 1989), (Kossiakoff & Sweet, 2005a) o (Kossiakoff & Sweet, 2005b), para tener una visión más completa-. El acercamiento más conveniente parece ser utilizar un mecanismo de lazo cerrado donde la entrada de f_k tome como valores de entrada el árbol de pregnancias de t - 1, calcule la desviación con respecto al estado anterior y devuelva *1* o *0* en función de la cercanía de esa variabilidad a *0*.

Por otro lado la opción de utilización de K_{max} es perfectamente válida dependiendo del tipo de servicio que se le quiera dar al usuario y se puede llegar a calcular realizando pruebas de simulación previas para comprobar en qué casos puede darse overfitting -con $\tau\ (v, x, y) = v \cdot x \cdot y$ no se produce sobreajuste-.

5.4. El mínimo producto viable

Si se dispone de un grupo aceptable de tester o usuario de pruebas -aceptable a nivel de población estadística- es posible lanzar al mercado un producto mucho más completo y adaptado. El modus operandi del departamento de UI en este caso suele ser enfrentar al usuario a la versión beta del producto, medir sus reacciones y realizarle encuestas en cuanto a satisfacción y expectativas.

Cuando no se dispone de un grupo de usuarios de pruebas relevante ni de departamento de UI es posible utilizar los AIE para configurar un mínimo producto viable a nivel de experiencia de usuario.

Es posible recoger el comportamiento de un grupo cerrado de usuarios, fusionar sus datos y configurar el AIE para que reacondicione el sistema partiendo de esos datos. Para ello es necesario implementar una pequeña arquitectura a nivel de preproducción que permita tanto la recogida como la fusión de estos datos.

5.4.1. Una arquitectura de pre-producción

Algunas aplicaciones como Google Analytics -ver (Palonka, Lora & Oziębło, 2017)- permiten recoger las interacciones de los usuarios y almacenarlas en la nube y de ellas se pueden extraer estadísticas y tomar decisiones relativas a la usabilidad, pero no es una plataforma inteligente, no toma sus propias decisiones (al menos no por ahora). Su idea de partida, aun así, es muy válida: almacenar en remoto los eventos que se producen en el momento en el que se producen para trabajar con ellos más tarde: un sistema unidireccional de cliente-servidor.

A la hora de almacenar los datos es importante guardar información adicional que permita trabajar con ellos. Resulta indispensable conocer el usuario que genera los datos, el AIE del que provienen. Para ello es imprescindible que al configurar el cliente se definan estos parámetros, pero también es necesario conocer cuál es el estado del AIE.

En el siguiente apartado veremos cómo realizar la fusión de datos, pero es importante conocer para qué sirve guardar el estado del AIE. Podemos tener a dos usuarios probando la misma aplicación, ambos con un AIE auditándola. Es posible que uno de ellos trabaje más con ella, sobrepasando el número máximo de

ciclos para actualización (K_{mas}), dando lugar a una mutación. La mutación puede haber producido un cambio en la estructura de los elementos, como por ejemplo un cambio en la posición de un elemento de menú. Las interacciones producidas a raíz de esa mutación ya no podrían asemejarse a las de otro usuario, los procedimientos mentales son distintos; y mucho más distintos serían aún si hubiera elementos que desaparecieran de la interfaz.

Una forma sencilla y rápida de comprobar que las estructuras de dos AIE son comparables es haciendo un *hash* de su estructura antes de almacenar sus valores. Este *hash* representa un número de versión del AIE y es muy útil a la hora de buscar y recuperar datos de una BD.

En el siguiente ejemplo vemos cómo funciona el sistema realizando el hash con un algoritmo MD5:

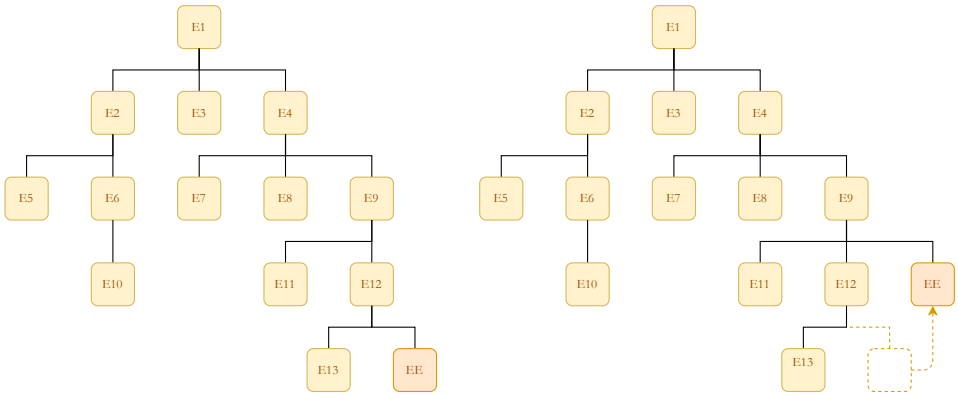


Figura 5.10: Hash del árbol de elementos

Para generar el hash se pueden utilizar los identificadores únicos de los elementos AIE compuestos en una cadena de texto:

árbol de elementos	Hash (MD5)
E1{E2{E2,E6{E9}},E3,E4{E7,E8,E9{E11,E12{E13,EE}}}}	0b461734450a6d6c223d85b1e6c67449
E1{E2{E2,E6{E9}},E3,E4{E7,E8,E9{E11,E12{E13},EE}}}	cad3810f5277ae4e1404a421c96c4e9b5

Cuadro 5.1: Hash del árbol de elementos

Como el uso de *hash* de uno u otro tipo no implica en este caso cuestiones de seguridad, se recomienda que en las implementaciones se prime la velocidad de cálculo a sus capacidades criptográficas.

El paso siguiente, una vez recogido el suficiente volumen de datos, es volcar los valores de las pregnancias y las interacciones desde el almacenamiento remoto hasta cada dispositivo, y dejar que el sistema AIE interprete los datos y modifique la interfaz.

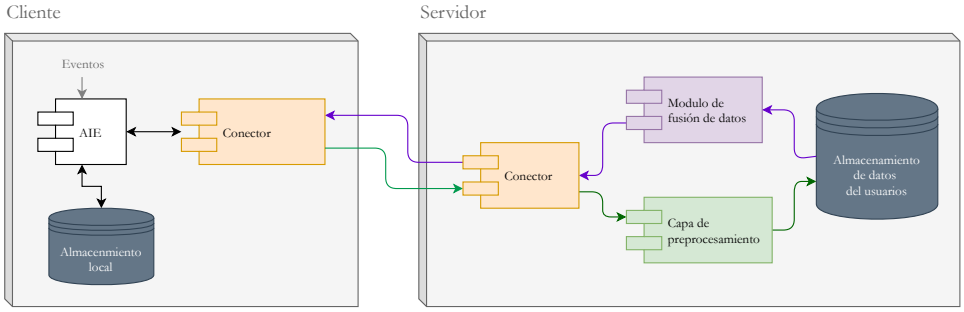


Figura 5.11: Arquitectura cliente servidor para fusión de datos

El AIE trabaja con un solo juego de valores de datos, por ello es importante que el almacenamiento devuelva como resultado la fusión de datos de los ambientes, de acuerdo a lo expuesto en el siguiente punto.

5.4.2. Fusión de datos de usuario

El servidor de almacenamiento debe devolver al cliente un único esquema de AIE: una estructura en forma de árbol con todos los datos que precise. Los datos mínimos que se deben almacenar son el número de interacciones de cada elemento y la pregnancia, pero también puede almacenarse otra información que ayude a simplificar los cálculos o a realizar recuperaciones de datos más efectivas.

La recuperación y fusión de la información sigue siempre el mismo procedimiento: recuperar estados comparables y realizar la suma de los datos. Los datos se considerarán comparables, como hemos visto en la sección anterior, si pertenecen al mismo AIE y tienen el mismo código de versión (*hash*).

La fusión de datos de interacciones y pregnancias se hace forma distinta. Los datos de las interacciones se suman directamente. La pregnancia calculada para un elemento AIE será el resultado de todas las interacciones, por tanto:

$$in_i = \sum_{a=1}^{A} in_i^a \tag{5.8}$$

siendo:

in_i *= interacciones del elemento i*

i =elemento i del ambiente a

A = número de total de AIE recuperados

Por otro lado la pregnancia del elemento *i* final será la suma ponderada de las pregnancias de los elementos *i* de los ambientes recuperados. Se puede dar el caso que las pregnancias totales de los ambientes recuperados fueran distintas por eso antes de guardar los datos es importante normalizarlos (se recomienda transformar todos los ambientes para que la pregnancia total sea 1):

$$P_i = \frac{\sum_{a=1}^{A} P_i^a * n_i^a}{\sum_{a=1}^{A} n_i^a} \tag{5.9}$$

En el siguiente ejemplo podemos ver cómo quedarían los cálculos para un caso sencillo en que se recuperen únicamente dos AIE.

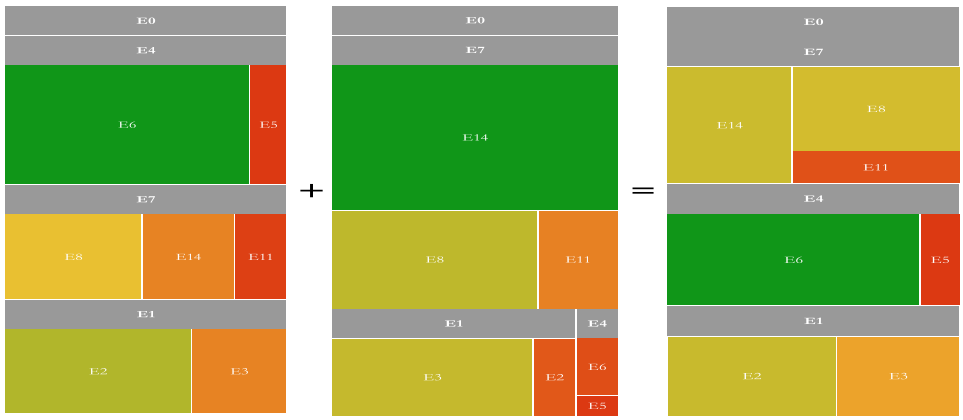


Figura 5.12: TreeMap: Fusión de pregnancias e interacciones

Cuadro 5.2: Fusión de pregnancias de 2 AIE comparables

	AIE 1		AIE 2		AIE General	
Elemento	Pregnacia	Interaccione	Pregnacia	Interacciones	Pregnacia	Interacciones
E0	1	2000	1	500	1	2500
E1	0,3	600	0,24	120	0,288	720
E2	0,2	400	0,04	20	0,168	420
E3	0,1	200	0,2	100	0,12	300
E4	0,395	790	0,04	20	0,324	810
E5	0,05	100	0,01	5	0,042	105
E6	0,345	690	0,03	15	0,282	705
E7	0,305	610	0,72	360	0,388	970
E8	0,15	300	0,208	104	0,1616	404
E9	0,05	100	0,108	54	0,0616	154
E10	0,1	200	0,1	50	0,1	250
E11	0,055	110	0,08	40	0,06	150
E12	0,05	100	0,05	25	0,05	125
E13	0,005	10	0,03	15	0,01	25
E14	0,1	200	0,432	216	0,1664	416
E15	0,02	40	0,05	25	0,026	65
E16	0,05	100	0,38	190	0,116	290
E17	0,03	60	0,002	1	0,0244	61

Como se puede apreciar en la Fig. 5.12 el elemento más significativo (los elementos de mayor pregnancia están coloreados en verde y agrupados por ambientes) es el E6 y el ambiente 1, que tiene mayor número de interacciones tiene más peso en la composición final.

Resultados y aplicaciones

6.1. Aplicación de los AIE en entornos web

A continuación se mostrarán unos ejemplos de la aplicación de AIE sobre una aplicación web. Veremos cómo afecta a elementos web desde componentes pequeños a vistas más complejas. Se recomienda antes de seguir leyendo consultar en Anexo II en el que se habla de cómo añadir elementos descriptivos AIE en código HTML.

6.1.1. Particionando las vistas

El paso previo antes de aplicar un AIE a una interfaz es hacer un pequeño análisis de cómo está conformada. En la mayoría de casos, si se encuentra bien diseñada desde el punto de vista arquitectónico (computacionalmente hablando) los componentes suelen dejar ya hecha esa división.

Figura 6.1: Particionando la interfaz

El segundo paso es decidir a qué propiedades físicas afecta la pregnancia, establecer los máximos y los eventos a tener en cuenta. En el caso de la Fig. 6.1 se pueden considerar por ejemplo las posiciones de los objetos (orden en los listados) y sus dimensiones (altura y anchura).

En código HTML si inducirá la etiqueta aie-properties, que contiene los límites máximo y mínimos de los recursos a utilizar:

```
<div id='example' aie-name="searcher" >
    <div aie-name="form"
      aie-properties="height|max:250|min:120px,position">
      <div aie-name="g_top" aie-properties="position">
        <button class="bt"aie-pregnancy="0.1"
          aie-name="b_mail" aie-trigger="click">
          Mail</button>
        <button class="bt"aie-pregnancy="0.1"
          aie-name="b_sheets" aie-trigger="click">
          Sheets</button>
        <button class="bt"aie-pregnancy="0.1"
          aie-name="b_draw" aie-trigger="click">
          Draw</button>
        <button class="bt"aie-pregnancy="0.1"
          aie-name="b_maps" aie-trigger="click">
          Maps</button>
      </div>
      <div aie-name="g_searcher">
        <div id="logo">
          Searcher
        </div>
        <div aie-name="searcher_bts"
```

```
            aie-properties="width|max:300|min:150">
            <input aie-pregnancy="0.3" aie-name="input"
              aie-trigger="keypress" type="text"/>
            <button class="btn" aie-pregnancy="0.2"
              aie-name="b_fell" aie-trigger="click">
              I'm Felling Lucky</button>
          </div>
        </div>
      </div>
      <div aie-name="g_fix" aie-properties="position">
        <button class="bt" aie-pregnancy="0.05"
          aie-name="b_gdpr" aie-trigger="click">
          Cookies noticer for RDPR
        </button>
        <button class="bt" aie-pregnancy="0.05"
          aie-name="b_help" aie-trigger="click">
          Help?
        </button>
      </div>
    </div>
```

En el ejemplo vemos: **"height| max:250| min:120px"**, que representa qué elemento puede modificar su altura siempre y cuando mantenga una dimensión que varía entre 120 y 250 píxeles.

6.1.2. Ambientes en componentes

En el caso de tener componentes con una funcionalidad no básica la aplicación de AIE puede no resultar trivial. En el ejemplo que vemos a continuación se aplica a un selector inteligente, que presenta al usuario los resultados en función de su interés.

