Introduciendo a los Usuarios Finales en el Desarrollo de Sistemas Pervasivos. Una Aproximación MDD

Tesis del máster oficial en Ingeniería del Software, Métodos Formales y Sistemas de Información

María Francisca Pérez Pérez

Supervisores
Dr. Pedro J. Valderas Aranda
Dr. Joan Fons Cors

Diciembre 2009

Centro de Investigación en Métodos de Producción de Software

UNIVERSIDAD POLITECNICA DE VALENCIA

Contenido

Lista de Figuras

Capítulo 1

Introducción

Los usuarios finales conocen el problema y sus necesidades mientras que los desarrolladores, conocen la tecnología para resolverlo. Sin embargo, estos usuarios finales no conocen los lenguajes y notaciones abstractas utilizadas por los analistas para representar los sistemas, y por tanto participar en las etapas del proceso de desarrollo. El trabajo desarrollado en esta tesis de máster introduce a los usuarios finales en las etapas tempranas del proceso de desarrollo de sistemas pervasivos. Para ello, proponemos una aproximación metodológica basada en técnicas y metáforas aceptadas en el área de *End-user Development*. Esto permite a los usuarios finales describir su hogar inteligente mediante representaciones visuales y conceptos que resultan familiares para ellos. Las descripciones realizadas, se integran en un método de desarrollo dirigido por modelos con capacidad de generación de código.

El resto de este capítulo está estructurado en cinco secciones. La Sección 1.1 motiva el desarrollo de este trabajo. La Sección 1.2 presenta en detalle los problemas planteados. La Sección 1.3 resume las contribuciones. La Sección 1.4 presenta el contexto de este trabajo. Finalmente, la Sección 1.5 describe la estructura del documento.

1.1 Motivación

En las últimas décadas, han surgido muchas iniciativas de investigación interesadas en hacer realizar la visión de computación ubicua [1]. Estas iniciativas centran sus esfuerzos en integrar la tecnología en nuestro entorno cotidiano [2]. Cada vez más, se centran esfuerzos en transformar las casas dónde vivimos en entornos de computación ubicua. Sin embargo, para lograr estas promesas de computación ubicua, los sistemas pervasivos y en concreto las casas inteligentes, tienen que mejorar las actividades cotidianas sin perder la aceptación del sistema por parte del usuario [3].

Los sistemas pervasivos son sistemas complejos, no sólo porque se involucran muchos dispositivos distintos, sino también porque los usuarios finales requieren estos dispositivos para estar (1) perfectamente integrados y (2) adaptados a sus necesidades particulares. Los usuarios finales no sólo son usuarios "informales", "novatos" o "inocentes", pueden ser químicos, bibliotecarios, arquitectos, profesores o contables que tienen necesidades computacionales y necesitan hacer un uso intenso de computadores. No obstante, estos usuarios no están interesados en convertirse en programadores profesionales [4]. Boehm et al. [] predijeron el crecimiento exponencial del número de usuarios finales desarrolladores comparado con el número de profesionales de software, destacando la importancia de la investigación en *End-user Development*. *End-user Development* se define como un conjunto de actividades o técnicas que permiten a personas que no son desarrolladores de software profesionales, crear o modificar un artefacto software.

En *End-user Development* muchas técnicas tienen como objetivo permitir a los usuarios finales programar o personalizar su sistema. El uso de técnicas *End-user Development* puede proporcionar varios beneficios en el caso particular de las casas inteligentes. Por ejemplo, proporciona a los usuarios control sobre combinaciones imprevisibles de interoperabilidad de dispositivos [5] y también, permite a los usuarios personalizar servicios [6]. Por tanto, *End-user Development* persigue una alineación natural entre las expectaciones del usuario final y las capacidades del sistema.

Por otro lado, es bien conocido que actividades de ingeniería de

requisitos eficientes, son esenciales para desarrollar sistemas software que satisfagan las expectaciones del usuario adecuadamente. Estas actividades son incluso más importantes en el caso particular de los sistemas pervasivos. Los sistemas pervasivos intentan construir entornos donde los elementos de computación desaparezcan desde punto de vista del usuario, pero su funcionalidad, que proporciona a los usuarios servicios específicos, se continua proporcionando [1].

Se necesitan inversiones importantes para desarrollar y desplegar tanto la infraestructura hardware que tiene que ser integrada en el entorno y, el software que utiliza esta infraestructura para proporcionar los servicios adecuados. Así, los cambios en los requisitos de un sistema pervasivo una vez que el proceso de desarrollo ya ha comenzado, puede exigir cambios en el software y en la infraestructura hardware. Estos cambios implican un esfuerzo adicional, tiempo y dinero.

En conclusión, nos hemos centrado en usuarios finales por cuatro razones: (1) los usuarios finales conocen en profundidad sus actividades y entorno más que cualquier desarrollador, (2) si sólo un desarrollador puede controlar el comportamiento del sistema, los usuarios finales no podrán evolucionarlo cuando su entorno o actividades cambien, (3) para potenciar el desarrollo por parte de los usuarios finales y que por sí mismos puedan personalizar los sistemas y, (4) en sistemas de casas inteligentes, la mayoría de las acciones de los sistemas se basan en información percibida e interpretada implícitamente del usuario final.

1.2 Problema Planteado

El problema de diseñar un sistema de programación para el usuario final es muy difícil, como explicó Nardi [7]. En el caso de permitir a los usuarios finales participar en el desarrollo de casas inteligentes, la dificultad no es sólo el diseño de un sistema de programación para ellos. En este tipo de sistemas, es necesario proporcionar servicios diferentes para respaldar las actividades diarias de los habitantes de la casa. Con el fin de hacer esto, los sistemas de casas inteligentes realizan de forma automática acciones como encender una luz [8], control de un termostato, cierre de persianas, etc. Sin embargo, todas estas acciones

deben realizarse de acuerdo con el entorno del usuario, preferencias y necesidades.

Además, no siempre es una tarea fácil adaptar sistemas de casas inteligentes a las necesidades de usuarios finales debido a la falta de comprensión que puede existir entre usuarios finales y desarrolladores de sistemas ya que, poseen distintos tipos de conocimiento. Los usuarios finales conocen el problema y sus necesidades mientras que, los desarrolladores conocen la tecnología para resolver el problema. Los usuarios finales son los dueños del dominio de conocimiento, tienen más conocimiento sobre los servicios que se deben proporcionar por el sistema y el entorno en el que el sistema se va a utilizar. Sin embargo, muchas veces no tienen la capacidad de transmitir esta información correctamente. Este es un problema general, conocido como *"clients do not really know what they want"*.

Con el fin de abordar el desarrollo de sistemas pervasivos, en nuestro centro de investigación de Métodos de Producción de Software (ProS), se ha desarrollado un método *Model Driven Development* (MDD) [9] para el desarrollo de sistemas pervasivos. Su objetivo es soportar el desarrollo de software integrando componentes múltiples y heterogeneos (dispositivos y sistemas software) para proporcionar un conjunto de servicios pervasivos. También, se ha desarrollado un método *Model Driven Development-Software Product Line* (MDDSPL) que aborda la adaptación de sistemas pervasivos en tiempo de ejecución. Sin embargo, los usuarios finales no pueden participar en la especificación de su sistema pervasivo utilizando estos métodos debido a la complejidad de especificar mediante modelos. Para especificar estos modelos, se requieren expertos en el modelado y desarrollo de sistemas pervasivos. Además, se requiere que estos expertos estén familiarizados con el dominio del sistema a desarrollar (la casa inteligente) y sus primitivas (por ejemplo: dispositivo, servicio o *trigger*). Esto es un problema porque los usuarios finales no pueden personalizar el sistema pervasivo por sí mismos. Para solucionar este problema, proponemos ofrecer mecanismos que permitan a los usuarios finales participar activamente en el proceso de desarrollo para transmitir sus necesidades y personalizar su sistema cuando sus necesidades cambien.

1.3 Contribuciones

En esta tesis de máster, se presenta una aproximación que permite a los usuarios finales describir sistemas pervasivos de acuerdo a sus expectativas y necesidades. Esta aproximación se ha desarrollado para lograr de los siguientes objetivos: (a) permitir a los usuarios finales describir las principales características de un sistema pervasivo (concretamente un hogar digital) de acuerdo a sus necesidades, (b) proporcionar a los usuarios finales mecanismos que les permitan especificar la información de su sistema de acuerdo a sus conocimientos, entorno y sistema, (c) integrar las descripciones del usuario final en el método de MDD [9] para el desarrollo de sistemas pervasivos propuesto en nuestro centro de investigación y (d) proporcionar a los usuarios finales herramientas que les permitan describir y personalizar su sistema pervasivo.

Las principales contribuciones de esta tesis de máster se han desarrollado para abordar los problemas planteados en la sección anterior y los objetivos presentados anteriormente. Son las siguientes:

1. **Análisis para introducir a los usuarios finales** en los métodos dirigidos por modelos (PervML y MDDSPL) desarrollados previamente en nuestro centro de investigación.

2. **Aproximación metodológica** que extiende el método MDDSPL para permitir a los usuarios participar en las etapas tempranas del proceso de desarrollo de su hogar digital. Esta aproximación sigue un proceso que hemos propuesto donde se describe en detalle qué fases y pasos a seguir para crear la descripción del sistema y, cómo los usuarios finales y los analistas interactúan entre sí.

3. **Representación visual** inspirada en técnicas y metáforas *End-user Development* que representa el entorno del usuario final con conceptos familiares y, permite que los usuarios finales describan y visualicen las características de su sistema. Para dar soporte a esta representación visual, hemos implementado una herramienta a nivel de prototipo.

Estas contribuciones están descritas en detalle en los Capítulos 4, 5 y 6 respectivamente.

1.4 Contexto

Esta tesis de máster ha sido desarrollada en el Centro de Investigación en Métodos de Producción de Software (ProS) de la Universidad Politécnica de Valencia. El trabajo realizado para el desarrollo de esta tesis, ha sido posible en el contexto de los siguientes proyectos de investigación:

- SESAMO: Construcción de Servicios Software a partir de Modelos. CYCIT project referenced as TIN2007-62894.
- OSAMI Commons: Open Source Ambient Intelligence Commons. ITEA 2 project referenced as TSI-020400-2008-114.
- Atenea: Arquitectura, Middleware y Herramientas. ProFIT project referenced as FIT-340503-2006-5.