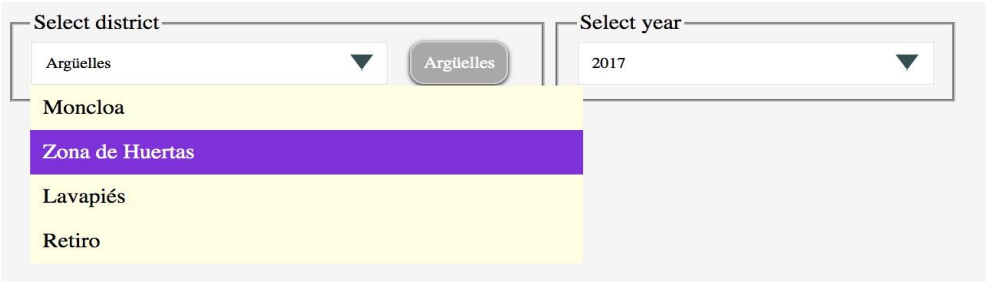


Figura 6.2: Ejemplo de ambiente en un componente

En el selector se definen dos subambientes, pudiendo pasar elementos del segundo al primero -esto se indica con la propiedad «position»-. De esta forma si

hay alguna opción que destaque del resto pasará a estar más arriba en el listado o incluso fuera del listado, para que su selección sea más directa.

En la Fig. 6.2, se puede observar como «Arguelles», un ítem del listado del selector de la izquierda, salta desde el listado para estar siempre visible para el usuario, al tratarse de la opción más utilizada, como se puede ver en el TreeMap de la pregnancia del componente que se muestra a continuación:

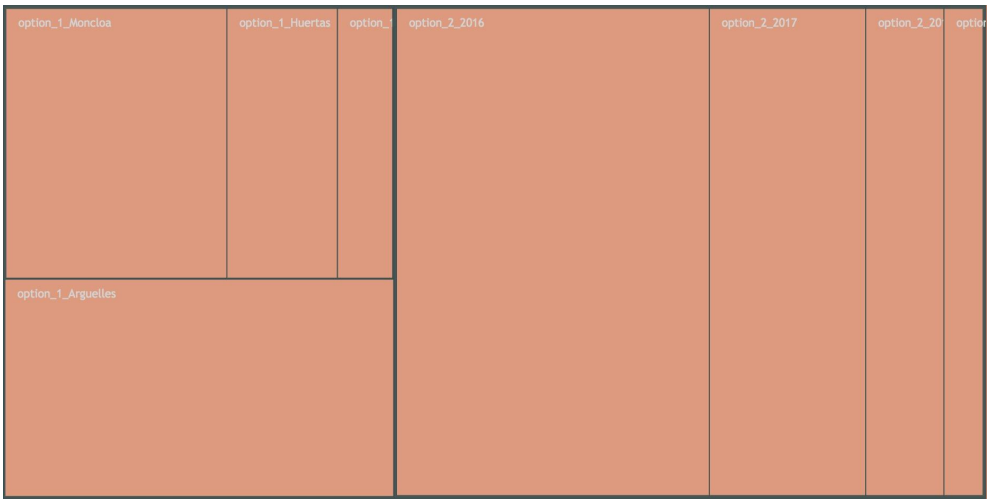


Figura 6.3: TreeMap de la pregnancia. Ejemplo de aplicación en componente.

El código HTML en este caso sería:

```
<div id='form-context' aie-name="filters">
    <fieldset
      id="options_district"
      class="selector"
      aie-properties="level"
      aie-name="districts_group">
      <legend>Select district</legend>
      <input readonly/>
      <div id="selector_options_district"
        aie-name="districts" aie-properties="position">
        <div aie-pregnancy="0.125"
          aie-name="option_1_Moncloa"
          aie-trigger="click">Moncloa</div>
        <div aie-pregnancy="0.125"
          aie-name="option_1_Arguelles"
          aie-trigger="click">Arg elles</div>
        <div aie-pregnancy="0.125"
          aie-name="option_1_Huertas"
          aie-trigger="click">Zona de Huertas</div>
        <div aie-pregnancy="0.125"
          aie-name="option_1_Lavapies"
          aie-trigger="click">Lavapi s</div>
```

```
        </div>
      </fieldset>
      <fieldset
        id="options_year"
        class="selector"
        aie-name="years_group">
        <legend>Select year</legend>
        <input readonly/>
        <div id="selector_options_year"
          aie-name="years" aie-properties="position">
          <div aie-pregnancy="0.125"
            aie-name="option_2_2018"
            aie-trigger="click">2018</div>
          <div aie-pregnancy="0.125"
            aie-name="option_2_2017"
            aie-trigger="click">2017</div>
          <div aie-pregnancy="0.125"
            aie-name="option_2_2016"
            aie-trigger="click">2016</div>
          <div aie-pregnancy="0.125"
            aie-name="option_2_2015"
            aie-trigger="click">2015</div>
        </div>
      </fieldset>
    </div>
```

En este ejemplo la utilización de un AIE no solo aportaría una solución a la usabilidad del componente sino que aporta una funcionalidad extra que no necesita ser programada, basta con definir correctamente los componentes.

6.1.3. Describiendo una aplicación

Describir una aplicación completa no es una tarea sencilla y mucho menos sabiendo que con ello se le delegará parte del control a una biblioteca externa. El uso de AIE, así como el uso de cualquier otra biblioteca externa puede introducir importantes variaciones en el funcionamiento de las aplicaciones, por ello resulta primordial tener pleno conocimiento de cómo funcionan antes de aplicarla.

A continuación veremos un ejemplo de uso sobre una aplicación online (ver Fig. 6.4). La aplicación es una single-page web application y el AIE controlará la posición de todos los elementos que se muestran.

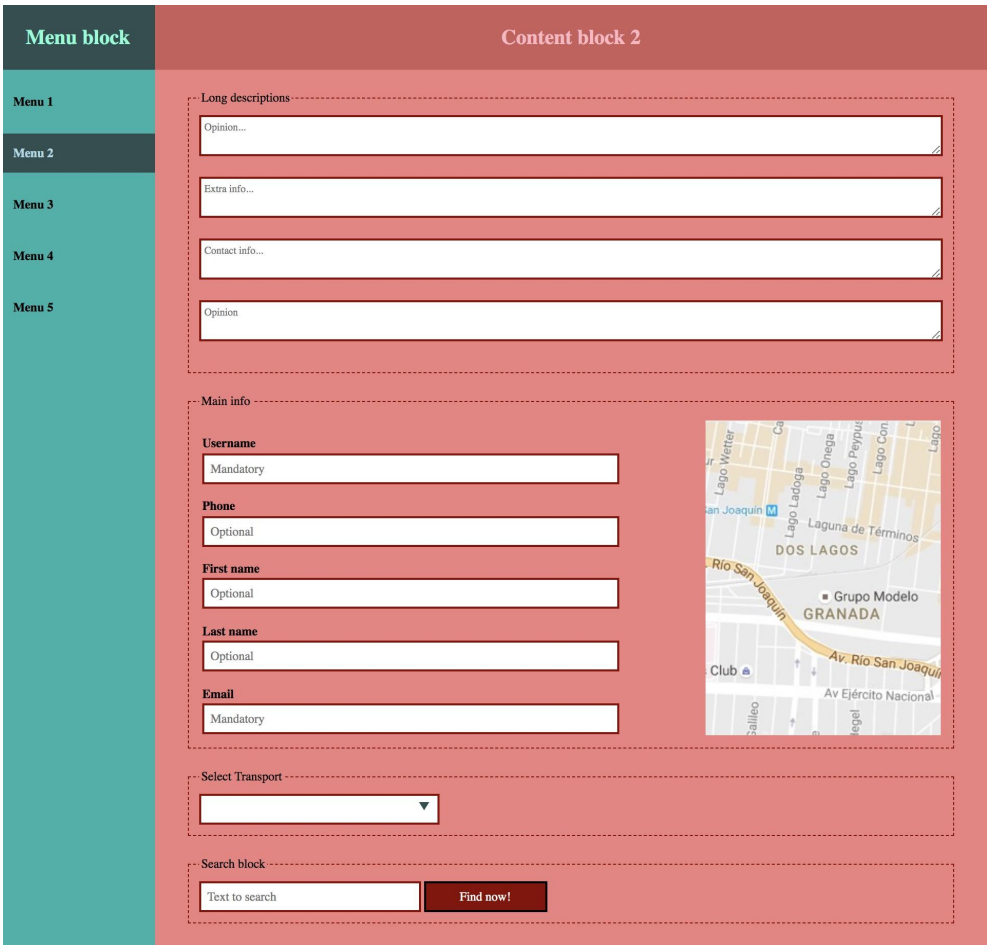


Figura 6.4: Ejemplo de descripción de una aplicación

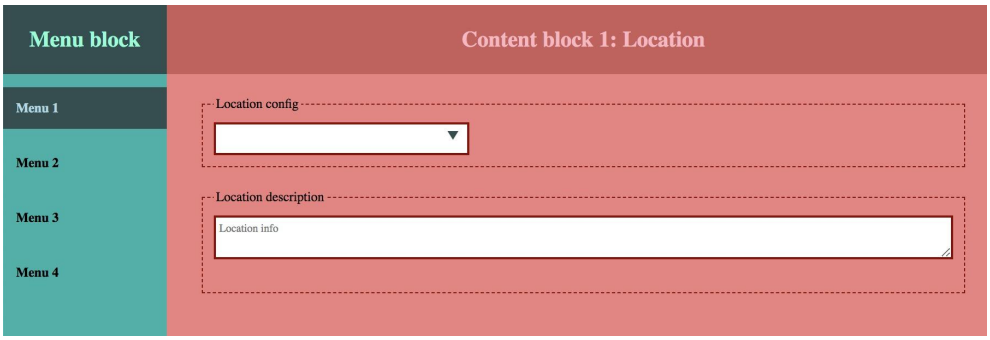


Figura 6.5: Ejemplo de descripción de una aplicación. Localización

La aplicación está formada por cuatro pantallas, de las cuales una de ellas no

tiene interés desde el punto de vista de la UI al tratarse únicamente de información legal (ver Fig. 6.6). El resto de pantallas contiene formularios de distinta índole con cajas de entrada unilínea, cajas multilínea, selectores o mapas. La distribución de elementos entre pantallas es temática y para este supuesto consideramos que debe ser inalterable.

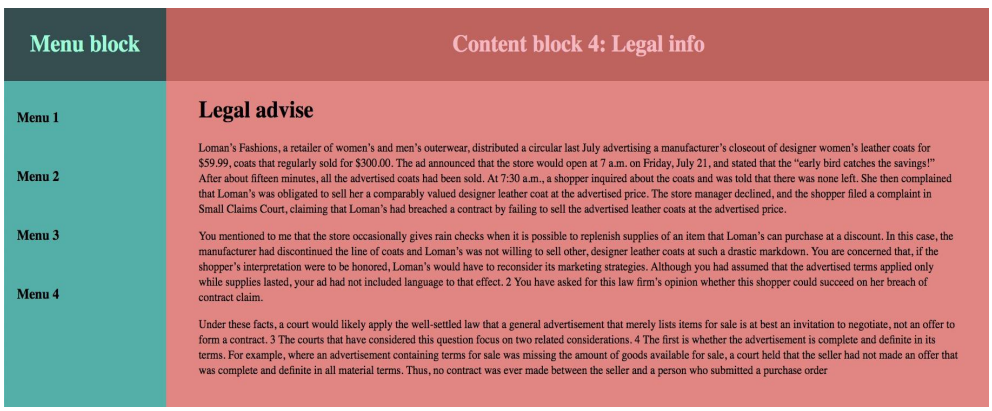


Figura 6.6: Ejemplo de descripción de una aplicación. Aviso legal

Además de los formularios la aplicación tiene un menú en el lado izquierdo que da acceso a las pantallas.

El primer paso para describir correctamente la aplicación para el AIE es distinguir los elementos a los que puede afectar y los que no. La pantalla cuatro, que es la tiene el contenido de la información legal referida anteriormente, se ignorará.

Figura 6.7: Ejemplo de descripción de una aplicación. Identificación de ambientes

En segundo lugar se agrupan los elementos en ambientes. La primera división que nos encontramos es entre el menú y el contenido. A partir de ahí los bloques de formulario marcan la estructura de los ambientes.

El tercer paso, que es el más complicado, es la asignación inicial de pregnancias a los elementos. La solución inicial más básica a este problema es dividir 1 entre un número de elementos 'rama' del árbol generado a partir de los ambientes de AIE y asignar esa pregnancia a dichos elementos.

Una solución un poco más efectiva y que ayudará a encontrar una solución más rápidamente al AIE es otorgar a los elementos una pregnancia aproximada al peso que estos tienen en la interfaz. Indicando si fuera necesario igualmente la

pregnancia libre.

Para el caso que nos atañe se ha decidido asignar al sistema una pregnancia libre de $0,18$ (sobre una pregnancia total de la aplicación de 1) al estar el contenido de dos de las pantallas mucho menos cargado. La distribución de la pregnancia se puede ver en la figura. La pregnancia libre se ha repartido de forma equitativa entre las secciones 1 y 3.

Figura 6.8: Ejemplo de distribución inicial de pregnancia

La pantalla con el contenido de mayor pregnancia es el bloque 2 (ver Fig. 6.4).

Si observamos el Treemap generado a partir de las pregnancias, podemos adivinar que la mayor parte pertenece a esta sección (ver Fig. 6.9).

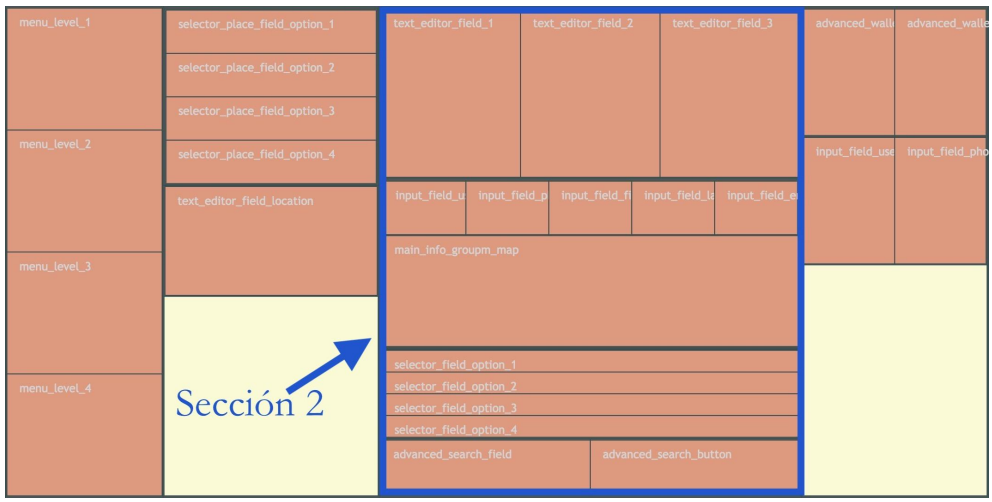


Figura 6.9: Ejemplo de descripción de una aplicación. Treemap de pregnancia

Realizando una simulación de la interacción del usuario podemos observar como varía la pregnancia de los elementos.