1.5 Estructura del Documento

El resto del documento de esta tesis está estructurado en los siguientes capítulos:

Capítulo 2: Estado del arte en End-user Development. Este capítulo presenta el concepto de *End-user Development*, un estudio de técnicas y metáforas aplicables a sistemas pervasivos y, algunas implementaciones de ellas.

Capítulo 3: Background. Este capítulo presenta conceptos y tecnologías relacionadas con el trabajo presentado en esta tesis.

Capítulo 4: Introduciendo a los Usuarios Finales en el Desarrollo de Sistemas Pervasivos. Este capítulo presenta desafíos para introducir a los usuarios finales en el desarrollo de sistemas pervasivos en el método MDD. También, analiza los desafíos para introducir a los usuarios finales en el método MDD-SPL en tres actividades del proceso de desarrollo:

elicitación de requisitos, diseño e implementación y, mantenimiento. Después, presenta brevemente una aproximación metodólogica para introducir a los usuarios finales en el método MDD y, otra aproximación metolodológica para el métodos MDD-SPL.

Capítulo 5: Aproximación MDDSPL para Introducir a los Usuarios Finales en las Etapas Tempranas del Proceso de Desarrollo. Este capítulo describe en detalle la aproximación metodológica para introducir a los usuarios en el método MDDSPL.

Capítulo 6: Herramienta. Este capítulo presenta la herramienta prototipo desarrollada para dar soporte a la aproximación MDD-SPL descrita en el capítulo anterior.

Capítulo 7: Conclusiones. Este capítulo concluye con un análisis de la solución propuesta en esta tesis. También, resume las contribuciones y propone trabajos futuros.

Capítulo 2

Estado del arte en End-user Development

Este capítulo presenta un análisis de los proyectos de investigación más importantes en *End-user Development*, técnicas e implementaciones. Se prestará especial atención a las técnicas e implementaciones relacionadas con sistemas pervasivos.

2.1 *End-user Development*

Lieberman et al. [10] opina que en los próximos años, el objetivo de *Human-Computer Interaction* (HCI) evolucionará a partir de los sistemas fáciles de usar (a pesar de que ese objetivo todavía no se ha cumplido por completo) para hacer que los sistemas sean fáciles de desarrollar. Por ahora, la mayoría de personas se han familiarizado con la funcionalidad básica y las interfaces de los ordenadores. Sin embargo, desarrollar nuevas aplicaciones o modificar aplicaciones que apoyen eficazmente las los objetivos de los usuarios todavía requiere una experiencia considerable en la programación que no se puede esperar

de la mayoría de usuarios. Así, un desafío fundamental para los próximos años es desarrollar entornos que permitan a los usuarios que no tienen experiencia en programación desarrollar o modificar sus propias aplicaciones, con el objetivo de potenciar a los usuarios a utilizar de forma flexible información avanzada y tecnologías de comunicación.

Los usuarios finales en general no están cualificados ni interesados en la adaptación de sus sistemas al mismo nivel que los profesionales de software. Sin embargo, es muy conveniente habilitar a los usuarios a adaptar los sistemas a un nivel de complejidad que se adecue a sus habilidades individuales y situaciones. Esto es **el objetivo principal de *End-user Development*** (EUD): capacitar a los usuarios finales a desarrollar y adaptar los sistemas por sí mismos).

Entonces, la definición más común para *End-user Development* es la siguiente [10]:

> ***End-User Development* es un conjunto de actividades o técnicas para permitir a los usuarios, que no son desarrolladores de software profesionales, en algún momento crear o modificar un artefacto software.**

La creciente cantidad de software embebido dentro de los consumidores y los productos profesionales también apunta a la necesidad de promover la EUD para una correcta utilización de estos productos. En el plano político la EUD es importante para la plena participación de los ciudadanos en la emergente Sociedad de la Información. La Sociedad de la Información se caracteriza por redes de ordenadores, la integración de otros medios de comunicación tradicionales dentro de las redes digitales y, facilitar el acceso a través de una variedad de dispositivos de interacción que van desde pequeños teléfonos móviles a grandes pantallas planas. Sin embargo, la creación de contenidos y la modificación de la funcionalidad de esta infraestructura de la red son difíciles para los usuarios no programadores profesionales, lo que para muchos sectores de la sociedad es una división del trabajo entre quienes producen y quienes los consumen. EUD tiene el potencial para contrarrestar estos efectos.

El emergente campo de investigación de la EUD integra diferentes hilos de discusión de *Human Computer Interaction (HCI)*, Ingeniería de

Software (SE), de equipos compatibles trabajo colaborativo (CSCW), y inteligencia artificial (AI). Conceptos tales como: *tailorability, configurability, end-user programming, usability, visual programming, natural programming,* y *programming by example*, forman ya una base fructífera pero que necesitan estar mejor integrados y, la sinergia entre ellos explotada más plenamente.

Podemos identificar dos tipos de actividades de los usuarios finales de una perspectiva de diseño centrada en el usuario:

1. **Parametrización o personalización.** Actividades que permiten a los usuarios elegir entre comportamientos alternativos (presentaciones o mecanismos de interacción) ya disponibles en la aplicación.

2. **Creación de programas y personalización.** Actividades que implican alguna modificación de software dirigida a crear desde cero o modificar algún artefacto software.

EUD abarca de forma más adecuada el segundo tipo de actividad ya que con el primer tipo, la modificación del software se limita estrictamente a opciones predeterminadas o formatos. Técnicas actuales de *End-User Development* e implementaciones que pertenecen al segundo tipo se describen en detalle en las siguientes secciones.

2.2 Proyectos de investigación actuales

Dos Congresos Nacionales de ciencias determinaron que un usuario final de software, requiere seria atención [11]. Dos recientes esfuerzos colaborativos, uno en U.S. (**the EUSES Consortium** [1]), y otro en Europa (**the Network of Excellence on End-User Development** [2]) han producido un gran número de resultados en esta área. Algunos ejemplos de trabajo actual son los siguientes:

- **El proyecto *"Natural Programming"*** de la Universidad Carnegie Mellon [12], ha estado trabajado desde hace más de 10

[1] http://eusesconsortium.org/
[2] http://giove.cnuce.cnr.it/eudnet. htm

años con el objetivo de hacer la programación más "natural" o, acercar su conocimiento para ayudar a novatos, profesionales y programadores end-user. Se han realizado muchos estudios (e.g., [10; 13]), nuevos lenguajes y ambientes de programación. Algunas aplicaciones creadas en estos proyectos son: *the Whyline*, la cual permite a los programadores preguntar "Por qué" y "Por qué no" acerca del funcionamiento de sus programas y, les ayuda a entender mejor las causas de ejecución de sus programas.

- **El proyecto *"End-User Software Engineering"*** de la Universidad del Estado de Oregon, tiene como objetivo mejorar la fiabilidad del software producido para usuarios finales, programadores en general y, en particular, para todos los usuarios de hojas de cálculo [14]. Algunos resultados incluyen *"What You See Is What You Test"* (WYSIWYT), integrado con localización facultativa, detección semi-automática de combinaciones erroneas en módulos de hojas de cálculo y, nuevos métodos que envuelven a los usuarios finales en el proceso de depuración de los programas *"machine-learning"*.

- **El proyecto *Gender HCI*** [3], una colaboración de la Universidad del Estado de Oregon y la Universidad Drexel, tiene como objetivo dar soporte a la solución de los problemas de los géneros femenio y masculino, especialmente en las tareas de desarrollo de sofware para usuario final. El resultado de este proyecto presenta que las mujeres están menos dispuestas que los hombres a intentar adoptar características software que soporten *testing* y *dataflow-oriented debugging*, y que remotamente, los hombres y las mujeres utilizan estratégias diferentes en el proceso de *debugging*.

- **El proyecto *Informal Learning in Software Construction*** estudia las situaciones y comunidades del mundo real que puedan motivar y ayudar a los "no programadores" y usuarios finales en el aprendizaje de herramientas de programación [14]. Trabajos previos han caracterizado los problemas de aprendizaje de los profesores de escuelas públicas para construir simulaciones

[3] http://eusesconsortium.org/gender/

visuales y, diseñar entrenamientos con materiales mínimos y código reusable que pueda servir como plataforma. Recientes investigaciones en este proyecto, están estudiando los modelos mentales de la construcción de software para la web. Dicha construcción está sostenida por usuarios finales sofisticados y, están usando estos resultados para desarrollar herramientas enfocadas en la construcción de aplicaciones web simples.

La siguiente subsección describe técnicas *End-user development* para sistemas pervasivos.

2.3 Técnicas *End-user Development*

2.3.1 *Programming by example*

En los sistemas *Programming by demonstration* (PBD), también onocidos comoc *Programming by example* (PBE), el usuario demuestra con ejemplos cómo desea el comportamiento del sistema [15]. PBE fue presentado por Smith a mediados de los 70 [16], donde el comportamiento computacional deseado era demostrado por los usuarios finales a través de ejemplos concretos, en forma abstracta (eg. código). Originalmente PBE fue enfocado exclusivamente a los ambientes de escritorio pero, recientemente, un número de investigadores ha comenzado la explorar cómo estas ideas pueden ser aplicadas a hogares inteligentes, inventando los sistemas embebidos y distribuidos (normalmente integrados en las aplicaciones del hogar).

La funcionalidad de estos sistemas fue demostrada directamente por los usuarios finales a través de ejemplos concretos [17]. Actualmente, la mayoría de las herramientas de usuarios finales, para aplicaciones pervasivas, están basadas en la metáfora de la programación procedural (ver algunos ejemplos en la Subsección 2.3.3). Esto requiere que el usuario mentalmente manipule la construcción, la cual resulta más familiar para la mayoría de programadores (aunque en una gráfica o en una macro). Los Sistemas PBE tienen dos niveles de representación [18]:

1. **El nivel *GUI***. Este nivel caracteriza a las típicas ventanas, íconos y menus. Por ejemplo, en una aplicación de procesamiento de palabras, como *Word*, este nivel representa el contenido directamente manipulado por los usuarios, como son: las operaciones de escribir un texto nuevo, dar formato a textos y el uso del cursor para navegar por un documento.

2. **El nivel de programa**. Este nivel captura la manipulación del usuario directamente dentro de los programas así, el usuario puede volverlos a ejecutar. En *Word*, este nivel lo incorpora *Visual Basic*. Las manipulaciones de los usuarios son guardadas en *scripts* de *Visual Basic*, entonces, los usuarios asignan *scripts* a través de comandos de teclado, o de la herramienta de comandos definida por los usuarios.

Dey et al. [19] cree que los PBE ofrecen un gran potencial para soportar comportamientos dinámicos y complejos. PBE es una tecnica interesante que permite a los usuarios finales personalizar su sistema, pero no todos lo utilizan. Esto se debe a que representa un cambio radical de lo que conocemos como programación; esto no puede ser remediado, a pesar de lo extendida que se encuentra la existencia de los sistemas de información, demostrando su viabilidad y valor la mejora de las aplicaciones en una gran variedad de dominios. El conservatismo de la comunidad de programadores, es el obstaculo más grande de la expansión del uso de PBE [15].

2.3.2 *Natural Programming*

Natural Programming es una aplicación de un proceso de diseño estándard centrado en los usuarios para el dominio específico de los entornos y lenguajes de programación [13].

La premisa de esta propuesta es que los programadores tendrán un trabajo más facil si sus tareas de programación son más naturales. Por "natural", Myers entiende "fiel representación de la naturaleza o la vida" [13], lo cual implica que su trabajo será de la misma forma que la gente espera. Por "Programación Natural" apunta a que los lenguajes y ambientes de trabajo sea de la misma forma que para los no

programadores esperan. De este modo, la Programación Natural tiene como objetivo hacer posible que la gente exprese sus ideas de la misma forma que las piensan.

El proceso de diseño de *Natural Programming*, trata la usabilidad como un objetivo fundamental. Para alzanzarlo, se siguen los siguientes pasos:

1. Identificar el público objetivo y el dominio. Esto es, el grupo de personas usuarios del sistema y los tipos de problemas en que ellos trabajarán.

2. Entender el público objetivo, estudiar el lenguaje actual, técnicas, y los pensamientos que ellos naturalmente usan cuando intentan resolver los problemas. Esto incluye un conocimiento de los principios generales de HCI como trabajo previo, la psicología de la programación y estudios empíricos. Cuando surjan temas o preguntas del trabajo previo, es necesario dirigirse a nuevos estudios de usuarios y examinarlos.

3. Diseñar el nuevo sistema basado en ésta información.

4. Evaluar el sistema para medir el éxito y entender cualquier problema nuevo que los usuarios tengan. Rediseñar el sistema basado en esta evaluación y luego, reevaluarlo siguiendo el principio de diseño iterativo del estandar HCI.

Natural Programming es de una significante importancia para *End-user Development*. Además, para soportar nuevo conocimiento y herramientas directamente, la aproximación centrada en el usuario seguida por *Natural Programming*, proporciona un modelo de una metodología que puede ser seguido por otros desarrolladores e investigadores cuando diseñan sus propios lenguajes y ambientes. Myers et al. [13] consideran que que la *Natural Programming* dará origen a herramientas más usables y efectivas, que le permitan a usuarios finales y profesionales, escribir de forma más facil y correcta sus programas.

2.3.3 Metáforas

Esta subsección presenta tres metáforas:

1. ***Magnetic Poetry sets*** [20]: consiste en reducidos, flexibles e individuales imanes, donde cada uno tiene una palabra impresa. Los usuarios pueden combinar palabras en un "Poema" . *Magnetic Poetry sets* algunas veces se centra en un tópico o en un tema, los cuales pueden ser "amor" u "ordenadores" y contiene palabras relacionadas con ese tema, generando asi poemas orientados al tópico analizado.

2. ***Jigsaw*** [6]:esta basado en la familiaridad evocada por la noción y el indicio intuitivo de conectar y ensamblar piezas de puzle juntas. Esencialmente, esto le permite a los usuarios conectar componentes y componer varios planes a través de series de izquierda-a-derecha de piezas. La restricción de que las conexiones se realicen de izquierda a derecha, proporciona a los usuarios la misma sensación que describe un flujo de información. La metáfora del *Jigzaw* es un sistema basado en reglas.

3. ***Butler*** [7]: esta es una metáfora para la interacción de un usuario enseñandole a un mayordomo. La entrada de esta modalidad es un diálogo hablado con el entorno. La metáfora del mayordomo es un sistema basado en reglas. Por ejemplo, si el usuario desea encender la luz de la cocina, el usuario le debe decir al mayordomo "enciede la luz de la cocina" y entonces el sistema enciende la luz de la cocina. Previamente, el usuario debe programar las acciones del mayordomo, y cuando el sistema esté en tiempo de ejecución, el usuario debe recordar alguna información básica para usar el sistema.

La siguiente sección presenta una implementación para algunas técnicas y metáforas presentadas anteriormente.

2.4 Implementaciones de Usuario Final para sistemas pervasivos

Esta sección presenta 5 implementaciones de usuario final, siguiendo algunas de las técnicas y metáforas descritas en la sección anterior. Estas implementaciones se han enfocado en sistemas pervasivos.

2.4.1 *Pervasive Interactive Programming* (PiP)

Pervasive Interactive Programming (PiP) [17] sigue la aproximación descrita *Programming by Example* (PBE) (ver Subsección 2.3.1). Esto permite un acercamiento no-técnico a los usuarios finales para que programen su entorno y satisfagan sus necesidades particulares. PiP proporciona una plataforma que utiliza el espacio físico del usuario como entorno de programación así, proporciona un mecanismo natural y familiar para que el usuario "programe" la funcionalidad que requiere para satisfacer sus necesidades particulares. Por tanto, con un esfuerzo mínimo, un usuario final no-técnico, puede personalizar la funcionalidad coordinando grupos de dispositivos del sistema pervasivo que sólo podian ser manipulados por programadores convencionales. Todos lo que los usuarios finales necesitan hacer es simplemente expresar las funcionalidades y comportamiento que requiere del sistema realizando acciones físicas dentro del entorno (ver Fig. 2.1).

Chin et al. [17] se ha inspirado en la facilidad con la cual la gente lleva a cabo sus tareas diarias rutinarias (por ejemplo, encender las luces cuando una habitación se encuentra oscura, silenciar el sonido de la televisión cuando un teléfono suena, etc). Por eso, decide dirigir sus enfoques para **encontrar una forma de programación que sea más natural y que imite las prácticas más familiares, sin la necesidad de que los usuarios sigan un conjunto de secuencias lógicas y acciones estrictas**.

PiP se ha diseñado para trabajar en tiempo real dentro de un entorno pervasivo. La comunicación entre PiP, el usuario final y el entorno es a través de mecanismos de eventos. Por tanto PiP tiene una arquitectura asíncrona orientada a objetos basada en eventos. A diferencia de los lenguages macro, donde la secuencia de acciones es

Fig. 2.1: Demostración que requiere acciones físicas

Fig. 2.2: PiP encuentro de ambientes pervasivos

importante, PiP asume que la secuencia lógica de las acciones no es importante. PiP impulsa la tecnología UPnP[4] como su *middleware* y protocolo de comunicación, permitiendo conectividad simple y robusta entre dispositivos y ordenadores. UPnP tiene un *framework* modular, comprendido por seis modulos, los cuales trabajan juntos para soportar una red de cómputo de tiempo real.

En un ambiente PiP (ver Firgura 2.2), el usuario puede escoger cuando informar al sistema que se encuentra listo para comenzar

[4]UPnP red tecnológica que ofrece dispositivos y servicios a su red de clientes. Mas detalles: http://www.upnp.org/

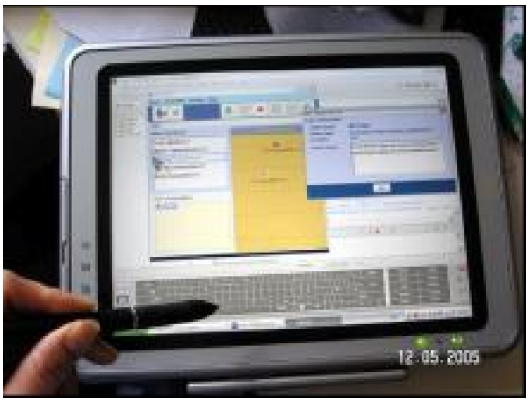


Fig. 2.3: PiP on tablet view

a programar (*show*) la funcionalidad del dispositivo, esto puede ser llevado a cabo usando alguno de estos tres métodos: (1) interactuando físicamente con el dispositivo, 2) usando paneles de control de Interfaz de Usuario (UI) (ver Figure 2.3) (3) una combinación entre los dos métodos. Según las acciones del usuario, los dispositivos pervasivos generan los eventos apropiados, los pasan a la red y PiP codifica su información como un conjunto de reglas con dos partes; un antecedente (condiciones) y un consecuente (el resultado de la acción), PiP "escucha" y "captura" las acciones que el usuario presenta.

Los resultados muestran que el 83% de los participantes donde se encuentra disponible el PiP, programan sus ambientens con poca asistencia o sin ella, aunque el tiempo que toma llevar a cabo estas tareas varía de participante en participante.

2.4.2 *a CAPpella*

A CAPpella es un entorno de prototipado *Context-Aware* pensado para usuarios finales [19]. Esto sigue la técnica *Programming by Example* descrita en la Subsección 2.3.1. Los usuarios "programan" su comportamiento *context-aware* deseado (situaciones y acciones asociadas) sin escribir ningún código, sólo demostrando *a CAPpella* y anotando las partes relevantes de la demostración. A *CAPpella* es una aproximación válida que permite a los usuarios finales construir

sus propias aplicaciones *context-aware*. Sin embargo, esto es limitado a situaciones donde los usuarios pueden suponer razonablemente un diseño estático, reglas bien especificadas y que de una forma precisa describan el comportamiento *context-aware* deseado.