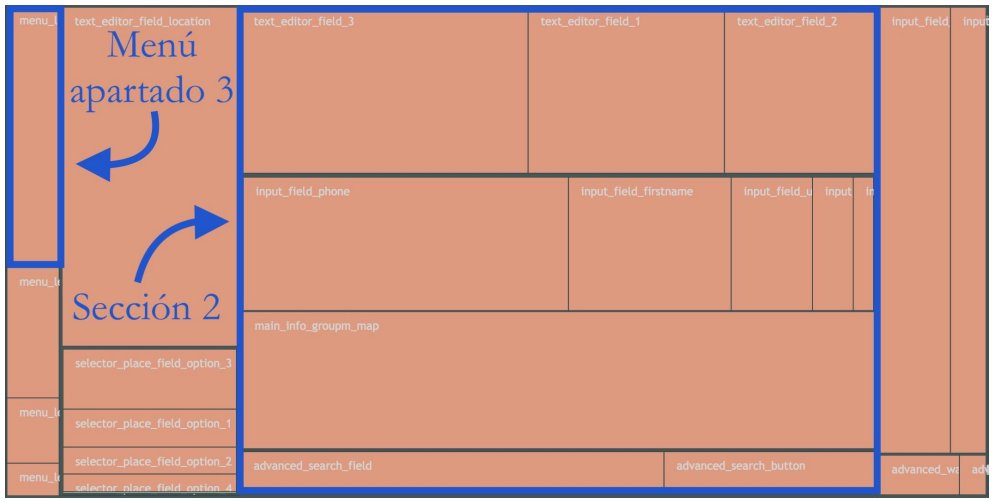


Figura 6.10: Ejemplo de descripción de una aplicación. Treemap de pregnancia final

En el ejemplo la mayoría de las interacciones del usuario han ido centradas en la sección 2, a pesar de que el botón de menú más utilizado ha sido el que da acceso al apartado 3. Esto tiene sentido al ser la sección 2 la más amplia.

Tras forzar la mutación los elemento de la interfaz se han reordenado y la pregnancia libre de las secciones 1 y 3 se ha consumido. El aspecto de la aplicación ha variado notablemente ajustándose a la utilización del usuario.

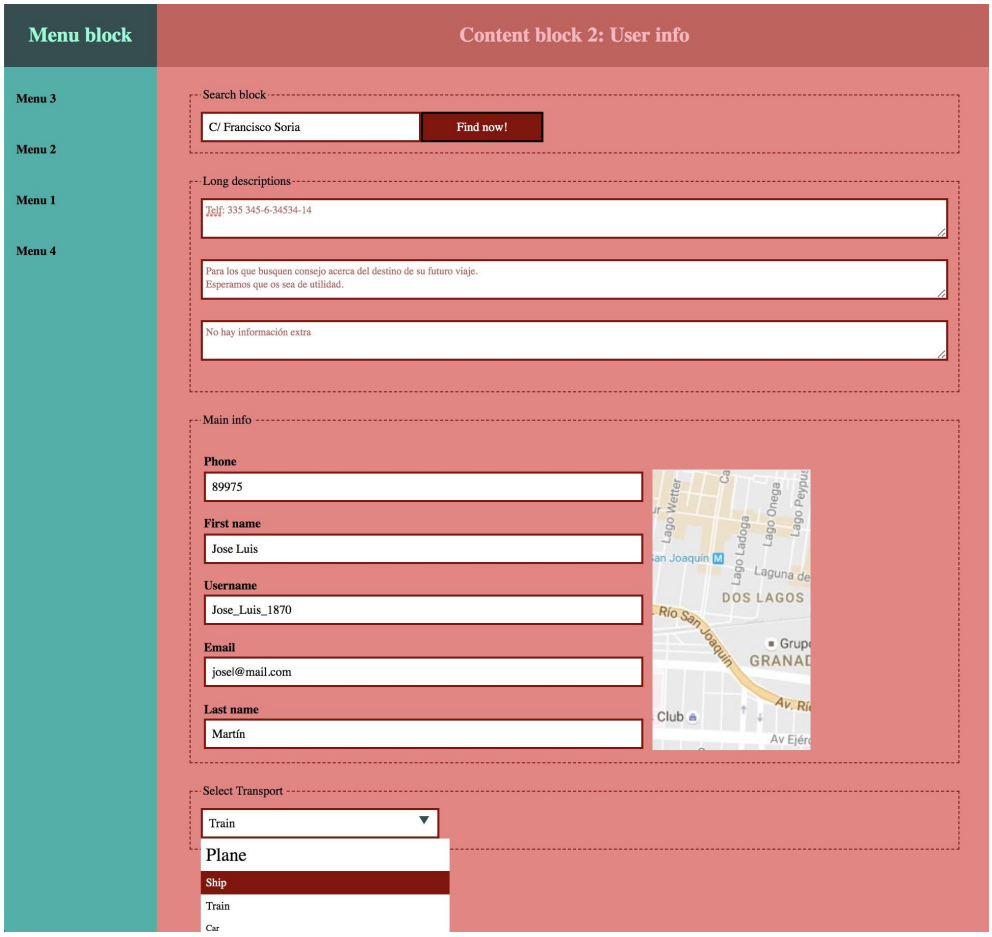


Figura 6.11: Ejemplo de descripción de una aplicación. Estado final de la aplicación

6.1.4. Impacto en el desarrollo de sistemas FFDD

Podemos estimar qué impacto tiene la utilización de un desarrollo de *feature-flags* en una aplicación comercial que utilice AIE con desactivación automática de *features*.

Para ello partimos del estudio del Pew Research Center de citado anteriormente sobre sobre el uso de Twitter La aplicaciones de Twitter, tanto las apps para smartphones como la aplicación web, tienen una gran cantidad de código y funcionalidad que ese 28 % no utiliza. Analizando el Javascript utilizado en la aplicación web a través de las herramientas que facilita Google Chrome, podemos lo ver siguiente:

Cuadro 6.1: Listado de archivos Javascript cargados en la web de Twitter para su funcionamiento

Script	Tamaño (kB)	Tiempo de carga
loader.WideLayout.f7897504.js	5,6	23 ms
main.29ffce14.js	166	147 ms
ondemand.emoji.es.fdff9774.js	47,3	36 ms
ondemand.EmojiPicker.5107dfc4.js	52,3	40 ms
ondemand.EmojiRainAnimation.16f139f4.js	1,5	58 ms
ondemand.HoverCard.b1fb6514.js	5,9	21 ms
ondemand.InlinePlayer.66fe0754.js	16	262 ms
ondemand.PlayerBase.dde9a9c4.js	22,9	236 ms
ondemand.PlayerBase~ ondemand.lex.d95a82b4.js	7,8	239 ms
ondemand.PlayerHls12.cdd6ca14.js	72,9	240 ms
ondemand.PlayerUi.8a1e4644.js	34,6	264 ms
polyfills.b548f144.js	15,3	39 ms
shared~ bundle.DirectMessages.a51ff9c4.js	31,6	30 ms
shared~ bundle.HomeTimeline.c1bc0384.js	179	76 ms
shared~ bundle.Settings.b6c043b4.js	11,3	53 ms
shared~ bundle.UserProfile.6cd733b4.js	6,3	30 ms
vendors~ main.b3a10864.js	139	88 ms
vendors~ ondemand.EmojiPicker.55c92664.js	4,6	25 ms

La página central de Twitter carga alrededor de 1217, 3 kB que componen la base mínima del código que se utiliza en la página. A ello hay que se sumar 9, 2 kB que son los destinados en gestionar la carga del listado de tweets del muro del usuario. Con ello tenemos un total 1226, 5 kB.

Si navegamos por las distintas secciones (notificaciones, publicaciones mensajes directos, etc...) tenemos que se cargan los siguientes scripts adicionales:

Sección	Script	Size (kB)
User profile	bundle.UserProfileTimelines.22352704.js	6,6
	bundle.UserProfile.473843d4.js	61,5
	bundle.UserLists.89bc77d4.js	77,7
Notifications	bundle.Notifications.2e226034.js	34
Direct messages	bundle.DirectMessages~ bundle.TweetMediaDetail.94d31c04.js	22,8
	bundle.DirectMessages.0c8a7cd4.js	177
Bookmarks	bundle.Bookmarks.433c33b4.js	2,6
Publish	bundle.RichTextCompose.10ef88a4.js	304
Timeline	bundle.HomeTimeline.5388edd4.js	9,2

Cuadro 6.2: Listado de archivos Javascript cargados en la web de Twitter por sección

Rápidamente es posible percatarse de que para ese 28 % de usuario el 45 % del código, en este caso escrito en Javascript, no le es necesario y parte de la GUI de Twitter reservada para tal efecto tampoco. Por suerte, la web de Twitter está muy bien diseñada desde el punto de vista arquitectónico, y el código javascript se carga de forma dinámica únicamente cuando se invoca una funcionalidad a través de un service worker del navegador.

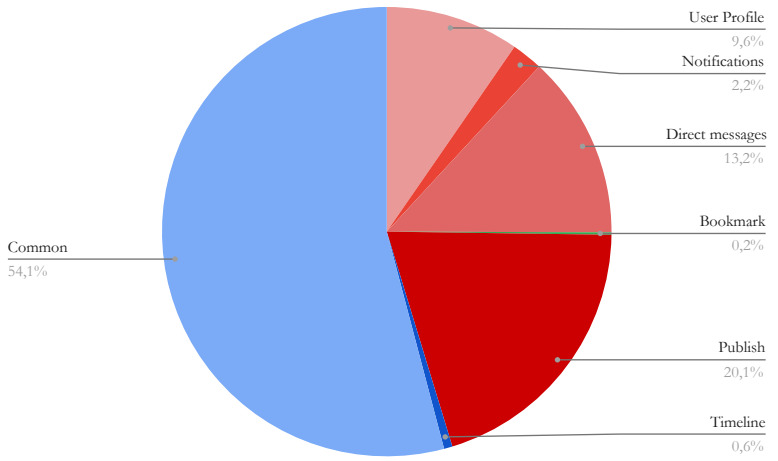


Figura 6.12: Porcentaje de código Javascript de Twitter en función de su funcionalidad asociada

Partiendo de estos datos podemos estimar que un porcentaje similar de código está dedicado a esta tareas en la apps para dispositivos móviles. Si la aplicación de

Twitter en un navegador (HTML + Javascript + CSS + Imágenes) es de 3 MB, el tamaño de las funcionalidades no asociadas es de aproximadamente un 22 %. Llevando estos porcetaje a la app de Android oficial de Twitter tenemos que si ocupa entre 33 MB y 35 MB (dependiendo la versión de Android para la que han sido compiladas) en los dispositivos, el ahorro de recursos de almacenamiento sería bastante grande (aunque posiblemente no se llegue a los 6, 7 MB que representan el 20 % del peso total de la app).

Si se decidiera implementar un sistema FFDD + AIE ese porcentaje de código bajaría alrededor de un 15 % (pues habría que incluir la biblioteca de AIE en la app), más la creación de un repositorio para features fuera del Google Play Store, que supondría un coste adicional en almacenamiento y desarrollo, pero que no repercute al usuario en términos de consumo de recursos. Esta adopción de un formato FFDD + AIE, por contra, supondría también una mejora en el proceso de desarrollo, como hemos visto, dando más flexibilidad a la inclusión de nuevas funcionalidades, adaptabilidad en UI a usuarios o flexibilidad en los despliegues.

Conclusiones

7.1. Visión general

El objetivo principal de este trabajo es analizar si es posible la creación de un sistema que posibilite la automatización en el proceso de adaptación de interfaces, de forma personalizada y automática, basándose en el comportamiento del usuario. Como se ha podido comprobar a lo largo de este documento la búsqueda de la solución ha sido exitosa y además aplicable a la solución de problemáticas de áreas vinculadas.

El resultado de la investigación es la definición de los Ecosistemas de Interfaz Adaptativos. Un AIE recoge información de las interacciones del usuario y la combina con información descriptiva de la interfaz, para poder obtener datos que permiten la modificación de la interfaz, de forma que esta se adapte más al flujo de trabajo del usuario.

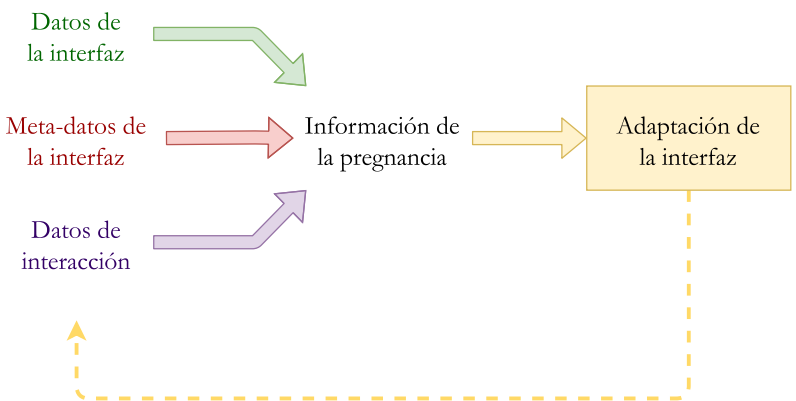


Figura 7.1: Flujo de un AIE

Los datos de descripción de la interfaz permiten saber qué tipos de transformaciones puede soportar ésta, mientras que los datos las interacciones del usuario determinan cómo realizarlas. El resultado final de la fusión de estas dos fuentes de datos es el mapa de pregnancia del aplicación. En mapa de pregnancia podría entenderse como un plano que representa los puntos de interés de la aplicación para el usuario.

El mapa de pregnancia es un mapa personalizado, diferente para cada usuario, que permite ajustar la aplicación dándole más relevancia a los elementos más interesantes para dicho usuario.

En determinadas ocasiones el mapa de pregnancia podría entenderse como un *Heatmap* (Mapa de calor) pero su función final es muy diversa, no trata de identificar zonas calientes, sino componentes o elementos de la interfaz que resulten útiles para el usuario. Una vez identificados estos componentes, se pueden presentar de una forma que resulten más accesible al usuario.

La adaptación de la interfaz se realizará de acuerdo a los mismos criterios que permiten definir la pregnancia de los elementos, trasladando los elementos más relevantes para el usuario a posiciones más relevantes. De esta forma puede ignorarse la naturaleza física de la interfaz en la aplicación del procedimiento descansando sobre la implementación la responsabilidad última de la transformación de la capa de presentación.

El rediseño de la interfaz en función de la pregnancia no es proceso universal ni cerrado, de hecho depende completamente del entorno socio-cultural del usuario. Las leyes de la Gestalt pueden marcar cuáles son las normas a la hora de confeccionar una interfaz gráfica en función de la naturaleza del usuario, pero no son normas válidas para por ejemplo las interfaces auditivas.

7.1.1. Aplicación de sistemas AIE

La aplicación de sistemas AIE sobrepasa su utilización dentro del entorno de interfaces para integrarse como parte de un elemento arquitectónico dentro de sistemas orientados al desarrollo de features-flags (FFDD) y ayudar agilizar la activación y desactivación de éstas gracias a su información sobre la pregnancia.

Los sistemas FFDD son sistemas derivados de los sistemas orientados al desa-

rrollo de *features* (FDD) que incluyen un control dinámico de activación y desactivación de características (*features*) a través de *flags*.

La activación y desactivación de features oferta grandes posibilidades a la hora de ofrecer al usuario un mejor aprovechamiento de los recursos a nivel de hardware y una mejor experiencia dando la posibilidad de tener acceso a features experimentales, añadiendo o quitando características en función de sus preferencias, o mejorando las interfaces y procesos internos a través de test A/B.

El resultado de la aplicación de FFDD+AIE presenta una mejora en el proceso de desarrollo dando más flexibilidad a la inclusión de nuevas funcionalidades tanto en arquitecturas horizontales como verticales, además mantiene los beneficios del desarrollo orientado a features como compartimentación del código, que aporta mayor flexibilidad en despliegues y aislamiento de errores.