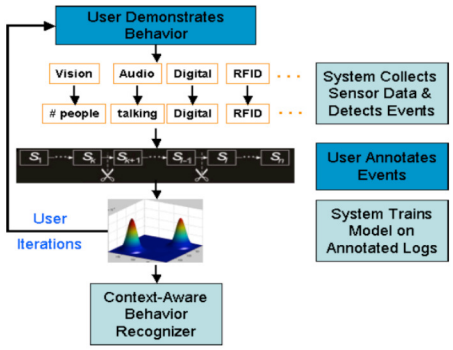


Fig. 2.4: A CAPpella. Diseño para comportamientos *context-aware*

A CAPpella usa una combinación de máquina de aprendizaje y entradas de usuario, para soportar la construcción de aplicaciones *context-aware* a través de *Programming by Example*. Un usuario de *a CAPpella* demuestra un comportamiento *context-aware* que incluye una situación y una acción asociada (Figure 2.4). El usuario emplea una Interfaz gráfica (GUI), para indicar que partes de la demostración son relevantes para el comportamiento, y entrenar *a CAPpella* su comportamiento mientras se dan múltiples ejemplos. Una vez entrenado el sistema, el usuario puede ejecutar *a CAPpella* para representar la demostración del comportamiento (representando las acciones demostradas siempre que se detecte la situación).

La interfaz de usuario se divide en tres partes. En el frame izquierdo, habrá un reproductor de video que permitira que el usuario observe los videos grabados y escuche las grabaciones de audio. En la parte superior del frame derecho, el usuario puede ver los eventos detectados por el sensor que graba los datos y, en la parte inferior, se pueden observar las acciones que ha llevado a cabo el usuario durante la sesión. Después de observar los datos capturados, el usuario anota los datos: seleccionando los flujos de información que considere relevantes para el comportamiento que ha sido creado, y la acciones que quiere representar

de parte de una *a CAPpella*. El usuario comienza una serie y finaliza el tiempo para todos los flujos, indicando cuando un comportamiento ha comenzado y cuando ha finalizado, esto es mostrado en la figura 2.5.

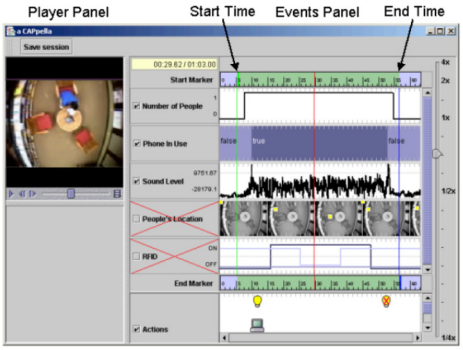


Fig. 2.5: La interfaz de usuario de a CAPpella

En resumen, un usuario primero graba un comportamiento, una situación y una acción que quiera aprender *a CAPpella*. Entonces, el usuario selecciona los eventos relevantes de los datos grabados y los utiliza para el entrenamiento. Después de un número ejemplos de entrenamiento suficiente, *a CAPpella* puede reconocer la situación, y cuando esto sucede, puede representar las acciones demostradas.

2.4.3 *Capture and Access Magnetic Poetry (CAMP)*

Truong et al. [20] desarrollaron CAMP (*Capture and Access Magnetic Poetry*), un entorno de programación de usuario final que permite a los usuarios crear aplicaciones *context-aware* para el hogar. CAMP tiene una interfaz gráfica de usuario (GUI) basada en la metáfora *magnetic poetry sets* descrita en la Subsección 2.3.3; esto le permite a los usuarios crear aplicaciones orientadas a tener ventajas de flexibilidad de lenguaje natural. *Magnetic poetry sets* requiere pocas, o ninguna instrucción especializada, ni conocimiento previo del uso, ya que tiene las ventajas del lenguaje natural (ver Figura 2.6).

CAMP permite a los usuarios crear programas que reflejen el comportamiento que ellos conciben de la aplicación deseada, mas que las necesidades que los usuarios especifican en las aplicaciones en términos

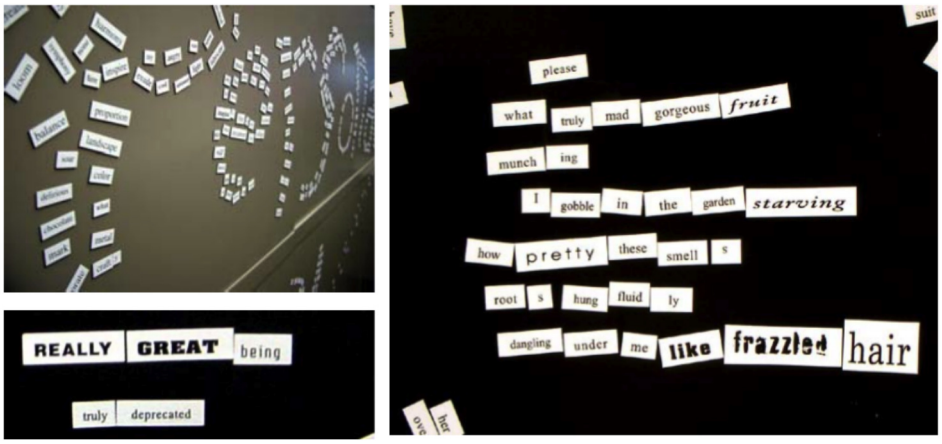


Fig. 2.6: Ejemplo de *Magnetic poetry sets*

de dispositivos. Para los usuarios, *Magnetic poetry sets* está basada en descripción de la aplicación, CAMP genera una especificación a partir de una captura válida de la aplicación, la cual puede ser ejecutada en una captura habilitada en un entorno. CAMP hace uso de las restricciones y un vocabulario de dominio específico, esto evita muchas de las dificultades que envuelven el proceso de conversión del lenguaje natural. CAMP le sirve de interfaz a INCA, una infraestructura que proporciona una abstracción para el desarrollo y captura de acceso a las aplicaciones. La interfaz se diseña para permitir a las personas utilizar un lenjuaje de entrada con el cual se pueden sentir cómodos y les permite expresar sus ideas de forma flexible; CAMP automáticamente genera la tecnología orientada a la aplicación, dando las especificaciones necesarias para la realización de las mismas.

Los usuarios usan cuatro categorías (quién, qué, dónde, cuándo) para capturar, acceder y describir las aplicaciones (ver Fig. 2.7). Algunos ejemplos de cada uno son los que se pueden ver a continuación:

- **Quién:** Yo, todos, no uno, familia, desconocido, bebé, esposa, Billy, etc.
- **Qué:** pintura, audio, vídeo, conversación, etc.
- **Dónde:** cocina, sala, casa, todas partes, etc.

- **Cuándo:** siempre, mas tarde, nunca, a.m, mañana, día, semana, mes, antes, hora, minuto, sábado, enero, una vez, ahora, todo el tiempo, etc.

- **General:** 1, 2, a, la, grabación, recordar, vista, guardar, mantener, micrófono, altavoz, etc.

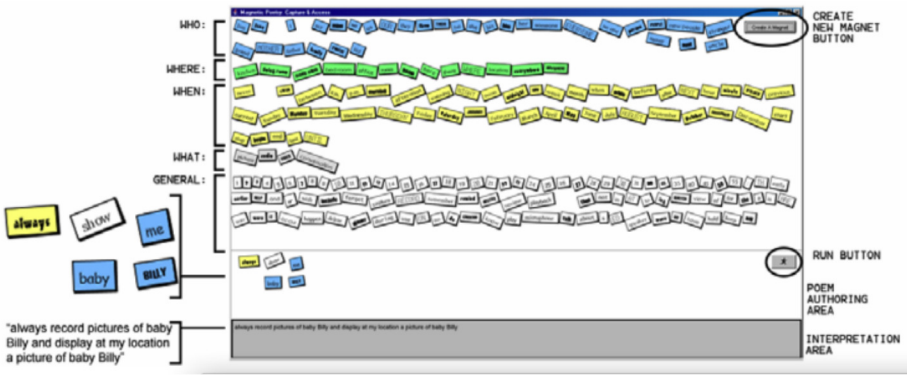


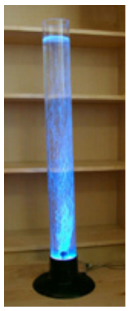
Fig. 2.7: La captura & Acceso a la interfaz *Magnetic Poetry*

CAMP soporta el inicio y parada de captura y acceso cuando dos condicines de contextos específicos ocurren: tiempo, presencia o ausencia de una persona en un lugar. Las condiciones de tiempo son soportadas a través de un reloj que notifica cada cierto tiempo el alcance y la cantidad de datos reportados.

En conclusión, CAMP ayuda a reducir el gap entre las necesidades tecnológicas ofreciendo interacciones que no son técnicamente simples, pero que encajan con el concepto natural del usuario de aplicaciones computacionales ubicuas. La versión actual de este sistema solo permite descripciones de una sola aplicación; esto se complica cuando se desean trabajos que soporten la realización de múltiples aplicaciones en ejecución al mismo tiempo.

2.4.4 The Accord Toolkit

ACCORD ha desarrollado el *Tangible Toolbox*, basado en espacios de datos compartidos, que le permiten a las personas administrar



(a) Bubbles indicates traffic to the family website.

(b) News displayed in the kitchen table and controlled with the ”shaker” objects.

Fig. 2.8: Escenarios desarrollados por Accord Toolkit

fácilmente y reconfigurar servicios basados en dispositivos embebidos alrededor de la casa [21]. Esta herramienta también permite que los dispositivos se integren a través de editores diferentes. El trabajo está fundado sobre estudios etnográficos de hogares, y las actividades que ocurren comunmente dentro de los mismos. Por ejemplo, dos escenarios desarrollados con *Accord Toolkit* se ilustran en la Fig. 2.8.

La infraestructura subyacente está basada en un modelo y un conjunto de editores que direccionan diferentes grupos de usuarios y sus necesidades. Estos editores son:

- ***The Graph Editor*** está dirigido para usuarios expertos, los cuales como programadores, visualizan todos los componentes en un gráfico. Este editor da un buen punto de vista de todos los componentes existentes y cómo se encuentran conectados entre sí. Otro editor apunta a los habitantes de la casa. Con el *Linker Device* los usuarios pueden explorar que propiedades físicas tienen los dispositivos en el hogar, y pueden acceder a dichas propiedades físicas de los dispositivos (ver Figura 2.9(a)).