La adición de AIE a la arquitectura soluciona sobretodo uno de los problemas más acuciantes del uso de *flags*, que es el de la deuda técnica.

La versatilidad que permite la temporización de activación de *flags* va más allá de resolver el problema de deuda técnica, también puede ser la base para la implementación de un sistema de garbage collector que permita la limpieza periódica de los datos de la aplicación.

7.2. Contribuciones a la investigación

Recuperando la comparativa entre los problemas más comunes de las interfaces -ver (Ehlert, 2003), (Zuffi y col., 2007) y (Carroll, 1988)- y las soluciones aportadas por las distintas metodologías analizadas -ver las tablas 3.2, 3.2 y 3.2-, podemos considerar que los AIE presentan una alternativa más completa aportado soluciones en todas las áreas :

	AIE	TRIDENT	PMT	AOP
(1)	Parcial	No	Sí	No
(2)	Parcial	No	Sí	Sí
(3)	OK	No	Sí	No
(4)	OK	Parcial	No	No
(5)	OK	No	No	Sí

Cuadro 7.1: Comparativa AIE con el resto de metodologías

Siendo:

- (1) El uso de opciones de menú con un orden erróneo y una separación confusa.
- (2) Inapropiado e inconsistente uso de colores, sonidos, iconos, y demás contenido multimedia que no se ajusta a la cognición del usuario.
- (3) Ilegibilidad de texto debido a problemas de apariencia.
- (4) Carencia de la propiedad de adaptación de la interfaz a los gustos del usuario.
- (5) Desaprovechamiento de los recursos computacionales asociados a la creación de interfaces.

La utilización de AIE no es, a la luz de lo que se puede ver en las tablas 7.1 y 7.1, la solución definitiva, pero sí es más completa que las que propuestas anteriormente. Los AIE, por su propia naturaleza, se ocupan fundamentalmente de la parte representativa dejando aparcado el contenido.

En relación a los objetivos planteados inicialmente, se han complido de forma total los siguientes apartados:

- Se han estudiado de las tendencias en el desarrollo de interfaces adaptativas, metodologías de diseño y experiencia de usuario.
- Se ha revisado revisado el estado del arte, fuera del ámbito de la computación, de teorías y técnicas que ayudan a la automatización de tareas de diseño de interfaz y experiencia de usuario.
- Se han identificado y evaluado los principales los problemas más comunes de interacción persona-ordenador.
- Se ha modelado un sistema que permite dar solución a los problemas identificados partiendo de los datos de interacción del usuario.
- Se ha adaptado el sistema propuesto para su integración en proyectos en funcionamiento con un coste de tiempo y recursos pequeño, logrando mantener un alto grado de autonomía entre el sistema y el proyecto.

- El sistema propuesto es suficientemente genérico para cubrir el mayor tipo de interfaces posibles, con independencia de su naturaleza física, así como el mayor tipo de plataformas posibles.
- El sistema puede integrarse dentro de los ciclos de vida de las metodologías actuales sin necesidad de la interacción con un equipo de diseño o UX.
- El sistema propuesto se adapta a los estándares de la industria, así como a sistemas de producción de software no comerciales.
- El sistema da solución a la creación de un mínimo producto viable, partiendo de una idea generalista del producto creado a partir de las experiencias colectivas de múltiples y heterogéneos usuarios.
- Se han evaluado los resultados en modelos de producción.

7.3. Evaluación y líneas de investigación futuras

De acuerdo con lo expuesto, han quedado fuera del ámbito de este estudio tres problemas habituales de las interfaces:

- Inapropiada terminología o iconografía.
- Contenido erróneo de los textos.
- Ausencia de sistema de ayuda.

La dificultad en estos tres campos, que no se puede solucionar con la propuesta aquí expuesta, radica en la generación de nuevos elementos dentro de la interfaz, pero ello depende mucho del significado de los campos y no de su apariencia.

7.3.1. Inapropiada terminología o iconografía

El primero de estos problemas, que es de naturaleza eminentemente visual, radica en que los signos y símbolos utilizados no son reconocibles por todos los usuarios de una aplicación. Esto puede deberse fundamentalmente a dos causas.

La primera, que es una aproximación desde la perspectiva del diseño de la interfaz, es una mala elección de la iconografía, y puede darse por un mal diseño de la interfaz causado por un desconocimiento de la semiótica que circunscribe a aplicación, ver (Brejcha, 2015). La segunda causa la encontramos realizando una aproximación desde el punto de vista de usuario, dado que podremos encontrarnos con sujetos con poca experiencia la utilización de interfaces y/o una capacidad de abstracción perceptiva baja, de tal forma que les sea imposible asociar los símbolos representados. Bajo esta premisa el problema podría encontrar solución teniendo un repositorio de símbolos lo suficientemente grande para ofrecer alternativas al usuario a la hora de reemplazar la simbología utilizada por otra que le resulte más reconocible.

A día de hoy podemos encontrar frecuentemente la aparición de "temas" (*themes*) como una opción dentro de los paneles de configuración/personalización de muchas aplicaciones y pese a que esto originalmente no ha sido pensado como una solución a este problema, esta funcionalidad puede llegar a ser la base para solventar el problema de la iconografía.

7.3.2. Contenido erróneo de los textos

El contenido erróneo de textos es un problema muy ligado al de una incorrecta iconografía, en el sentido que los mensajes de la interfaz no llegan a ser comprensibles para el usuario. La diferencia fundamental entre estos dos problemas radica en que la tradición iconográfica es más universal y la textual más local.

El error en los textos puede venir dado por muchas causas, desde problemas tipográficos a causas gramaticales, aunque en este ámbito el problema más interesaría resolver sería el estrictamente formal, es decir la elección correcta de vocablos para definir los conceptos y acciones a los que hace referencia la interfaz.

No es posible proponer un grupo de glosarios temáticos para que el usuario seleccione a su conveniencia (como se hace con la iconografía) y se intuye que la automatización al respecto puede seguir otros caminos como por ejemplo la generación de glosarios y diccionarios comunes del que puedan nutrirse distintas aplicaciones y así aprovechar la experiencia del usuario al cambiar de unas aplicaciones a otras.

7.3.3. Ausencia del sistema de ayuda

La creación de sistemas de ayuda adecuados a la experiencia del usuario es un tema que se ha tratado ampliamente en estudios relacionados con gamificación. La confianza, experiencia y capacidad de adaptación del usuario son determinantes en el dinamismo que supone el aprendizaje del uso de una aplicación - ver (Rousseau, Sitkin, Burt & Camerer, 1998) y (Mayer, Davis & Schoorman, 1995).

Los Ecosistemas de Interfaz Adaptativos se basan en gran medida en sistemas gamificados, por ello su adaptación a sistemas de ayuda no debería ser excesivamente compleja.

La diferencia fundamental entre el sistema de ayuda y el sistema «ayudado» es que la demanda y utilización de elementos de ambas interfaces es inversamente proporcional, es decir, un usuario necesitará con el tiempo más ayuda para aquellas funcionalidades de la interfaz que utilice con menos frecuencia, asumiendo que aquellas que utilice con más frecuencia las conocerá mucho mejor.

Por otro lado podemos encontrar sistemas de ayuda cuya funcionalidad se encuentra inmersa en la funcionalidad del sistema al que ayuda, como por ejemplo en el caso de existir un componente asistencial. En este supuesto la aplicación de un AIE debería revisarse en función de la complejidad del asistente, y no existe por tanto un solución evidente partiendo de lo aquí tratado.

7.3.4. Mejora de la integración de los AIE

A lo largo de este estudio se ha tratado de impulsar una visión de los AIE como un elemento ajeno al proyecto, con el nivel de acoplamiento más bajo posible para que sea fácilmente reutilizable, pero en vistas a conseguir una mayor facilidad de utilización de cara al desarrollador, un caso de estudio a evaluar sería la integración completa de un AIE dentro de una biblioteca de componentes de interfaz.

Las ventajas en este supuesto serían notables de cara a poder disponer por defecto de una recogida de interacciones más afectiva y un aprovechamiento de recursos más adecuado, además de poder tener acceso a propiedades de los elementos de la interfaz, que no se pudieran manipular desde una capa de abstracción superior.

El programador dispondría también de configuraciones por defecto y la descripción de la interfaz se haría menos verbosa, identificando algunos elementos

como menús, botones, etc. a partir de su definición como componente.

7.3.5. Experiencias negativas e intenciones finales

Siguiendo con los estudios de (Martin-Hammond y col., 2018) y (Kirisci & Thoben, 2018), el usuario puede ofrecer una retroalimentación negativa a la interfaz partiendo de experiencias poco satisfactorias.

La información adicional que se podría extraer de esas experiencias ayudaría a confeccionar con mayor exactitud el mapa de pregnancia de una aplicación. La detección de los patrones de uso insatisfactorios conlleva una pérdida de pregnancia a los elementos involucrados. La dificultad de la detección automática de estos patrones ya está avanzada en varios estudios como el de Martin-Hammond, pero no está resuelto cómo ponderarla para adaptarla a un modelo de pregnancia como de los Ecosistemas de Interfaz Adaptativos. Las dificultades fundamentales son las siguientes:

- Para llegar a calibrar correctamente los pasos en falso que realiza un usuario será necesario reconocer flujos completos de una acción, es decir, identificar cuál ha sido la intención final del usuario. De esa forma se evitará por una parte penalizar a elementos de la interfaz por acciones que carecen de importancia y por otro lado ajustar los resultados de acuerdo con una lógica que se adecue más al comportamiento del usuario.

  Para entender un poco más este problema supongamos el siguiente ejemplo: *Un usuario interacciona con una aplicación de edición de textos. Su intención final es añadir un enlace de hipertexto a una palabra. Los iconos de la aplicación le resultan muy confusos. Presiona varios de ellos con el texto seleccionado y consecutivamente a cada acción el botón de «Deshacer acción» hasta que consigue su objetivo.* Parece evidente que no deben penalizar, en términos de pregnancia, las acciones de búsqueda infructuosas del usuario, sino únicamente premiar los componentes que le permiten llegar a la consecución final de su objetivo, en este caso el botón de añadir un hiperenlace, obviando además la reiteración de uso del botón "Deshacer".

- El uso infructuoso de una acción puntual dentro del periodo de adaptación

de un usuario a la interfaz, puede penalizar el acceso a esa acción en el futuro. Pueden existir grandes diferencias entre los periodos de adaptación de unos usuarios a otros y eso es algo que los AIE no tienen aún en cuenta a la hora de realizar sus cálculos. Las capacidades perceptivas, memorísticas y adaptativas de los propios usuarios a la tecnología son muy distintas por muy parecido que sea su perfil socio-cultural.

- Una respuesta negativa de un usuario a una acción puede estar provocada por un problema en el elemento reproductivo se utiliza la interfaz: una pantalla táctil defectuosa, un altavoz o un micrófono de baja calidad y no la propia interfaz.

7.3.6. Adaptación de leyes de la Gestalt a interfaces de distintas naturalezas

Las leyes de la Gestalt fueron diseñadas como un paradigma de percepción visual, pero como hemos visto en este estudio pueden ser adaptadas a otros sistemas perceptivos reduciendolas a sus parámetros más generales y eliminado las referencias a lo que es estrictamente el campo visual.

La adaptación de las leyes de la Gestalt no se realizado en su totalidad, puestos que para que el modelo presentado no era necesario. De esta forma reside en la implementación del sistema la responsabilidad y la libertad de hacer hacer los cálculos para la obtención de la pregnancia.

Uno de los puntos más atractivos que nace a partir de este estudio es la búsqueda de leyes de carácter más universal que pueden determinar con más precisión las características de la percepción. Uno de los puntos claves en ese futuro estudio debería ser la incidencia de la variabilidad socioeconómica y cultural, pero también otras como la edad o el sexo.

A día de hoy sabemos que las circunstancias en las transcurre el aprendizaje sensorial afecta al desarrollo de nuestro cerebro y por ende a la forma en la que captamos nuestro entorno, por ello comprendiendo y cuantificando de forma correcta los factores externos del desarrollo será posible calcular, de una forma más determinista, la pregnancia de elementos que percibimos.

7.3.7. FFDD y el versionado de aplicaciones

Como consecuencia de la descomposición de la aplicación en *features*, el versionado se vuelve más complejo, ya que depende del número de features y las versiones de cada una de ellas en el sistema. Si únicamente hubiera un número limitado de features y todas ellas fueran parte de una release o un repositorio debidamente actualizado, el número de versión de la aplicación sería sencillo de calcular. Pero precisamente una de las ventajas de utilizar features es que cada funcionalidad puede nacer y evolucionar de forma independiente, por ello el versionado se parecerá más al de un sistema de paquetes.

Glosario

A continuación se incluyen algunos términos que serán de uso frecuente en este documento. Se recogen aquí para evitar su repetición dificultando la lectura del documento.

Análisis de componentes principales (PCA)

Esta técnica fue inicialmente desarrollada por Pearson a finales del siglo XIX. La idea fundamental es transformar linealmente un conjunto original de datos a nuevo sistema de coordenadas en el cual la varianza de mayor tamaño del conjunto de datos es capturada en el primer eje de coordenadas (llamado el Primer Componente Principal) y la segunda varianza más grande es el segundo eje de coordenadas, y así sucesivamente, es decir reducir la dimensionalidad del conjunto inicial de coordenadas.

El primer paso es calcular la matriz de covarianza o matriz de coeficientes de correlación (una matriz simétrica que condiciona la existencia de una base completa de vectores propios de la misma). Transformar las coordenadas de origen a las coordenadas de la nueva base es precisamente la transformación lineal necesaria para reducir la dimensionalidad de datos.

Dada una muestra con n individuos para cada uno de los cuales se han medido m variables, el PCA permite encontrar un número de factores subyacentes **p<m** que explican el valor de las m variables para cada individuo. Si existen estos p factores subyacentes, puede interpretarse como una reducción de la dimensionalidad de los datos. Cada uno de los p encontrados se denomina componente principal, de ahí el nombre del método.

Agente software

En la bibliografía no podemos encontrar una definición universalmente aceptada de lo que es un agente software. Quizás lo más habitual es definirlos desde el punto de vista cualitativo, basándose en las propiedades o en las habilidades del agente, descritas por Wooldridge:

"Un agente es un sistema informático autónomo, autocontenido, reactivo y proactivo, generalmente con un enfoque central de control, que puede comunicarse con otros agentes a través de un Lenguaje de comunicación del agente(ACL)".

(Wooldridge, Müller & Tambe, 1996)

Un uso más concreto de los agentes se da dentro de un sistema informático, donde conceptualizan o implementan en términos de conceptos más generalmente aplicados a los humanos (como creencias, deseos e intenciones), dando lugar a una arquitectura BDI (beliefs, desires and intentions) o desde el punto de vista operacional según el test Huhns-Singh que rige las organizaciones de agentes):

"Un sistema que contiene uno o más agentes reputados (aceptados en consenso por el resto de la organización) debe cambiar sustancialmente si se agrega otro agente reputado al sistema".

(Huhns & Singh, 1997)

En (Ingham, 1997), es posible encontrar un amplio resumen bibliográfico de las definiciones de agentes.