- ***The Puzzle Editor*** es una interfaz gráfica que está compuesta de servicios para componentes. Este editor esta basado en la metáfora del puzle recreacional (ver Subsección 2.3.3). Un usuario conecta componentes a través de una serie de parejas de izquierda a derecha (ver Figura 2.9(b)).

(a) The Graph Editor

(b) The Puzzle Editor

(c) The Paper Puzzle Editor

Fig. 2.9: Editores Accord Toolkit

- ***The Paper Puzzle Editor*** está enfocado en la creación de una interfaz que no es percibida como una interfaz computacional. Cada componente se representa como una pieza física de puzle de la misma forma que en un editor gráfico de puzle; un servicio esta creado a través de conectar aquellas piezas en un orden de izquierda a derecha (ver Figura 2.9(c)).

2.4.5 Alfred

Alfred [7] sigue la métáfora del mayordomo, descrita en la Subsección 2.3.3. Alfred acerca y habilita al usuario para componer programas via *Programming by Example*, utilizando interacciones verbales o físicas. Alfred es una interfaz de programación de usuario final, la cual le da al usuario la habilidad de programar el sistema de acuerdo a sus necesidades particulares y preferencias. Alfred permite a un usuario final, "programar" el sistema diciendole el nombre de un nuevo objetivo o meta, demostrando uno o más planes para conseguir dicha meta y, finalmente indicándole al sistema las condiciones bajo las cuales debe preferir un plan u otro. Similarmente, el usuario puede nombrar eventos que sugiere el entorno y decirle al sistema que metas deben ser puestas cuando dichos eventos sean sugeridos. Cada uno de estos pasos

pueden ser llevados a cabo por simples comandos verbales o por formas naturales de interacción.

Alfred trabaja con **Rascal** y **ReBa**, dos sistemas formales para componentes adaptativos y reactivos.

En resumen, al utilizar Alfred el usuario puede: 1) Grabar una nueva Macro, 2) Añadir un disparador de *hardware* y 3) Invocar las tareas. En la fase de grabación, Alfred puede observar sus acciones deliberadas (como los comandos que pronuncia, accion y control de dispositivos, etc.) y grabarlos. La información contenida en el notificador de eventos es suficiente para "recrear" los eventos más adelante. La infraestructura para hablar, permite actualizaciones instantáneas del vocabulario, así, nuevas secuencias de tareas pueden ser nombradas, para luego poder ser accedidas y grabadas. Cuando una tarea se lanza (ya sea por comandos o por un lanzamiento de *hardware*), Alfred contacta con los agentes apropiados si el previamente, ha grabado los comandos y eventos que dan lugar a la solicitud lanzada. Al contrario que las aplicaciones para desarrolladores, Alfred pone al usuario en el centro de la creación de la funcionalidad inteligente del entorno. Ocasionalmente, Alfred encuentra errores mientras se ejecutan las secuencias de tareas. En estos casos, Alfred llama al usuario en la tarea que ha fallado y le pregunta si debería intentar ejecutar nuevamente la tarea, o abortar la secuencia.

2.5 Conclusiones

Las implementaciones de técnicas *End-user Development* que han sido introducidas en este capítulo entán enfocadas en desarrollar sistemas pervasivos. Cada implementación tiene benefios y desventajas ya que su aplicación depende principalmente del sistemas pervasivos a desarrollar/personalizar y también depende de la habilidad de los usuarios finales que utilizan la implementación (por ejemplo, el usuario final puede tener diferentes niveles de conocimiento en sistemas pervasivos).

Por ejemplo, la implementación Accord requiere usuarios expertos porque tienen que estar familiarizados con conceptos específicos de dominio. Alfred requiere que el comportamiento de los paquetes estén

escritos por desarrolladores *softwre* y entonces, los usuarios no pueden personalizarlo cuando ellos lo utilicen. También, Alfred ocasionalmente presenta errores cuando ejecuta las secuencias de tareas. A CAPpella es un entorno prototipo *context-aware* que puede ser utilizado en situaciones limitadas.

Las técnicas actuales de programación están principalmente enfocadas en proporcionar metáforas apropiadas para usuarios experimentados, mientras que ninguna de estas técnicas se orienta a la evolución de las necesidades del usuario y los cambios de dispositivos. Otras técnicas end-user, como *Programming by Example* ofrece a los usuarios la posibilidad de personalizar y configurar conjuntos de dispositivos.

Aunque estas técnicas animan a que los usuarios finales a participar en la creación o modificación de los sistemas software, no abordan un proceso dónde los usuarios finales puedan especificar requerimientos de un sistema. Concretamente, estas técnicas permiten que el usuario final personalice el sistema cuando está en run-time y no permiten que los usuarios finales participen en las etapas tempranas del proceso de desarrollo como la actividad de elicitación de requisitos.

Capítulo 3

Background

Este capítulo presenta el *background* relacionado con el trabajo de esta tesis para introducir a los usuarios finales en el desarrollo de sistemas pervasivos.

El resto de este capítulo está estructurado en 4 secciones, son las siguientes: La sección 3.1 introduce los sistemas pervasivos y sus principales características. La Sección 3.2 presenta el método PervML desarrollado en nuestro centro de investigación y el Lenguaje Específico de Dominio para describir los modelos en el método. La Sección 3.3 introduce la Línea de Productos Software y por último, la Sección 3.4 describe el método MDDSPL desarrollado en nuestro centro de investigación que aplica una Línea de Productos Software para el desarrollo de sistemas reconfigurables.

3.1 Sistemas Pervasivos

Los sistemas pervasivos intentan hacer la visión descrita por Weiser a principios de los 90's realidad [1]. Esta visión se basa en la construcción de entornos de computación integrados adecuadamente con los usuarios.

El gran desafío de esta visión, es la integración de varias tecnologías existentes (ordenadores de bolsillo, comunicaciones de banda ancha, sensores, sistemas de software, etc) en un todo homogéneo.

Una descripción completa de las características de los sistemas pervasivos está fuera del alcance de esta tesis de máster, sin embargo, algunas de sus características pueden ayudar a comprender la necesidad de nuevas técnicas y métodos para poder desarrollar estos sistemas. A continuación, se describen estas características (identificadas por [22]) y se indica la influencia decisiva que tienen en la obtención de los requisitos de un sistema pervasivo:

- **Sensibilidad al contexto:** es la capacidad de un sistema para hacer frente a la información de contexto. El contexto puede ser considerado como información sobre su entorno físico, el usuario, o la situación social. Por ejemplo, algunas funciones del sistema podrían desactivarse en función de la hora o el comportamiento del sistema puede cambiar en función del número de los usuarios en una habitación.

- **Proactividad:** es la capacidad de un sistema para proporcionar funcionalidad sin una petición explícita por parte del usuario. Para soportar esta característica, el sistema debe saber cuando tomar la iniciativa. En general, el sistema pervasivo reaciona frente alguna situación de contexto, por lo que la proactividad está estrechamente relacionada con la sensibilidad al contexto. Por ejemplo, el sistema pervasivo muestra la información sobre un paciente cuando este entra en la consulta médica.

- **Movilidad:** desde que los sistemas pervasivos proporcionan funcionalidad en un entorno físico, los usuarios pueden moverse dentro de él. Esta característica tiene que tenerse en cuenta en el sistema para ofrecer los servicios adecuados en la localización actual del usuario (por ejemplo, no tiene sentido proporcionar los mismos servicios en la cocina que en la sala de estar). Además, el mismo tipo de funcionalidad (como el control de la iluminación) puede estar disponible en lugares distintos pero con un alcance físico distinto.

- **Personalización:** es la capacidad de adaptar el sistema a las características del usuario y sus preferencias. Cada usuario tiene diferentes habilidades, gustos y objetivos. Un sistema pervasivo debe adaptarse de forma dinámica para mejorar la experiencia del usuario. Por tanto, la misma funcionalidad puede ser proporcionada de una manera distinta dependiendo del usuario. Por ejemplo, algunos usuarios prefieren activar el servicio de climatización cuando la temperatura alcanza 23 grados centígrados, mientras que otros prefieren que se active cuando alcance 25 grados.

3.2 PervML: un Método guiado por Modelos para el Desarrollo de Sistemas Pervasivos

Para abordar el desarrollo de sistemas pervasivos guiados por modelos (MDD), se ha propuesto en nuestro centro de investigación PervML [23]. PervML sigue un paradigma MDD en el que podemos construir un modelo de un sistema de software que se puede transformar en un elemento real. El objetivo de este paradigma es traducir automáticamente una especificación abstracta del sistema en un producto de software completamente funcional. La Figura 3.1 presenta una vista general de PervML.

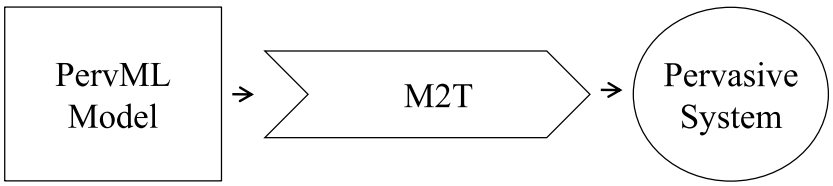


Fig. 3.1: Vista general del método PervML

A continuación, se describe brevemente cada elemento de la figura:

Pervasive Modeling Language (PevML) [24]: es un Lenguaje Específico de Dominio (DSL) para describir sistemas pervasivos

utilizando conceptos de alto nivel de abstracción. Este lenguaje se centra en especificar entornos concretos tales como los servicios de una casa inteligente. Estos servicios se pueden combinar para ofrecer una funcionalidad más compleja por medio de interacciones. Estos servicios también pueden iniciar la interacción como una reacción a cambios del entorno. Los principales conceptos de PervML son:

- **Servicio:** coordina la interacción entre los proveedores para llevar a cabo tareas específicas (estos proveedores pueden ser sistemas de hardware o software).
- **Binding Provider (BP):** vincula entre distintas tecnologías (por ejemplo, elementos del entorno físicos y lógicos).
- **Interacción:** es una descripción de un conjunto de invocaciones entre servicios.
- **Trigger:** es una regla ECA (Evento Condición Acción), que describe cómo un servicio reacciona a los cambios en su entorno.