Computación ubicua

La computación ubicua es un área de la computación que se centra en el entorno del usuario, de forma que los elementos electrónicos forman parte del ambiente y no son elementos diferenciados.

La idea general es que los dispositivos electrónicos se encuentren embebidos en el entorno, formando parte de otros objetos de uso común. La tecnología subyacente que soporta la computación ubicua incluye el Internet, el middleware, sistemas operativos, código móvil, sensores, microprocesadores, interfaces de usuario,

redes, protocolos de comunicación, posicionamiento y ubicación y nuevos materiales. Dependiendo del contexto en el que se use también puede referirse a ella con los términos Pervasive Computing, Ambient Intelligence, Calm Technology o Things That Think, y resulta innegable su vinculación con el Internet of Things (IoT).

Pregnancia

Cualidad que poseen las figuras que pueden captarse a través del sentido de la vista. Está vinculada a la forma, el color, la textura y otras características que hacen que la persona que observa pueda captarla de manera más rápida y simple.

En este documento la pregnancia se aplica desde una perspectiva más amplia, no únicamente visual, sino haciendo referencia también al resto de los sentidos.

La pregnancia es también una de las leyes de la Gestalt. La Ley de Pregnancia fue establecida por la Psicología de la Gestalt, véase (Behrens, 1998), y adoptada por la Escuela de la Bauhaus en Alemania en 1919. Según esta ley, existen figuras más pregnantes que tienden a percibirse primero o producir un mayor impacto visual.

SIFT (Scale-invariant feature transform)

Scale-invariant feature transform o SIFT es un algoritmo de detección de características, creado por David Lowe y patentado por la Universidad de British Columbia de Canadá en 1999.

La idea fundamental del SIFT es extraer vectores de características de objetos en imágenes y compararlos con otros vectores de características etiquetados previamente almacenados en una base de datos.

Del conjunto de vectores de características similares extraídos de la base de datos, se identifican subconjuntos de puntos clave que concuerdan con el objeto y su ubicación, escala y orientación en la nueva imagen para filtrar las coincidencias. Para localizar rápidamente estos subconjuntos se utiliza una implementación eficiente de tabla hash de la transformada de Hough generalizada -ver (Gorte & Sithole, 2012)-. Por cada conjunto de tres o más características que concuerdan con

un objeto y su posición, se someten a un detalle más exhaustivo descartando los valores atípicos. Finalmente, la probabilidad de que un conjunto particular de características indique la presencia de un objeto se calcula a partir de la precisión del ajuste y el número de posibles coincidencias falsas. Los vectores de características que pasan todos los filtros pueden aceptarse como correctos con un alto grado de confianza.

La descripción completa del algoritmo se puede encontrar aquí (Lowe, 1999).

Self-explanatory

Concepto muy asociado al de user-friendly. Hace hincapié en la propiedad de simpleza de un software del que no se necesitan conocimientos previos para entender su funcionamiento.

Single-page application

Una single-page application es una aplicación web que interactúa con el usuario sin la necesidad de cargar nuevas páginas desde el servidor. La navegación (en el caso de existir) se simula mediante la reescritura de la URL gracias la api del navegador. La carga de contenido, e incluso de código asociado a la funcionalidad, se realiza de forma dinámica. Este cambio de paradigma respecto de la navegación web clásica implica que la experiencia de usuario sea más cercana a la de una aplicación de escritorio. La página no se recarga en ningún momento ni cede el control a otra página. La interacción con la aplicación de una sola página a menudo implica una comunicación dinámica con el servidor web de forma transparente al usuario (habitualmente mediante llamadas Ajax o Websockets).

Tests A/B

El test A/B es una herramienta muy utilizada para la optimización de aplicaciones. Su uso más habitual se da en aplicaciones web. Su forma de operar consiste en ofrecer dos versiones de la aplicación, A y B, al usuario y comparar empíricamente los resultados. Cada usuario solo ve una de las dos versiones, y el objetivo es

determinar con qué versión se opera de forma más óptima. Para ello previamente se establecen grupos de usuario.

La optimización de operatividad en el caso de la aplicaciones web va ligada habitualmente a tasas de conversión, es decir, al porcentaje de éxito en la finalización de procesos.

Transformada de Coseno Discreta (DCT)

Es una transformada basada en la Transformada de Fourier discreta, pero utilizando únicamente números reales. Se usa fundamentalmente para comprimir imágenes.

Existen variaciones a la hora de calcularla, pero lo más usual es:

$$f_j = \sum_{k=0}^{n-1} x_k \cos\left[\frac{\pi}{n} j \left(k + \frac{1}{2}\right)\right] + 0 \tag{8.1}$$

Transpilación (de código)

La transpilación es un caso particular de la compilación donde los lenguajes de origen y destino de la traducción se encuentran al mismo nivel de abstracción. La diferencia fundamental con la compilación es que esta última realiza la traducción a un nivel igual o menor de abstracción. Por ello todas las transpilaciones son compilaciones pero no viceversa. Un ejemplo muy claro y actual de compilación sería el paso de C a código binario y uno de transpilación sería la traducción de lenguaje «TypeScript», una variedad de Javascript que incluye tipado de datos.

UIDE (User Interface Design Environment)

El UIDE es un software que recopila todos los requisitos del producto y los organiza en áreas coherentes, cada una centrada en su intención principal, generando una interfaz gráfica a partir de componentes que permite realizar las funcionalidades descritas en los requisitos.

Tal y como se describe en (Foley & Sukaviriya, 1995), el UIDE modela los ob-

jetos de datos, acciones y atributos de una aplicación, y las condiciones previas y posteriores asociadas con las acciones. Luego, el modelo se utiliza para una variedad de servicios de tiempo de diseño y tiempo de ejecución, tales como: crear automáticamente ventanas, menús y paneles de control; evaluar el diseño en criterios de consistencia e integridad; controlar la ejecución de la aplicación; habilitar y deshabilitar elementos de menú y otros controles y hacer que las ventanas sean visibles e invisibles. Existen multitud de UIDE para diferentes aplicaciones y entornos, los más utilizados son las interfaces visuales para bases de datos.

User-friendly

El término **user-friendly** describe un dispositivo de hardware o una interfaz de software que es fácil de usar. Por fácil de usar es entiende que no es difícil de aprender o entender, es decir, que tiene los siguientes atributos:

- *Simple*. Una interfaz fácil de usar o bien no es demasiado compleja, o bien mantiene oculta con complejidad y proporciona un acceso rápido a funciones o comandos comunes.
- *Bien organizada*. Lo que facilita la localización de diferentes herramientas y opciones.
- *Intuitiva*. Toda representación de funcionalidad debe tener sentido para el usuario promedio y debe requerir una explicación mínima sobre cómo usarla.
- *Confortable*. Un producto que no aporte un alto grado de confiabilidad no es fácil de usar, ya que causará una frustración excesiva para el usuario, por tanto no debe funcionar mal o ni bloquearse.

Usuario conformista

Un usuario conformista es aquel usuario que no modifica sus hábitos esperados en función del diseño de la interfaz. Su capacidad de adaptación a la actividad

habitual depende de la adquisición de un conocimiento progresivo de las funcionalidades.

Usuario inquieto

Un usuario inquieto es aquel usuario que intenta readaptar la interfaz a sus propias inquietudes, que no han de coincidir necesariamente con sus necesidades. Puede llegar a modificar o ampliar las funcionalidades del sistema en función de sus intereses y conocimientos.

Anexo I: Describiendo interfaces a través de los estándares

En la actualidad uno de los sistemas de creación de interfaces más utilizados es la definición de éstas mediante un lenguaje de markup, ya sea HTML, JSX, n-templates (para páginas web) o XML (para aplicaciones Java para Android e Storyboards de iOS). Muchos entornos de desarrollo (IDE) van incorporando también estándares SGML para definir las interfaces para Rich Client Application en entornos de escritorio independientemente del lenguaje de desarrollo.

La versatilidad de los descriptores SGML es muy válida a la hora de trabajar con Entornos de Interfaz Adaptativos, puesto que es un soporte fantástico sobre el que añadir información a la interfaz, sin necesidad de crear otro esquema y asegurándose la utilización de un estándar.

No todos los elementos de una interfaz tendrán importancia dentro de un AIE, habrá elementos cuya representación se haya determinando intocable por distintas razones, a pesar de que para el usuario carezca de relevancia y es posible que deseara no lidiar con ellos en la interfaz. Por ello es importante que el lenguaje descriptor sea flexible y no excesivamente verboso.

A un AIE únicamente le hace falta saber la forma de interacción del usuario con los elementos de la interfaz y si existen límites a la hora de recomponerlo una vez se reconstruya la interfaz de acuerdo con las preferencias de este, es decir, el número de descriptores que se ha de añadir a la descripción de objetos de interfaz es muy pequeña. Se entiende también que como el AIE tiene acceso a las propiedades de representación de los elementos, éstas no deben estar necesariamente expuestas y pueden estar definidas aparte, por ejemplo en una guía de estilos.

Los elementos que deben describirse de un componente de interfaz, para que el AIE pueda realizar correctamente los cálculos son los siguientes:

Tipo	Valor
Identificador	Identifica que ese elemento es de interés para un AIE dentro de la interfaz. El hecho de emplear un nombre para el elemento y no utilizar un identificador autogenerado ayudará posteriormente a comprobar las evoluciones del sistema y la depuración del mismo.
Propiedades	Incluir un listado de las propiedades a las que deben afectar los cambios de pregnancia del elemento es muy importante. Por cada elemento de este listado debe existir un tipo componente asociado en el sistema que permita comunicarse el elemento físico y modificarlo convenientemente.
Mínimos / máximos	En aquellas propiedades que lo requieran deben poderse indicar los valores máximos y mínimos que pueden tomar a los recursos, en las medidas correspondientes.
Eventos	A igual que las propiedades, los eventos asociados a cada elemento deben ser un evento real asociado que sea capturado por un elemento AIE dependiendo del entorno y de la implementación de los AIE.

Cuadro 9.1: Elementos descriptivos de la interfaz

9.1. XML

XML es uno de los lenguajes de marcado más utilizados. Su uso principal es como formato de intercambio en Internet, esto se debe a su posibilidad de extensibilidad y su facilidad de descripción de su gramática.

XML y sus derivados permiten la definición tanto de nuevas etiquetas para describir elementos, como nuevas de propiedades para las etiquetas. Si dentro del documento XML (desde la versión 1.0) se hace referencia a un Document Type Definition (DTD), entonces la utilización de atributos propios puede hacerse más a medida.

El XML se utiliza también para describir las interfaces en proyectos para Android e iOS. En los entornos de programación más utilizados para estos sistemas (Android Studio y xCode) existe un Interface Builder cuyos archivos de intercambio utilizan XML aplicando cada uno distintos esquemas.

En los archivos de esquema (XSD) es donde se especifica la estructura del documento XML. Los nuevos elementos y propiedades que se definan para un AIE deben cumplir por tanto esas especificaciones, lo que conlleva soluciones distintas para cada plataforma. A continuación veremos cómo realizar un acercamiento al problema en plataformas android y en entornos web.

9.2. Plataformas Android

En plataformas android se utilizan por defecto las especificaciones aportadas por estándar de android *http://schemas.android.com/apk/res/android*. Android permite la definición de elementos y propiedades propios y propone definirlos en un archivo aparte, como por ejemplo: *http://schemas.android.com/apk/res/**aie***. Una vez definidos estos atributos personalizados, se pueden usar en los archivos XML de diseño al igual que los atributos integrados. La única diferencia es que sus atributos personalizados pertenecen a un espacio de nombres diferente. En lugar de pertenecer al espacio de nombres específico de android, pertenecen a ***aie***. Por ejemplo, en el código siguiente, se puede observar cómo usar estos atributos definidos para PieChart:

```
<?xml version="1.0" encoding="utf-8"?>
 <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:custom="http://schemas.android.com/apk/res/aie">
  <com.example.aie.charting.AIEPieChart
    custom:name="my-aie-chart" />
 </LinearLayout>
```

La definición de nuevos elementos conlleva replicar todos elementos a gráficos definidos en el estándar de Android, cada vez que se desee vincular un elemento de AIE a uno de ellos. La tarea resultaría muy árdua, y difícilmente mantenible (en cada versión de Android se podrían añadir nuevos componentes, lo que supondría una actualización continua).

La aproximación aquí propuesta pasa por definir un único componente, que admita como propiedades propias de los elementos AIE, que como elementos hijos pueda tener un único elemento, que será el elemento de la interfaz de Android al cual se vincularía.

Bajo esta premisa la definición del elemento AIE quedaría así:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  version="1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:aie="http://aie.org/schema/"
  xmlns:tools="http://schemas.android.com/tools"
  xmlns:android="http://schemas.android.com/apk/res/android">
  <xs:element name="AIEElement">
    <xs:complexType>
```

```
        <xs:sequence>
          <xs:element
            name="AIETiggers"
            type="AIETiggersType"
            minOccurs="0"
            maxOccurs="1"/>
          <xs:element
            name="AIEProperties"
            type="AIEPropertiesType"
            minOccurs="0"
            maxOccurs="1"/>
          <xs:any
            namespace="http://schemas.android.com/apk/res/android"
            minOccurs="1"
            processContents="lax"
            maxOccurs="1"/>
        </xs:sequence>
        <xs:attribute
          name="name"
          type="xs:string"/>
      </xs:complexType>
    </xs:element>

    <xs:complexType name="AIETiggersType">
      <xs:sequence>
        <xs:element
          name="AIETigger"
          type="AIETiggerType"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType
      name="AIETiggerType">
      <xs:attribute
        name="name"
        type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="AIEPropertiesType">
      <xs:sequence>
        <xs:element
          name="AIEProperty"
          type="AIEPropertyType"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="AIEPropertyType">
      <xs:attribute name="name" type="xs:string"/>
      <xs:attribute name="min" type="xs:string"/>
      <xs:attribute name="max" type="xs:string"/>
      <xs:attribute name="unit" type="xs:string"/>
    </xs:complexType>

  </xs:schema>
```

A continuación se puede observar como sería la definición de elementos dentro de una plantilla XML de Android. El ejemplo contiene una lista con una caja de texto y un botón:

```
  <?xml version="1.0" encoding="utf-8"?>
```

```
<AIEElement
  xmlns:aie="http://aie.org/schema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  name="example-layout">
  <AIETiggers>
    <AIETigger name="click"/>
    <AIETigger name="scroll"/>
  </AIETiggers>
  <LinearLayout
    xmlns="http://schemas.android.com/apk/res/android"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <AIEElement
      xmlns:aie="http://aie.org/schema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      name="example-button">
      <AIETiggers>
        <AIETigger name="click"/>
        <AIETigger name="scroll"/>
      </AIETiggers>
      <AIEProperties>
        <AIEProperty name="layout_width" max="200" min="100" unit="px" />
      </AIEProperties>
      <Button android:id="@+id/button"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Hello, I am a Button" />
    </AIEElement>
    <AIEElement
      xmlns:aie="http://aie.org/schema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      name="example-button">
      <AIETiggers>
        <AIETigger name="click"/>
      </AIETiggers>
      <AIEProperties>
        <AIEProperty name="font_size" min="6" max="20" unit="px" />
      </AIEProperties>
      <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    </AIEElement>
  </LinearLayout>
</AIEElement>
```

9.3. Entornos web

El estándar de HTML es mucho más flexible que el XML de Android, permitiendo añadir descriptores dentro de las etiquetas ya creadas. Por lo general el árbol de elementos que define el HTML de una web suele ser más grande que las vistas de Android, y los navegadores ofrecen una gran robustez a la hora de interpretarlo (permiten que haya etiquetas mal cerradas o errores de sintaxis a pesar de permitir la visualización). Por ello incluir más elementos que generan una complejidad del

árbol no merece la pena.

La aproximación propuesta a continuación se basa en la adición de propiedades específicas de los AIE a los componentes que lo requieran, sin necesidad de añadir elementos nuevos. Estas propiedades serán similares a las del XML: nombre, eventos y propiedades físicas a la pregnancia.

Se ha de considerar que en los entornos web se utilizan otras dos tecnologías además del lenguaje de marcado (HTML): las hojas de estilo en cascada (CSS) y el lenguaje de script (Javascript) tendrán también gran importancia en cómo hacer la adaptación.

En el entorno web el lenguaje en que se programa ya sea de manera directa o indirecta (transpilando el código) en JavaScript. Lo que conlleva que el manejo de eventos sobre cualquier elemento se hará a través de este lenguaje, a diferencia de en los entornos Android, en los que dependerá del lenguaje en que se programe (Kotlin, Java -nativos- o C# o .Net -con Xamarin-). Una de las ventajas que permite JavaScript con respecto a los eventos es que todos los nombres de eventos están definidos como pseudo-constantes (no podría decirse que son constantes, pero sí que se encuentran definidas dentro del estándar del lenguaje). Con lo cual a la hora de especificarlos en el HTML se pueden utilizar los mismos nombres que se utilizan habitualmente al programar la web, y es mucho más fácil interpretarlos.

Con CSS y las propiedades físicas resulta un caso similar al de Javascript con los eventos. No hay ninguna propiedad fuera del rango de las aceptadas por CSS que influya en la apariencia de una web. Por ello lo más lógico es utilizar el mismo nombre para las propiedades.

Se propone el nombre de *aie-properties* para las propiedades, *aie-triggers* para los eventos y *aie-name* para el identificador del evento.

En el ejemplo que se muestra a continuación, vemos cómo se define un ambiente formado por un elemento y un sub-ambiente que a su vez contiene dos elementos. Como se puede observar, el etiquetado para un AIE al definirse a través de atributos que no generan conflicto con los estándar permite convivir perfectamente con el etiquetado para el navegador, simplemente incluye una información extra que no está necesariamente indicada en el código HTML:

```
<ul aie-name="menu"
  aie-properties="font-size, position, level">
```

```
    <li>
      <label aie-name="item_1" aie-trigger="click">
        Menu Option 1
      </label>
    </li>
    <li>
      <label aie-name="item_2" aie-trigger="mouseover, click">
        Menu Option 2
      </label>
      <ul aie-name="submenu_1" aie-properties="position">
        <li>
          <label aie-name="item_2_1" aie-trigger="click">
            Menu Option 1
          </label>
        </li>
        <li>
          <label aie-name="item_2" aie-trigger="click">
            Menu Option 2
          </label>
        </li>
      </ul>
    </li>
  </ul>
```

Como se puede apreciar esta definición dista bastante de la Android, donde eventos y etiquetas tienen su propio árbol con un listado de elementos y aquí se incluyen como una cadena de elementos separados por comas.

Para definir máximos y mínimos de una forma más rápida se ha optado también por sugerir una nomenclatura abreviada que cumple la siguiente sintaxis:

$$<propiedad_1>|\ <max{:}valor\text{-}max_1>|\ <min{:}valor\text{-}min_1>, \ldots, <propiedad_n>|\ <max{:}valor\text{-}max_n>|\ <min{:}valor\text{-}min_n>$$

Los campos con ¿"serían opcionales

Lo que daría lugar a un código como el siguiente:

```
<div ... aie-properties="position,font-size|max:20px|min:10px">
```

Anexo II: Ejemplo de código

10.1. Describiendo una single-page application con AIE

A continuación se incluye el código fuente utilizado en el ejemplo de aplicación de un AIE a una single-page application.

```
<div id="single-page-application" aie-name="single-page-application-content">
 <div class="menu">
  <div class="block-title">Menu block</div>
  <div aie-properties="position" aie-name="menu">
   <div class="item" aie-trigger="click" aie-pregnancy="0.04"
    aie-name="menu_level_1">
    <label class="title" for="content1">Menu 1</label>
   </div>
   <div class="item" aie-trigger="click" aie-pregnancy="0.04"
    aie-name="menu_level_2">
    <label class="title" for="content2">Menu 2</label>
   </div>
   <div class="item" aie-trigger="click" aie-pregnancy="0.04"
    aie-name="menu_level_3">
    <label class="title" for="content3">Menu 3</label>
   </div>
   <div class="item" aie-trigger="click" aie-pregnancy="0.04"
    aie-name="menu_level_4">
    <label class="title" for="content4">Menu 4</label>
   </div>
  </div>
 </div>
 <div class="content">
  <input id="content1" type="radio" name="form_selector"
   class="hidden form_selector" checked>
  <div class="content-form">
   <div class="content-title">Content block 1: Location</div>
   <div class="form"
    aie-properties="position,height|min:150|total:500"
    aie-name="form_place" aie-free='0.09'>
    <fieldset aie-name="selector_group_place">
     <legend>Location config</legend>
     <div class="selector">
     <input id="select_input_2" readonly/>
      <div class="selector_options"
       aie-name="selector_place_fields"
       aie-properties="position,font-size|min:10|max:20|total:50">
       <div class="selector_option" aie-pregnancy="0.02"
        aie-name="selector_place_field_option_1"
```

```
        aie-trigger="click">Madrid</div>
       <div class="selector_option" aie-pregnancy="0.02"
        aie-name="selector_place_field_option_2"
        aie-trigger="click">Berlin</div>
       <div class="selector_option" aie-pregnancy="0.02"
        aie-name="selector_place_field_option_3"
        aie-trigger="click">Paris</div>
       <div class="selector_option" aie-pregnancy="0.02"
        aie-name="selector_place_field_option_4"
        aie-trigger="click">London</div>
      </div>
     </div>
    </fieldset>
    <fieldset aie-name="text_group_place">
      <legend>Location description</legend>
     <textarea aie-trigger="keypress" aie-pregnancy="0.05"
      aie-name="text_editor_field_location"
      placeholder="Location info"></textarea>
    </fieldset>
   </div>
  </div>

  <input id="content2" type="radio" name="form_selector"
   class="hidden form_selector">
  <div class="content-form">
   <div class="content-title">Content block 2: User info</div>
   <div class="form" aie-properties="position" aie-name="form">
    <fieldset aie-name="description_group"
     aie-properties="height|min:120, position">
     <legend>Long descriptions</legend>
     <textarea aie-trigger="keypress" aie-pregnancy="0.05"
      aie-name="text_editor_field_1"
      placeholder="Opinion..."></textarea>
     <textarea aie-trigger="keypress" aie-pregnancy="0.05"
      aie-name="text_editor_field_2"
      placeholder="Extra info..."></textarea>
     <textarea aie-trigger="keypress" aie-pregnancy="0.05"
      aie-name="text_editor_field_3"
      placeholder="Contact info..."></textarea>
    </fieldset>
    <fieldset aie-name="main_info_group">
     <legend>Main info</legend>
     <table width="600px">
      <tr aie-name="main_info_group_columns" width="400px"
       aie-properties="width|min:300">
       <td aie-name="main_info_group_fields" aie-properties="position">
        <div aie-name="input_field_username" aie-pregnancy="0.01"
        aie-trigger="keypress">
         <label>Username</label>
         <input class="large" type="text" placeholder="Mandatory" />
        </div>
        <div aie-name="input_field_phone" aie-pregnancy="0.01"
        aie-trigger="keypress">
         <label>Phone</label>
         <input class="large" type="phone" placeholder="Optional" />
        </div>
        <div aie-name="input_field_firstname" aie-pregnancy="0.01"
        aie-trigger="keypress">
         <label>First name</label>
         <input class="large" type="text" placeholder="Optional" />
        </div>
        <div aie-name="input_field_lastname" aie-pregnancy="0.01"
        aie-trigger="keypress">
         <label>Last name</label>
         <input class="large" type="text" placeholder="Optional" />
        </div>
        <div aie-name="input_field_email" aie-pregnancy="0.01"
        aie-trigger="keypress">
```

```
          <label>Email</label>
          <input class="large" type="email" placeholder="Mandatory" />
         </div>
        </td>
        <td aie-name="main_info_groupm_map" width="200px"
         aie-pregnancy="0.1" aie-trigger="mouseover" class="map"></td>
       </tr>
      </table>
     </fieldset>
     <fieldset aie-name="selector_group">
      <legend>Select Transport</legend>
      <div class="selector">
       <input id="select_input" readonly/>
       <div class="selector_options" aie-name="selector_fields"
         aie-properties="position,font-size|min:10|max:20|total:50">
        <div class="selector_option" aie-pregnancy="0.02"
         aie-name="selector_field_option_1" aie-trigger="click">Ship</div>
        <div class="selector_option" aie-pregnancy="0.02"
         aie-name="selector_field_option_2" aie-trigger="click">Train</div>
        <div class="selector_option" aie-pregnancy="0.02"
         aie-name="selector_field_option_3" aie-trigger="click">Plane</div>
        <div class="selector_option" aie-pregnancy="0.02"
         aie-name="selector_field_option_4" aie-trigger="click">Car</div>
       </div>
      </div>
     </fieldset>
     <fieldset aie-name="advanced_search_group" aie-properties="position">
      <legend>Search block</legend>
      <input aie-name="advanced_search_field" aie-pregnancy="0.025"
       aie-trigger="keypress" type="text" placeholder="Text to search" />
      <button aie-name="advanced_search_button" aie-pregnancy="0.025"
       aie-trigger="click">Find now!</button>
     </fieldset>
    </div>
   </div>
   <input id="content3" type="radio"
    name="form_selector"
    class="hidden form_selector">
   <div class="content-form">
    <div class="content-title">Content block 3: Wallet</div>
    <div class="form"
     aie-properties="position,height|min:100|total:300"
     aie-name="form_wallet" aie-free='0.09'>
     <div aie-name="advanced_wallet_group"
      aie-properties="width|min:100|total:500" style="height:200px">
      <fieldset>
       <legend>Insert money</legend>
       <input aie-name="advanced_wallet_field"
        aie-pregnancy="0.025" aie-trigger="keypress" type="text"
        placeholder="Text to search" style="width:200px" />
       <button aie-name="advanced_wallet_button"
        aie-pregnancy="0.025"
        aie-trigger="click"
        style="height:200px" >Add mone now!</button>
      </fieldset>
     </div>
      <div aie-name="main_info_group_fields_wallet"
      aie-properties="position" style="height:200px">
       <div
        aie-name="input_field_username_wallet"
        aie-pregnancy="0.025" aie-trigger="keypress">
        <label>Account number</label>
        <input class="large" type="text" placeholder="Mandatory" />
       </div>
       <div aie-name="input_field_phone_wallet"
        aie-pregnancy="0.025" aie-trigger="keypress">
        <label>Bank</label>
        <input class="large" type="phone"
```