Este DSL se ha aplicado para desarrollar soluciones en el dominio de los hogares inteligentes [25]. En la siguiente subsección describe en detalle este lenguaje.

Model To Text (M2T): una vez que el sistema pervasivo ha sido modelado, el motor de transformación se aplica para generar el código. Para esto, se utiliza el lenguaje MOFScript que ofrece la capacidad de navegar entre modelos, la creación de archivos, etc. MOFScript toma como entrada un modelo y aplica sobre un metaelemento seleccionado una regla contextual. La regla aplicada puede acceder a las propiedades del elemento, navegar por los elementos del modelo relacionados e invocar otras reglas. En [1] hay más información acerca de las reglas de transformación y las herramientas para apoyar la generación de código.

Pervasive System: una vez el motor de transformación ha generado el código, se obtienen un conjunto de proyectos que pueden ser importados en el IDE de Eclipse. Entonces, un desarrollador llevará a cabo una serie de tareas[1] para implantar el código en el sistema pervasivo resultante.

[1] `www.pros.upv.es/laboratorios/proyectos/pervml`

3.2.1 El lenguaje de modelado de PervML

PervML promueve la separación de roles en *Pervasive System Analysts* (en adelante analistas) y *Pervasive System Architects* (en adelante arquitectos) (ver Figura 3.2). Por un lado, los analistas capturan los requisitos y describen el sistema pervasivo utilizado la metáfora de servicios como la principal primitiva conceptual. La metáfora de servicios consiste en seleccionar elementos que agrupan funcionalidad y coordinan la interacción entre proveedores para llevar a cabo tareas específicas.

Los analistas especifican tres modelos que se muestran en la Figura 3.2: el modelo de servicios (*the Services Model)*, el modelo estructural *the Structural Model*, y el modelo de interacción *the Interaction Model*.

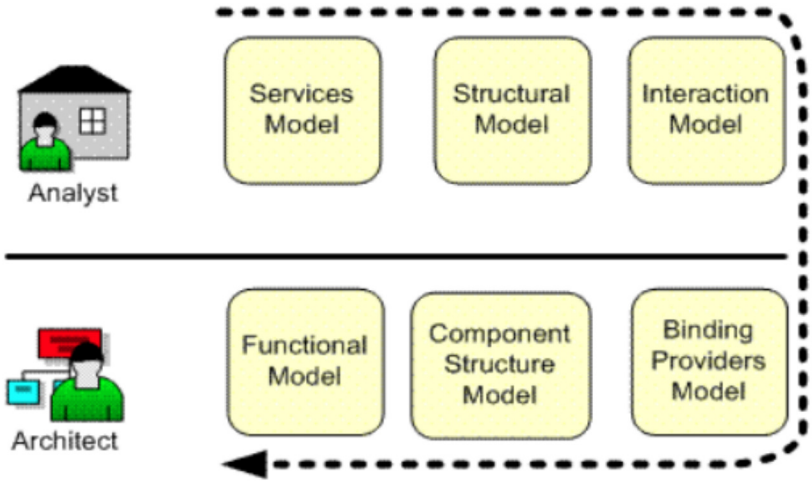


Fig. 3.2: Modelos PervML

El **modelo de servicios** describe los tipos de servicios que son proporcionados en el sistema. El diagrama que representa este modelo en la Figura 3.3 muestra que el sistema proporciona servicios para el control de la iluminación, para gestionar la seguridad, etc. Además de la información que se muestra en la Figura 3.3, la descripción de un tipo de servicio incluye (1) el funcionamiento y las pre y post condiciones (se expresan mediante el *Object Constraint Language* (OCL)) para cada operación, (2) un Protocolo de Máquina de Estados (que indica las

operaciones que pueden ser invocadas en un momento determinado), y (3) *triggers* (permiten especificar el comportamiento dinámico de los servicios).

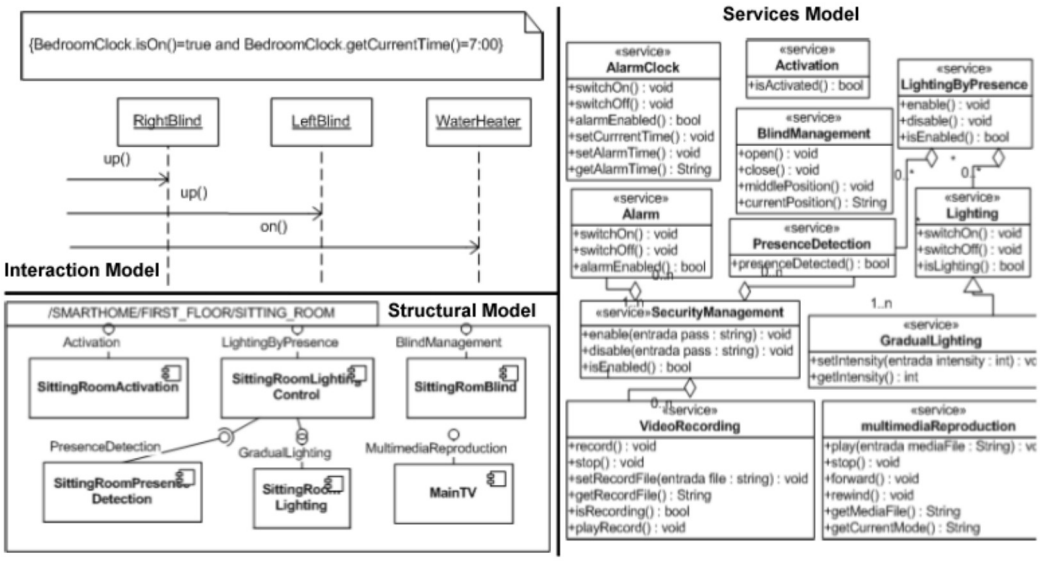


Fig. 3.3: Vista parcial de los modelos descritos por el analista

El **modelo estructural** indica las instancias de cada tipo de servicio que son proporcionadas por el sistema en cada localización del entorno. Estas instancias se llaman componentes porque son representadas como componentes de UML 2.0 y el tipo de servicio que cada uno proporciona está representado como una interfaz. Por ejemplo, la Figura 3.3 presenta el componente *SittingRoomBlind* que proporciona el servicio *BlindManagement*. Las relaciones de dependencia entre componentes, pueden ser incluidas para especificar que un componente utiliza la funcionalidad proporcionada por otro. Con el fin de no sobrecargar esta figura, sólo se muestran los componentes de la sala de estar (*sitting room*). Por ejemplo, se han definido varios componentes tales como: el *SittingRoomLightingControl* que utiliza la funcionalidad proporcionada por el *SittingRoomPresenceDetection* y el componente *SittingRoomLighting*, definidos en el modelo también.

El **modelo de interacción** especifica la comunicación que se

produce como una reacción a algún evento del sistema. Cada interacción se describe por un conjunto de diagramas de secuencia UML 2.0. Así, el analista identifica los componentes del modelo estructural que participan en la interacción, define los mensajes que los componentes deben de intercambiar y especifica la condición que activa los *triggers* mediante OCL. Las acciones descritas en el diagrama se ejecutarán cuando se cumpla la condición. Figura 3.3 presenta la interacción que se encarga de abrir las persianas y encender el calentador de agua cuando la alarma se apaga a las 7.00 a.m.

Por otro lado, los arquitectos especifican los dispositivos y/o sistemas software existentes soportan los servicios del sistema. Estos elementos (dispositivos y sistemas de software) son referidos como *binding providers* porque unen el sistema pervasivo con su entorno físico o lógico. Los arquitectos especifican tres modelos (*the Binding Providers Model, the Component Structure Model, and the Functional Model*), que se muestran en la Figura 3.4.

El ***Binding Providers Model*** describe los distintos tipos de dispositivos de que se utilizan en el sistema. La Figura 3.4 presenta algunos de los *binding providers* para un hogar inteligente. Por ejemplo, el diagrama especifica que el sensor de movimiento (*MovementDetector*) proporciona una operación (*movementDetected()*) para saber si se detecta algún movimiento.

El ***Component Structure Model*** se utiliza para asignar dispositivos y sistemas software a los componentes del sistema. Por ejemplo, el *SittingRoomLighting* utiliza los dispositivos *(ST_GL1)* y *ST_GL2* que son instancias del *Binding Provider*: *GradualLamp*. Además, el mismo dispositivo puede utilizarse para distintos componentes.

El ***Functional Model*** especifica las acciones que son ejecutadas cuando una operación de un servicio se invoca. Estas acciones se especifican utilizando el *Action Semantics Language* (ASL) de UML. Cada operación proporcionada para cada componente, tiene que tener asociada una especificación funcional. La Figura 3.4 presenta las acciones que son ejecutadas cuando la operación *open()* del componente *SittingRoomBlind* se invoca.

Una vez que el sistema pervasivo ha sido descrito utilizando los mod-

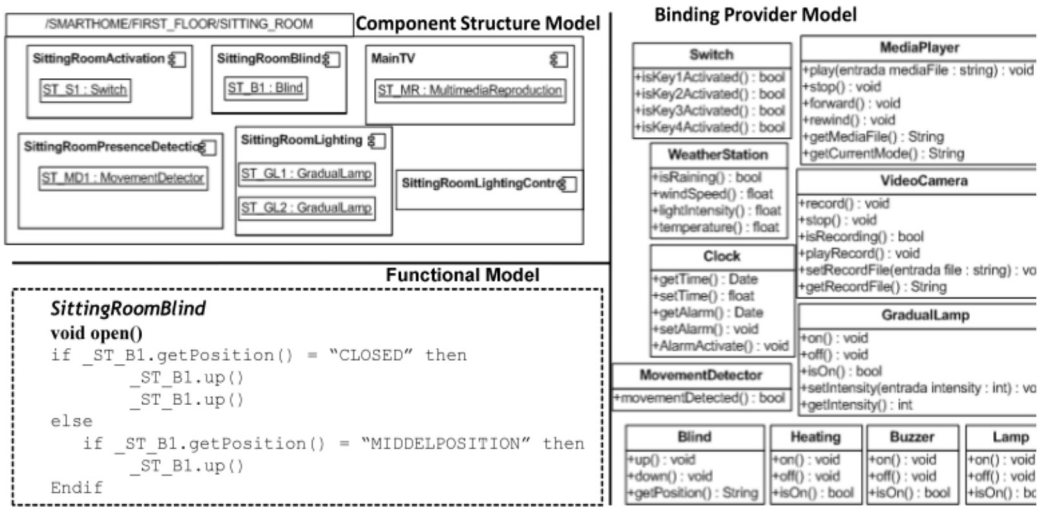


Fig. 3.4: Vista parcial de los modelos descritos por el arquitecto

elos PervML, se puede obtener automáticamente una implementación del sistema pervasivo en código Java aplicando tranformaciones de modelos [23].

3.3 Línea de Productos Software

Una línea de productos de software (SPL) [26] es un conjunto de sistemas software que comparten y gestionan un conjunto de características comunes para satisfacer las necesidades específicas de un segmento determinado de mercado o misión y, que ha sido desarrollada a partir de un conjunto de activos. Las características son unidades (por ejemplo, incrementan la funcionalidad de la aplicación) por las cuales los productos se pueden definir y distinguir dentro de una SPL.

En resumen, una SPL consiste en que a partir de unos activos de entrada, un producto se transforma en un sistema de salida de acuerdo a la configuración especificada en un modelo de decisión. La Figura 3.5 presenta la vista general de una SPL.

Las SPLs están emergiendo rápidamente como un paradigma

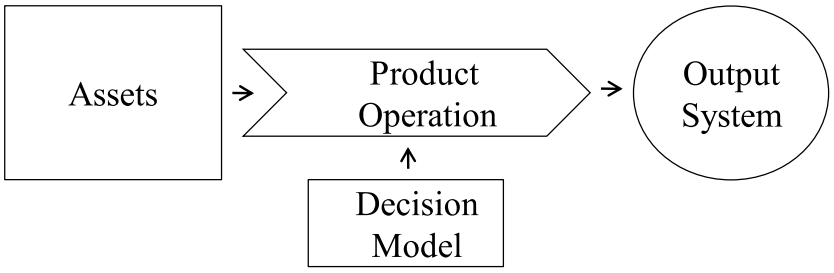


Fig. 3.5: Vista general de una Línea de Productos Software

alternativo, viable e importante de desarrollo de software, permitiendo a las empresas obtener mejoras en el tiempo de comercialización, costes, productividad y calidad. Una SPL también puede permitir una rápida entrada a un mercado, una respuesta flexible y proporcionar capacidad de personalización en masa.

3.4 MDDSPL: una Línea de Productos guiada por Modelos para el Desarrollo de Sistemas Reconfigurables

El método de desarrollo de sistemas reconfigurables desarrollado en nuestro centro de investigación [27] propone un método de Línea de Productos Software (SPL) para desarrollar dinámicamente sistemas pervasivos adaptables de forma sistemática. Este método permite que los sistemas pervasivos hagan uso en run-time de los mismos modelos con los que fueron diseñados.

El método aplica dos ideas principales: (1) modelado colectivo en lugar de modelado individual, y (2) la aplicación de arquitecturas de líneas de productos (PLAs) [28]. Este trabajo extiende la operación de producción de la SPL con el objetivo de aumentar los productos de la SPL con modelos de variabilidad y también, algunos activos adicionales que permitan la reconfiguración. Se hace uso de los modelos de la variabilidad y los recursos disponibles para encontrar la "mejor" reconfiguración del sistema software para lograr los objetivos de los usuarios. Estos modelos de variabilidad asisten la estrategia de

ejecución para determinar los pasos que son necesarios para reconfigurar el sistema de software. Entonces, la PLA se reconfigura rápidamente a la configuración deseada. El uso de modelos de la variabilidad en tiempo de ejecución permite que el sistema pervasivo decida de forma dinámica cómo alcanzar los objetivos del usuario de una manera efficiente.

Esta aproximación de adaptación se centra en dos escenarios de adaptación muy comunes en los sistemas pervasivos: la evolución (recurso añadido) y la involución (recurso eliminado). Estos escenarios tienen requisitos diferentes en cuanto a la adaptación, y la la forma en que los modelos son gestionados en tiempo de ejecución deben considerarse requisitos particulares. Una aproximación basada en modelos se introduce en el método con el objetivo de organizar los conocimientos necesarios para la adaptación, de acuerdo a las demandas específicas de cada escenario de adaptación. En escenarios de involución, utiliza modelos con conocimiento precalculado para proporcionar una respuesta autonómica en una cantidad de tiempo reducida.

Los modelos que constituyen la base para el comportamiento de adaptación están disponibles en tiempo de diseño así, se puede realizar un análisis exhaustivo de las especificaciones de los efectos de validación y verificación. Una vez realizado el análisis, se puede garantizar un comportamiento de adaptación determinista en tiempo de ejecución, que es esencial para los sistemas fiables.

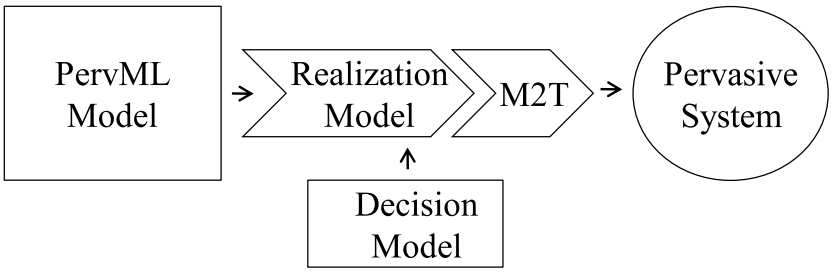


Fig. 3.6: Vista general del método MDD-SPL

La Figura 3.6 presenta una vista general del método MDD-SPL presentado en esta sección para desarrollar sistemas reconfigurables. Como puede observarse en la figura, el sistema pervasivo se modela

utilizando el lenguage específico de dominio **PervML** (ver Subsección 3.2.1). El **Modelo de decisión** representa el modelo de variabilidad donde se seleccionan características del sistema. El **modelo de realización** vincula las características del modelo de decisión con elementos de los modelos de PervML. Después, se realiza la transformación **modelo a texto** y se obtiene el **sistema pervasivo**.

3.5 Conclusiones

En este capítulo se han introducido conceptos relacionados con el trabajo presentado en esta tesis de máster (los sistemas pervasivos y Línea de Productos Sofware).

Además, se han presentado dos métodos desarrollados en nuestro centro de investigación: PervML y MDDSPL. PervML permite el desarrollo de sistemas pervasivos a partir de modelos que especifica un Analista y un Arquitecto. Este método aporta beneficios como son: productividad, calidad, adaptabilidad y mantenimiento. El método MDDSPL permite desarrollar sistemas pervasivos a partir de modelos que se utilizan también en tiempo de ejecución para reconfigurar el sistema. Este método aporta los mismos beneficios que PervML (productividad, calidad, adaptabilidad y mantenimiento) además de permitir la reconfiguración del sistema en tiempo de ejecución.

Pese a los beneficios que aportan los métodos PervML y MDDSPL para el desarrollo de sistemas pervasivos, los usuarios finales no pueden participar en ellos.

Capítulo 4

Introduciendo a los Usuarios Finales en el Desarrollo de Sistemas Pervasivos

Los métodos PervML y MDDSPL presentados anteriormente aportan diversas ventajas al desarrollo de sistemas pervasivos. Sin embargo, los usuarios finales no pueden participar desde las etapas tempranas del proceso de desarrollo en estos métodos. Una de las razones es la complejidad de describir su sistema pervasivo mediante modelos. Este capítulo razona por qué los usuarios finales no pueden participar en estos métodos, analiza los desafíos para introducirlos en estos métodos y plantea una aproximación para introducir a los usuarios finales en cada método.

El resto de este capítulo está estructurado en las siguientes secciones: la Sección 4.1 presenta los desafíos para introducir a los usuarios en los métodos PervML y MDDSPL. La Sección 4.2 propone una aproximación metodólogica para introducir a los usuarios en PervML. La Sección 4.3 presenta una aproximación metodológica para introducir a los usuarios en el método MDDSPL. La Sección 4.4 concluye el capítulo.

4.1 Desafíos

Introducir a los usuarios finales en las etapas tempranas del proceso de desarrollo permite proporcionar un sistema que encaje perfectamente con las necesidades de los usuarios finales. Para ello, se deben proporcionar a los usuarios finales mecanismos y herramientas que permitan describir su sistema de una forma familiar [17] para ellos, encontrando una forma de programación que sea más natural y que imite las prácticas más familiares, sin la necesidad de que los usuarios sigan un conjunto de secuencias lógicas y acciones estrictas.

Los métodos presentados anteriormente: PervML y MDDSPL (ver Sección 3.2 y Sección 3.4 respectivamente) aportan diversas ventajas para desarrollar sistemas pervasivos o sistemas pervasivos reconfigurables (ver Sección 3.5). Sin embargo, los usuarios finales no pueden participar desde las etapas tempranas del proceso de desarrollo en estos métodos. Las razones más destacadas son las siguientes:

- Es posible que los usuarios finales no estén familiarizados con el dominio de la casa inteligente. Por ejemplo, los usuarios finales pueden no conocer las dependencias entre un servicio y sus dispositivos o, configuraciones.

- No están familiarizados en especificar modelos para describir su sistema pervasivo.

- No están familiarizados con las primitivas conceptuales de PervML utilizadas en los modelos (Servicio, Trigger, interacción, etc.).

- Los métodos descritos anteriormente, están diseñados para requerir dos profesionales de software para especificar los modelos del sistema (analista y arquitecto).

Las siguientes subsecciones describen en detalle los desafíos para introducir a los usuarios finales en los métodos PervML y MDDSPL.

4.1.1 PervML

Introducir a los usuarios finales en PervML aporta diversos desafíos tanto para los usuarios finales como para el método. Los usuarios finales pueden no tener conocimientos sobre el dominio de las casas inteligentes, no estar familiarizados en especificar su sistema utilizando modelos y pueden no conocer las primitivas conceptuales que proporciona PervML mientras que, PervML está diseñado para requerir dos profesionales de software para especificar los modelos del sistema (roles Analista y Arquitecto descritos en la Sección 3.2.1).