```
          placeholder="Optional" />
        </div>
       </div>
     </div>
    </div>
    <input id="content4" type="radio"
     name="form_selector" class="hidden form_selector">
    <div class="content-form">
     <div class="content-title">Content block 4: Legal info</div>
     <div class="form">
      <h1>Legal advise</h1>
      <p>....</p>
     </div>
    </div>
   </div>
  </div>
```

Listado de figuras

Listado de tablas

Bibliografía

Adami, C. (2006). Digital genetics: unravelling the genetic basis of evolution. *Nature Reviews Genetics*, *7*(2), 109-118. doi:10.1038/nrg1771

Aleixo, F. A., Freire, M., Alencar, D., Campos, E. & Kulesza, U. (2012). A Comparative Study of Compositional and Annotative Modelling Approaches for Software Process Lines. En *2012 26th Brazilian Symposium on Software Engineering*. doi:10.1109/sbes.2012.11

Ali, M. F., Perez-Quinones, M. A., Shell, E. & Abrams, M. (2001). Building Multi-Platform User Interfaces with UIML. arXiv: cs/0111024v1 [cs.HC]

Apel, S., Lengauer, C., Möller, B. & Kästner, C. (2008). An Algebra for Features and Feature Composition. En *Algebraic Methodology and Software Technology* (pp. 36-50). doi:10.1007/978-3-540-79980-1_4

Arheim, R. (2005). *Arte y percepción visual: psicología del ojo creador* (A. Editorial, Ed.).

Avdiienko, V., Kuznetsov, K., Rommelfanger, I., Rau, A., Gorla, A. & Zeller, A. (2017). Detecting Behavior Anomalies in Graphical User Interfaces. En *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. doi:10.1109/icse-c.2017.130

Balaban, S. (2019). Deep learning and face recognition: the state of the art. *Proc. SPIE 9457, Biometric and Surveillance Technology for Human and Activity Identification XII, 94570B (15 May 2015)*. doi:10.1117/12.2181526. arXiv: 1902.03524v1

Balzert, H. (1995). From OOA to GUI — The Janus System. En *IFIP Advances in Information and Communication Technology* (pp. 319-324). doi:10.1007/978-1-5041-2896-4_53

Banfield, R., Lombardo, T. & Wax, T. (2015). *esign sprint: a practical guidebook for building great digital products* (I. O'Reilly Media, Ed.).

Barthel, M. & Shearer, E. (2015). *How do Americans use Twitter for news?* Pew Research Center.

Batory, D. [D.], Sarvela, J. & Rauschmayer, A. (2003). Scaling step-wise refinement. En *25th International Conference on Software Engineering, 2003. Proceedings.* doi:10.1109/icse.2003.1201199

Batory, D. [D.], Sarvela, J. & Rauschmayer, A. (2004). Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, *30*(6), 355-371. doi:10.1109/tse.2004.23

Batory, D. [Don]. (2005). Feature Models, Grammars, and Propositional Formulas. En *Software Product Lines* (pp. 7-20). doi:10.1007/11554844_3

Batory, D. [Don]. (2006). A Tutorial on Feature Oriented Programming and the AHEAD Tool Suite. En *Generative and Transformational Techniques in Software Engineering* (pp. 3-35). doi:10.1007/11877028_1

Behrens, R. R. (1998). Art, Design and Gestalt Theory. *Leonardo*, *31*(4), 299. doi:10.2307/1576669

Bentley, J. E. (1999). 14 Steps to a Good GUI. En *In SAS® Users Group International Conference* (Vol. 24).

Bodart, F., Hennebert, A.-M., Leheureux, J.-M., Provot, I., Sacré, B. & Vanderdonckt, J. (1995). Towards a Systematic Building of Software Architecture: the TRIDENT Methodological Guide. En *Eurographics* (pp. 262-278). doi:10.1007/978-3-7091-9437-9_16

Bodart, F. & Vanderdonckt, J. M. (1993). Expressing guidelines into an ergonomical styleguide for highly interactive applications. En *INTERACT '93 and CHI '93 conference companion on Human factors in computing systems*. doi:10.1145/259964.260036

Breivold, H. P., Crnkovic, I., Land, R. & Larsson, S. (2008). Using dependency model to support software architecture evolution. En *2008 23rd IEEE/ACM*

International Conference on Automated Software Engineering - Workshops. doi:10.1109/asew.2008.4686324

Brejcha, J. (2015). *Cross-Cultural Human-Computer Interaction and User Experience Design: A Semiotic Perspective*. doi:10.1201/b18059

Brown, T. & Katz, B. (2011). Change by Design: Change by Design. *Journal of Product Innovation Management*, *28*(3), 381-383. doi:10.1111/j.1540-5885.2011.00806.x

Bundesen, C., Vangkilde, S. & Petersen, A. (2015). Recent developments in a computational theory of visual attention (TVA). *Vision Research*, *116*, 210-218. doi:10.1016/j.visres.2014.11.005

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. & Vanderdonckt, J. (2003). A Unifying Reference Framework for multi-target user interfaces. *Interacting with Computers*, *15*(3), 289-308. doi:10.1016/s0953-5438(03)00010-9

Carroll, J. M. (1988). *Evaluation, Description and Invention: Paradigms for Human-Computer Interaction*. Defense Technical Information Center. doi:10.21236/ada204617

Cavarlé, G., Plantec, A., Costiou, S. & Ribaud, V. (2016). Dynamic Round-Trip Engineering in the context of FOMDD. En *Proceedings of the 11th edition of the International Workshop on Smalltalk Technologies - IWST'16*. doi:10.1145/2991041.2991056

Chong, A. Y.-L. (2013). A two-staged SEM-neural network approach for understanding and predicting the determinants of m-commerce adoption. *Expert Systems with Applications*, *40*(4), 1240-1247. doi:10.1016/j.eswa.2012.08.067

Ciurana, E. (2008). The design of a Google App engine application. En *Developing with Google App Engine* (pp. 33-40). doi:10.1007/978-1-4302-1832-6_3

Clement, J. (2019). Number of apps available in leading app stores as of 3rd quarter 2019. Recuperado desde https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

Coughlan, P. & Coghlan, D. (2002). Action research for operations management. *International Journal of Operations Production Management*, *22*(2), 220-240. doi:10.1108/01443570210417515

Czarnecki, K. (2002). Generative Programming: Methods, Techniques, and Applications Tutorial Abstract. En *Software Reuse: Methods, Techniques, and Tools* (pp. 351-352). doi:10.1007/3-540-46020-9_38

da Silva, A. R., da Silva, M. M. & Romão, A. (2000). Web-Based Agent Applications: User Interfaces and Mobile Agents. En *Telecommunications and IT Convergence Towards Service E-volution* (pp. 135-153). doi:10.1007/3-540-46525-1_10

da Silva, P. P. (2001). User Interface Declarative Models and Development Environments: A Survey. En *Interactive Systems Design, Specification, and Verification* (pp. 207-226). doi:10.1007/3-540-44675-3_13

Davidson, D. (2002). Dealers of lightning: xerox PARC and the dawn of the computer age [Book Review]. *IEEE Antennas and Propagation Magazine*, *44*(6), 108-108. doi:10.1109/map.2002.1167269

de Oliveira, L. R., Medeiros, R. M., de Bragança Terra, P. & Quelhas, O. L. G. (2011). Sustentabilidade: da evolução dos conceitos à implementação como estratégia nas organizações. *Production*, *22*(1), 70-82. doi:10.1590/s0103-65132011005000062

Delgado, A. [A.], Estepa, A., Troyano, J. & Estepa, R. (2016). Reusing UI elements with Model-Based User Interface Development. *International Journal of Human-Computer Studies*, *86*, 48-62. doi:10.1016/j.ijhcs.2015.09.003

Delgado, A. [Antonio], Estepa, A., Troyano, J. A. & Estepa, R. (2009). On the Reusability of User Interface Declarative Models. En *Computer-Aided Design of User Interfaces VI* (pp. 313-318). doi:10.1007/978-1-84882-206-1_29

Deterding, S., Dixon, D., Khaled, R. & Nacke, L. (2011). From game design elements to gamefulness: defining "gamification". En *Proceedings of the 15th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '11*. doi:10.1145/2181037.2181040

Dillon, A. & Watson, C. (1996). User analysis in HCI — the historical lessons from individual differences research. *International Journal of Human-Computer Studies*, *45*(6), 619-637. doi:10.1006/ijhc.1996.0071

DiStefano, B. & DiStefano, E. (1989). Doing real time. *IEEE Potentials*, *8*(3), 40-43. doi:10.1109/45.41536

Eden, C. & Ackermann, F. (2018). Theory into practice, practice to theory: Action research in method development. *European Journal of Operational Research*, *271*(3), 1145-1155. doi:10.1016/j.ejor.2018.05.061

Ehlert, P. (2003). *Intelligent User Interfaces: Introduction and Survey*. Delft University of Technology.

Elkan, C. (2001). Paradoxes of fuzzy logic, revisited. *International Journal of Approximate Reasoning*, *26*(2), 153-155. doi:10.1016/s0888-613x(00)00065-7

Engelbart, D. C. (1970). *Computer-Augmented ManagementSystem Research and Development of augmentation facility*. Defense Technical Information Center. doi:10.21236/ad0709211

Española, R. A. (2005). *Diccionario panhispánico de dudas* (Santillana, Ed.).

Fabijan, A., Olsson, H. H. & Bosch, J. (2016). Time to Say 'Good Bye': Feature Lifecycle. En *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. doi:10.1109/seaa.2016.59

Feng, J. & Liu, Y. (2015). Intelligent Context-Aware and Adaptive Interface for Mobile LBS. *Computational Intelligence and Neuroscience*, *2015*, 1-10. doi:10.1155/2015/489793

Ferreira, J., Sharp, H. & Robinson, H. (2012). Agile development and user experience design integration as an ongoing achievement in practice. En *2012 Agile Conference*. doi:10.1109/agile.2012.33

Florio, V. D. (2014). Behavior, Organization, Substance: Three Gestalts of General Systems Theory. doi:10.1109/NORBERT.2014.6893923. arXiv: 1403.4077v2 [cs.OH]

Foley, J. D. (1998). The Convergence of Graphics and Imaging. *Computer Graphics Forum*, *17*(3), xviii-xviii. doi:10.1111/1467-8659.00246

Foley, J. D. & Sukaviriya, P. (1995). History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Implementation. En *Interactive Systems: Design, Specification, and Verification* (pp. 3-14). doi:10.1007/978-3-642-87115-3_1

Fraser, S., Reinitz, R., Eckstein, J., Kerievsky, J., Mee, R. & Poppendieck, M. (2003). Xtreme programming and agile coaching. En *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA '03*. doi:10.1145/949404.949406

Frawley, W. J., Piatetsky-Shapiro, G. & Matheus, C. J. (1992). Knowledge Discovery in Databases: An Overview. *AI Magazine*, *13*, 57-70. doi:doi.org/10.1609/aimag.v13i3.1011

Freeman, R. B. (1977). *On the Origin of Species", The Works of Charles Darwin: An Annotated Bibliographical* (A. Books, Ed.).

Fukuda, T., Arai, F., Yamamoto, Y., Naito, T. & Matsui, T. (1994). Concept and Realization of Interactive Adaptive Interface Using Recursive Fuzzy Inference. *Transactions of the Japan Society of Mechanical Engineers Series C*, *60*(571), 970-977. doi:10.1299/kikaic.60.970

Gao, T. [Tingting], Liu, Y.-J., Liu, L. & Li, D. (2018). Adaptive neural network-based control for a class of nonlinear pure-feedback systems with time-varying full state constraints. *IEEE/CAA Journal of Automatica Sinica*, *5*(5), 923-933. doi:10.1109/jas.2018.7511195

Gao, T. [Tong], Dontcheva, M., Adar, E., Liu, Z. & Karahalios, K. G. (2015). DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. En *Proceedings of the 28th Annual ACM Symposium on User Interface Software Technology - UIST '15*. doi:10.1145/2807442.2807478

German, E. & Sheremetov, L. (2008). Specifying Interaction Space Components in a FIPA-ACL Interaction Framework. En *Languages, Methodologies and Development Tools for Multi-Agent Systems* (pp. 191-208). doi:10.1007/978-3-540-85058-8_12

González, C. (2014). *Videojuegos para la transformación social Aportaciones conceptuales y metodológicas* (Universidad de Deusto).

Gorte, B. & Sithole, G. (2012). Lookup Table Hough Transform for Real Time Range Image Segmentation and Featureless Co-Registration. *Journal of Sensor Technology*, *02*(03), 148-154. doi:10.4236/jst.2012.23021

Gould, J. D. & Lewis, C. (1985). Designing for usability: key principles and what designers think. *Communications of the ACM*, *28*(3), 300-311. doi:10.1145/3166.3170

Grant, C. M. (2001). *Greenaway, Peter*. doi:10.1093/gao/9781884446054.article.t097250

Guerrero-Garcia, J., Gonzalez-Calleros, J. M., Vanderdonckt, J. & Munoz-Arteaga, J. (2009). A Theoretical Survey of User Interface Description Languages:

Preliminary Results. En *2009 Latin American Web Congress*. doi:10.1109/la-web.2009.40

Guger, C., Allison, B. Z. & Müller-Putz, G. R. (2015). Brain-Computer Interface Research: A State-of-the-Art Summary 4. En *SpringerBriefs in Electrical and Computer Engineering* (pp. 1-8). doi:10.1007/978-3-319-25190-5_1

Halder, A., Jati, A., Singh, G., Konar, A., Chakraborty, A. & Janarthanan, R. (2012). Facial Action Point Based Emotion Recognition by Principal Component Analysis. En *Advances in Intelligent and Soft Computing* (pp. 721-733). doi:10.1007/978-81-322-0491-6_66

Hammant, P. (2013). What is Trunk Based Development? Recuperado desde https://paulhammant.com/2013/04/05/what-is-trunk-based-development/

Harris, R. A. (2005). Introduction. En *Voice Interaction Design* (pp. 3-31). doi:10.1016/b978-155860768-2/50001-3

Hassenzahl, M. (2008). User experience (UX): towards an experiential perspective on product quality. En *Proceedings of the 20th International Conference of the Association Francophone d'Interaction Homme-Machine on - IHM '08*. doi:10.1145/1512714.1512717

Highsmith, J. & Cockburn, A. (2001). Agile software development: the business of innovation. *Computer*, *34*(9), 120-127. doi:10.1109/2.947100

Hodgson, P. (2016). Feature Toggles. Recuperado desde http%20:%20/%20/%20martinfowler%20.%20com%20/%20articles%20/%20feature-toggles.htm

Huhns, M. & Singh, M. (1997). Agents on the Web. *IEEE Internet Computing*, *1*(5), 78-79. doi:10.1109/4236.623972

Hume, C., Wilhere, G., Stanley, S., Grigsby, S. & Slattery, K. (15). *Watershed Characterization for WRIA 7*. Washington State Department of Ecology.

Ingham, J. (1997). *What is an Agent?* Centre for Software Maintenance. University of Durham.

Isensee, S., Roberts, D., Berry, D. & Mullaly, J. (1998). User Interface Design with OVID. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, *42*(3), 310-314. doi:10.1177/154193129804200327

JA, C. [Capitán] & JA, C. [Cuesta]. (2011). Species assembly in model ecosystems, I: Analysis of the population model and the invasion dynamics. *Journal of theoretical biology*, *269*, 330-43. doi:10.1016/j.jtbi.2010.09.032

Jacobson, I., Booch, G. & Rumbaugh, J. (1999). *The Unified Software Development Process* (A.-W. L. P. Co, Ed.).

Jaquero, V. L., Montero, F., Molina, J. & Gonzlez, P. (2008). Intelligent User Interfaces: Past, Present and Future. En *Engineering the User Interface* (pp. 1-12). doi:10.1007/978-1-84800-136-7_18

Jardim, N. & Falcão, J. (2001). Wisdom — A UML Based Architecture for Interactive Systems. En *Interactive Systems Design, Specification, and Verification* (pp. 191-205). doi:10.1007/3-540-44675-3_12

Jones, T. (2012). Linux kernel co-scheduling and bulk synchronous parallelism. *The International Journal of High Performance Computing Applications*, *26*(2), 136-145. doi:10.1177/1094342011433523

Joshi, P. (2015). Contextually proximate approach to develop smart user interface. arXiv: 1508.04131v1 [cs.HC]

Junior, P. T. A. & Filgueiras, L. V. L. (2005). User modeling with personas. En *Proceedings of the 2005 Latin American conference on Human-computer interaction - CLIHC '05*. doi:10.1145/1111360.1111388

Kannizza, G. (1987). *Gramática de la visión: percepción y pensamiento* (P. Ibérica, Ed.).

Kashfi, P., Feldt, R., Nilsson, A. & Svensson, R. B. (2014). A Conceptual UX-aware Model of Requirements. arXiv: 1409.4194v2 [cs.SE]

Kästner, C. & Apel, S. (2013). Feature-Oriented Software Development. En *Lecture Notes in Computer Science* (pp. 346-382). doi:10.1007/978-3-642-35992-7_10

Katz, D. (1967). *Psicología de la forma* (Espaza-Calpe, Ed.).

Kawahara, Y., Morikawa, H. & Aoyama, T. (2007). Intelligent Interface Systems Utilizing User Context-Awareness. En *2007 International Conference on Machine Learning and Cybernetics*. doi:10.1109/icmlc.2007.4370494

Kharat, G. U. & Dudul, S. V. (2008). Neural Network Classifier for Human Emotion Recognition from Facial Expressions Using Discrete Cosine Transform.

En *2008 First International Conference on Emerging Trends in Engineering and Technology*. doi:10.1109/icetet.2008.22

Kiniry, J. & Zimmerman, D. (1997). A hands-on look at Java mobile agents. *IEEE Internet Computing*, *1*(4), 21-30. doi:10.1109/4236.612210

Kirisci, P. T. & Thoben, K.-D. (2018). A Method for Designing Physical User Interfaces for Intelligent Production Environments. *Advances in Human-Computer Interaction*, *2018*, 1-21. doi:10.1155/2018/6487070

Knight, W. (2019). *UX for developers* (Apress, Ed.).

Koch, N. (2001). *Software engineering for adaptive hypermedia systems: Reference model, modelling techniques and development process* (Universität Munchen).

Kosch, T., Funk, M., Schmidt, A. & Chuang, L. L. (2018). Identifying Cognitive Assistance with Mobile Electroencephalography: A Case Study with In-Situ Projections for Manual Assembly. *Proceedings of the ACM on Human-Computer Interaction*, *2*(EICS), 1-20. doi:10.1145/3229093

Kossiakoff, A. & Sweet, W. (2005a). Software Systems Engineering. En *Systems Engineering Principles and Practice* (pp. 361-408). doi:10.1002/0471723630.ch13

Kossiakoff, A. & Sweet, W. (2005b). Systems Engineering Management. En *Systems Engineering Principles and Practice* (pp. 90-116). doi:10.1002/0471723630.ch4

Kuznetsov, K., Avdiienko, V., Gorla, A. & Zeller, A. (2018). Analyzing the user interface of Android apps. En *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems - MOBILESoft '18*. doi:10.1145/3197231.3197232

Labrou, Y. & Finin, T. (1998). Semantics and Conversations for an Agent Communication Language. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97) August, 1997*. arXiv: cs/9809034v1 [cs.MA]

Lappin, J. S., Seiffert, A. E. & Bell, H. H. (2020). A Limiting Channel Capacity of Visual Perception: Spreading Attention Divides the Rates of Perceptual Processes. *Attention, Perception Psychophysics*. doi:10.3758/s13414-020-01973-9

Lauren, D., Arnaud, B., Paul, K., Hervé, G. & Olivier, M. (2018). Using Machine Learning Algorithms to Develop Adaptive Man-Machine Interfaces. En *Neuroergonomics* (pp. 237-238). doi:10.1016/b978-0-12-811926-6.00053-1

Law, E. L.-C., van Schaik, P. & Roto, V. (2014). Attitudes towards user experience (UX) measurement. *International Journal of Human-Computer Studies*, *72*(6), 526-541. doi:10.1016/j.ijhcs.2013.09.006

Lee, M. C. & Chang, T. (2010). Developing a lean design for Six Sigma through supply chain methodology. *International Journal of Productivity and Quality Management*, *6*(4), 407. doi:10.1504/ijpqm.2010.035891

Lee, N. Y., Kim, Y. & Sang, Y. (2017). How do journalists leverage Twitter? Expressive and consumptive use of Twitter. *The Social Science Journal*, *54*(2), 139-147. doi:10.1016/j.soscij.2016.09.004

Leichtenstern, K. & Andre, E. (2008). User-Centred Development of Mobile Interfaces to a Pervasive Computing Environment. En *First International Conference on Advances in Computer-Human Interaction*. doi:10.1109/achi.2008.10

Lennarz, H. (2017). AB-Testing. En *Growth Hacking mit Strategie* (pp. 73-79). doi:10.1007/978-3-658-16231-3_4

Leonardo, O. G. (2004). La definición del concepto de percepción en la psicología con base la teoría Gestalt. *Revista de Estudios Sociales*, (18), 89-96. doi:10.7440/res18.2004.08

Lewin, K. (1946). Action research and minority problems. *Journal of Social Issues*, *2*(4), 34-46. doi:10.1111/j.1540-4560.1946.tb02295.x

Liu, B., Hsu, W., Mun, L.-F. & Lee, H.-Y. (1999). Finding interesting patterns using user expectations. *IEEE Transactions on Knowledge and Data Engineering*, *11*, 817-832. doi:10.1109/69.824588

Locke, J. (1690). An Essay concerning Human Understanding. En P. H. Nidditch (Ed.), *The Clarendon Edition of the Works of John Locke: An Essay Concerning Human Understanding* (pp. 1-1). doi:10.1093/oseo/instance.00018020

Lowe, D. (1999). Object recognition from local scale-invariant features. En *Proceedings of the Seventh IEEE International Conference on Computer Vision*. doi:10.1109/iccv.1999.790410

Magues, D. A., Castro, J. W. & Acuna, S. T. (2016). Usability in agile development: A systematic mapping study. En *2016 XLII Latin American Computing Conference (CLEI)*. doi:10.1109/clei.2016.7833347

Malsburg, C. V. D. (1986). Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. En *Brain Theory* (pp. 245-248). doi:10.1007/978-3-642-70911-1_20

Martin-Hammond, A., Hamidi, F., Bhalerao, T., Ortega, C., Ali, A., Hornback, C., … Hurst, A. (2018). Designing an Adaptive Web Navigation Interface for Users with Variable Pointing Performance. En *Proceedings of the Internet of Accessible Things on W4A '18*. doi:10.1145/3192714.3192818

Matharu, G. S., Mishra, A., Singh, H. & Upadhyay, P. (2015). Empirical Study of Agile Software Development Methodologies: A Comparative Analysis. *ACM SIGSOFT Software Engineering Notes*, *40*(1), 1-6. doi:10.1145/2693208.2693233

Matthias, S. (2006). A Taxonomy for Semi-Supervised Learning Methods. En *Semi-Supervised Learning* (pp. 14-31). doi:10.7551/mitpress/9780262033589.003.0002

May, B. (2012). Applying lean startup: An experience report. En *2012 Agile Conference*. doi:10.1109/agile.2012.18

Mayer, R. C., Davis, J. H. & Schoorman, F. D. (1995). An Integrative Model of Organizational Trust. *The Academy of Management Review*, *20*(3), 709. doi:10.2307/258792

Meinel, C. & Leifer, L. (2011). Design Thinking Research. En *Design Thinking Research* (pp. 1-11). doi:10.1007/978-3-642-21643-5_1

Meixner, G., Paternò, F. & Vanderdonckt, J. (2011). Past, Present, and Future of Model-Based User Interface Development. *i-com*, *10*(3), 2-11. doi:10.1524/icom.2011.0026

Miller, S. (2014). The data from our UX industry survey is in! Recuperado desde https://www.usertesting.com/blog/the-data-from-our-ux-industry-survey-is-in

Mori, G., Paternò, F. & Santoro, C. (2002). CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, *28*(8), 797-813. doi:10.1109/tse.2002.1027801

Morrison, P., Holmgreen, C., Massey, A. & Williams, L. (2013). Proposing regulatory-driven automated test suites. En *2013 Agile Conference*. doi:10.1109/agile.2013.8

Myers, B. A. (1998). A brief history of human-computer interaction technology. *interactions*, *5*(2), 44-54. doi:10.1145/274430.274436

N, K., J, H. & M, A. (2010). Perceptual guidelines for creating rectangular treemaps. *IEEE transactions on visualization and computer graphics*, *16*, 990-8. doi:10.1109/TVCG.2010.186

NaumannIna, A. B., Wechsung, I. & Schleicher, R. (2009). Measurements and Concepts of Usability and User Experience: Differences between Industry and Academia. En H. Springer Berlin (Ed.), *Human Centered Design 2009*, (Vol. 5619). doi:10.1007/978-3-642-02806-9_72

Nielsen, J. (1993). Iterative user-interface design. *Computer*, *26*(11), 32-41. doi:10.1109/2.241424

Nielsen, J. & Farrell, S. (2014). User Experience Career Advice From 1,015 UX Professionals. Recuperado desde https://www.nngroup.com/articles/ux-career-advice

Nivethika, M., Vithiya, I., Anntharshika, S. & Deegalla, S. (2013). Personalized and adaptive user interface framework for mobile application. En *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. doi:10.1109/icacci.2013.6637474

Nunes, N. J. (2001). *Object modeling for user-centered development and user interface design: the wisdom approach* (Universidade da Madeira).

Okada, N., Inui, K. & Tokuhisa, M. (1999). delete Towards affective integration of vision, behavior, and speech processing. En IEEE (Ed.), *Proceedings Integration of Speech and Image Understanding*, doi:10.1109/ISIU.1999.824850

Oldenburger, R. (1963). *Adaptive and self-optimizing control*. Defense Technical Information Center. doi:10.21236/ad0295740

Orr, H. A. (2005). The genetic theory of adaptation: a brief history. *Nature Reviews Genetics*, *6*(2), 119-127. doi:10.1038/nrg1523

Ospina, O. M. S., Marín, P. A. R., Carranza, D. A. O. & Méndez, N. D. D. (2017). Interfaces adaptativas personalizadas para brindar recomendaciones en re-

positorios de objetos de aprendizaje. *Tecnura*, *21*(53), 107-118. doi:10.14483/22487638.9287

Palmer, S. R. & Felsing, M. (2001). *A Practical Guide to Feature-Driven Development*.

Palonka, J., Lora, M. & Oziębło, T. (2017). Google Analytics as a Prosumption Tool for Web Analytics. En *The Analytics Process* (pp. 127-144). doi:10.1201/9781315161501-6

Paternò, F. (2000). The ConcurTaskTrees Notation. En *Model-Based Design and Evaluation of Interactive Applications* (pp. 39-66). doi:10.1007/978-1-4471-0445-2_4

Paul, G. (2012). As origens da ideia da forma: (capítulo I - La Psychologie de la Forme1). *Phenomenologial studies - Revista da Abordagem Gestáltica*, *18*(1), 107-113. doi:10.18065/rag.2012v18n1.13

Poslad, S. (2009). *Ubiquitous computing*. doi:10.1002/9780470779446

Prehofer, C. (1997). Feature-oriented programming: A fresh look at objects. En *ECOOP'97 — Object-Oriented Programming* (pp. 419-443). doi:10.1007/bfb0053389

Rahman, M. T., Querel, L.-P., Rigby, P. C. & Adams, B. (2016). Feature toggles: practitioner practices and a case study. En *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16*. doi:10.1145/2901739.2901745

Rao, J., Ture, F., He, H., Jojic, O. & Lin, J. (2017). Talking to Your TV: Context-Aware Voice Search with Hierarchical Recurrent Neural Networks. arXiv: 1705.04892v1 [cs.IR]

Ratchford, B. (2019). The impact of digital innovations on marketing and consumers. *Marketing in a Digital World (Review of Marketing Research, Vol. 16)*, *16*, 35-61. doi:10.1108/s1548-6435201916

Rodriguez, P., Tabares, V., Duque, N., Ovalle, D. & Vicari, R. (2013). BROA: An agent-based model to recommend relevant Learning Objects from Repository Federations adapted to learner profile. *International Journal of Interactive Multimedia and Artificial Intelligence*, *2*(1), 6. doi:10.9781/ijimai.2013.211

Rousseau, D. M., Sitkin, S. B., Burt, R. S. & Camerer, C. (1998). Not So Different After All: A Cross-Discipline View Of Trust. *Academy of Management Review*, *23*(3), 393-404. doi:10.5465/amr.1998.926617

Rusu, C., Rusu, V. & Roncagliolo, S. (2008). Usability practice: the appealing way to HCI. En *First International Conference on Advances in Computer-Human Interaction*. doi:10.1109/achi.2008.14

Schermann, G. (2017). Continuous experimentation for software developers. En *Proceedings of the 18th Doctoral Symposium of the 18th International Middleware Conference on - Middleware '17*. doi:10.1145/3152688.3152691

Schittenkopf, C., Deco, G. & Brauer, W. (1997). Two Strategies to Avoid Overfitting in Feedforward Networks. *Neural Networks*, *10*(3), 505-516. doi:10.1016/s0893-6080(96)00086-x

Schwaber, K. (1997). SCRUM Development Process. En *Business Object Design and Implementation* (pp. 117-134). doi:10.1007/978-1-4471-0947-1_11

Seaborn, K. & Fels, D. I. (2015). Gamification in theory and action: A survey. *International Journal of Human-Computer Studies*, *74*, 14-31. doi:10.1016/j.ijhcs.2014.09.006

Sebek, J., Trnka, M. & Cerny, T. (2015). On Aspect-Oriented Programming in Adaptive User Interfaces. En *2015 2nd International Conference on Information Science and Security (ICISS)*. doi:10.1109/icissec.2015.7371024

Shamma, J. (2007). *Cooperative Control of Distributed Multi-Agent Systems: Shamma/Cooperative Control of Distributed Multi-Agent Systems* (J. S. Shamma, Ed.). doi:10.1002/9780470724200

Sharma, S. K., Govindaluri, S. M. & Balushi, S. M. A. (2015). Predicting determinants of Internet banking adoption: A two-staged regression-neural network approach. *Management Research Review*, *38*(7), 750-766. doi:10.1108/mrr-06-2014-0139

Shin, D.-H. & Biocca, F. (2017). Explicating user behavior toward multi-screen adoption and diffusion: User experience in the multi-screen media ecology. *Internet Research*, *27*(2), 338-361. doi:10.1108/intr-12-2015-0334

Smyth, B., McGinty, L., Reilly, J. & McCarthy, K. (2004). Compound Critiques for Conversational Recommender Systems. En *IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*. doi:10.1109/wi.2004.10098

Sole, A. D. (2018). Source Control with Git. En *Visual Studio Code Distilled* (pp. 125-155). doi:10.1007/978-1-4842-4224-7_7

Sourin, A., Earnshaw, R., Gavrilova, M. & Sourina, O. (2016). Problems of Human-Computer Interaction in Cyberworlds. En *Lecture Notes in Computer Science* (pp. 1-22). doi:10.1007/978-3-662-53090-0_1

Stapleton, J. (2003). *Business Focused Development* (P. Education, Ed.).

Steichen, B., Ashman, H. & Wade, V. (2012). A comparative survey of Personalised Information Retrieval and Adaptive Hypermedia techniques. *Information Processing Management*, *48*(4), 698-724. doi:10.1016/j.ipm.2011.12.004

Stull, E. (2018). Waterfall, Agile, and Lean. En *UX fundamentals for non-UX professionals* (pp. 209-219). doi:10.1007/978-1-4842-3811-0_30

Sulaiman, S. & Sohaimi, I. S. (2010). An investigation to obtain a simple mobile phone interface for older adults. En *2010 International Conference on Intelligent and Advanced Systems*. doi:10.1109/icias.2010.5716254

Tabassum, M. & Mathew, K. (2014). Software evolution analysis of linux (Ubuntu) OS. En *2014 International Conference on Computational Science and Technology (ICCST)*. doi:10.1109/iccst.2014.7045194

Teixes, F. (2014). *Gamificación: fundamentos y aplicaciones* (O. O. Publishing, Ed.).

Trifilio, S. (2014). A guide to UX careers. Recuperado desde http://www.onwardsearch.com/UX-Career-Guide

Trujillo, S., Batory, D. & Diaz, O. (2007). Feature Oriented Model Driven Development: A Case Study for Portlets. En *29th International Conference on Software Engineering (ICSE'07)*. doi:10.1109/icse.2007.36

Uruchrutu, E., MacKinnon, L. & Rist, R. (2005). User Cognitive Style and Interface Design for Personal, Adaptive Learning. What to Model? En *User Modeling 2005* (pp. 154-163). doi:10.1007/11527886_20

Vanderdonckt, J., Furtado, E., Furtado, J. J. V., Limbourg, Q., Silva, W. B., Rodrigues, D. W. T. & da Silva Taddeo, L. (2005). Multi-Model and Multi-Level Development of User Interfaces. En *Multiple User Interfaces* (pp. 193-216). doi:10.1002/0470091703.ch10

Vitharana, P. & Jain, H. (2000). Research issues in testing business components. *Information Management*, *37*(6), 297-309. doi:10.1016/s0378-7206(99)00056-7

Want, R. (2018). *Ubiquitous Computing Fundamentals* (C. John Krumm & Hall/CRC, Eds.). doi:10.1201/9781420093612

Wooldridge, M., Müller, J. P. & Tambe, M. (1996). Agent theories, architectures, and languages: A bibliography. En *Intelligent Agents II Agent Theories, Architectures, and Languages* (pp. 408-431). doi:10.1007/3540608052_81

Yi, J. & Maghoul, F. (2011). Mobile search pattern evolution: the trend and the impact of voice queries. En *Proceedings of the 20th international conference companion on World wide web - WWW '11*. doi:10.1145/1963192.1963276

Zuffi, S., Brambilla, C., Beretta, G. & Scala, P. (2007). Human Computer Interaction: Legibility and Contrast. En *14th International Conference on Image Analysis and Processing (ICIAP 2007)*. doi:10.1109/iciap.2007.4362786