La Fig. 4.1 ilustra dos etapas donde es interesante introducir a los usuarios finales en PervML. Estos son: los modelos de PervML y en el sistema pervasivo.

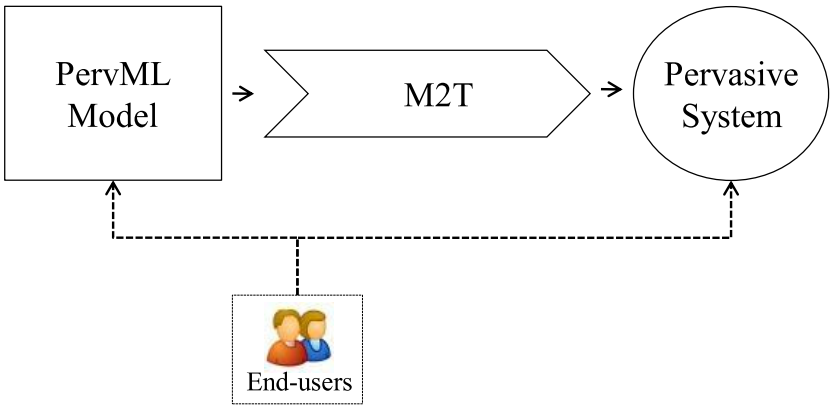


Fig. 4.1: Introducción usuarios finales en PervML

Introducir a los usuarios finales en los modelos de PervML, mediante mecanismos basados en técnicas *End-user Development*, permite que éstos puedan especificar el sistema ellos mismos de acuerdo a sus necesidades y obtener así, un sistema que se ajuste a sus expectativas. Sin embargo, esta introducción de los usuarios finales en los modelos de PervML supone los siguientes **desafíos**:

- Cómo ofrecer a los usuarios finales mecanismos que permitan describir de manera sencilla y que sea suficientemente expresiva.

- Construir una herramienta basada en técnicas *End-user Development* para permitir la especificación de los modelos necesarios para obtener la descripción del sistema.

- Familiarizar a los usuarios finales con el dominio y sus primitivas de dominio (servicios, *binding providers*, dispositivos, *triggers*, etc.).

También resulta interesante introducir a los usuarios finales en la personalización del sistema en tiempo de ejecución (*run-time*). Así, ellos mismos pueden adaptar el sistema frente a nuevas necesidades o preferencias. Sin embargo, PervML no está diseñado para la reconfiguración de sistema en run-time: sólo es posible modificar las variables de los servicios que están disponibles en el sistema (por ejemplo, modificar la intensidad de un servicio de iluminación gradual) pero, no es posible crear o modificar las interacciones entre servicios en run-time. Esto resulta interesante dada la naturaleza dinámica de este tipo de sistemas.

4.1.2 MDDSPL

Del mismo modo que ocurre para introducir a los usuarios finales en PervML, introducir a los usuarios finales en MDDSPL aporta diversos desafíos tanto para los usuarios finales como para el método. Los usuarios pueden no tener conocimientos sobre el dominio de la casa inteligente o, no estar familiarizados en especificar su sistema utilizando modelos.

La Fig. 4.2 ilustra dos etapas donde es interesante introducir a los usuarios finales en el método MDDSPL. Estos son: en el modelo de decisión y en el sistema pervasivo.

Para introducir a los usuarios finales en el modelo de decisión, se propone que los usuarios puedan seleccionar las características deseadas de su sistema activando o desactivando características del modelo. Esto supone los siguientes **desafíos**:

- Permitir a los usuarios finales seleccionar las características de su sistema de una forma que resulte familiar.

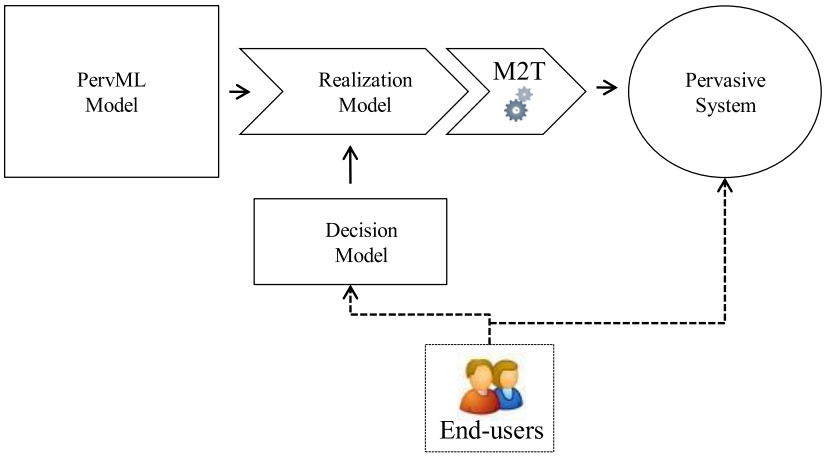


Fig. 4.2: Introducción usuarios finales en MDDSPL

- Construir una herramienta basada en técnicas *End-user Development* para permitir la especificación de los modelos necesarios para obtener la descripción del sistema.
- Cómo ofrecer mecanismos para integrar la descripción del sistema con los modelos de PervML y poder generar el sistema pervasivo deseado a partir de modelos.

Las **ventajas** de la introducción de los usuarios finales en el método MDDSPL respecto a introducirlos en PervML son las siguientes:

- El sistema no tiene que ser desarrollado desde cero para cada usuario. Esto implica más reutilización, menos tiempo de desarrollo y menos costes [29].
- Más sencillo y familiar para los usuarios finales seleccionar características disponibles en un catálogo que definir desde cero su sistema. Por ejemplo, para la compra de un coche el usuario selecciona características en un catálogo no define o diseña cada característica del coche.

También es interesante introducir a los usuarios finales en la personalización del sistema pervasivo en tiempo de ejecución (*run-time*).

Así, ellos mismos pueden adaptar su sistema a nuevas necesidades o preferencias. El objetivo más importante de introducir a los usuarios finales en run-time es proporcionar a los usuarios mecanismos y herramientas que permitan personalizar su sistema de una forma que les resulte familiar y segura (por ej. que el usuario no pueda definir una configuración que pueda poner en riesgo a otro usuario o al sistema). Una vez que los usuarios definen una nueva configuración en run-time, otro objetivo es mapear esa información que proporciona el usuario a los modelos del método para llevar a cabo la reconfiguración.

En conclusión, los objetivos para introducir a los usuarios en el método MDDSPL afectan a las siguientes actividades del proceso de desarrollo: elicitación de requisitos, diseño, implementación y mantenimiento. Las siguientes subsecciones describen en detalle los objetivos para cada una de estas actividades.

Elicitación de requisitos

Como se ha introducido anteriormente, un objetivo de este método MDDSPL es que los usuarios finales puedan describir los requisitos de su sistema de una forma que les resulte familiar.

En la actualidad, han surgido problemas relacionados con la actividad de elicitación de requisitos. Estos problemas se han presentado en trabajos como [30]. Uno de los problemas más importantes es el relacionado con los **problemas de entendimiento**. Estos problemas son resultado de la implicación necesaria de analistas, diseñadores, desarrolladores y usuarios finales en la obtención de requisitos. Los requisitos son producidos e interpretados por personas con diferentes niveles de experiencia y antecedentes. Por ejemplo, los usuarios finales no entienden la jerga de los desarrolladores de software y los desarrolladores a menudo no entienden la jerga de los usuarios finales [31]. La forma en que los requisitos se expresan y, el tamaño del sistema descrito, afectan también a la comprensión. Si los participantes en la obtención no entienden la salida del proceso correctamente, entonces los requisitos resultantes pueden ser ambiguos, incoherentes e incompletos. Los requisitos pueden ser también incorrectos o pueden no abordar las necesidades reales de los usuarios finales. Además, en esta actividad,

los requisitos del sistema son descubiertos a través de consultas con los usuarios, a partir de documentos del sistema, el conocimiento del dominio, y estudios de mercado. Aunque hay una serie de métodos y técnicas para elicitar requisitos [32], éstos siguen siendo capturados, en la mayoría de los casos, mediante sólo el uso de entrevistas tradicionales [33]. Sin embargo, las entrevistas tradicionales no siempre son la mejor opción para la extracción de las necesidades del usuario [34].

Por tanto, los **objetivos de esta actividad de elicitación de requisitos** son: (1) proporcionar a los usuarios finales mecanismos y herramientas para describir los requisitos de su sistema de manera que estén familiarizados, (2) minimizar la diferencia entre las necesidades del usuario y el sistema desarrollado y (3) obtener una descripción del sistema que pueda ser integrada con el método MDDSPL para implementar el sistema que se adapte perfectamente a los requisitos de los usuarios finales.

Diseño e implementación

En las actividades de diseño e implementación, el objetivo más importante es integrar las descripciones del sistema de los usuarios finales en el método MDDSPL.

Las principales razones para integrar la descripción del sistema que realiza el usuario en el método de MDDSPL, son nuestro amplio conocimiento de este método y la generación de código que este método proporciona. Además, la integración de estas actividades ofrece muchos beneficios que son históricamente conocidos en la comunidad de ingeniería de software. Los más importantes son los siguientes:

- **Productividad:** La herramienta de generación de código que proporciona el método MDDSPL construye código software en una fracción de tiempo inferior comparada con la que necesita un ingeniero software. Además, el código escrito a mano tiende a tener un gran número de errores de sintaxis que los ingenieros deben detectar y corregir desperdiciando tiempo. Al seguir una estrategia MDD, el código es generado sistemáticamente por una herramienta libre de errores.

- **Calidad:** Los grandes volúmenes de código a mano tienden a tener una calidad inconsistente porque los ingenieros tienen nuevos o mejores enfoques mientras trabajan. El código generado aumenta la calidad y disminuye el tiempo de desarrollo porque cuando los errores o deficiencias se encuentran, pueden fijarse en la herramienta de generación de código y aplicarse automáticamente a las siguientes aplicaciones.

- **Adaptabilidad:** Siguiendo una estrategia MDD, un producto software es desarrollado de forma independiente a las tecnologías de aplicación. Este producto software es desarrollado por la descripción en un modelo. Para obtener el código en una tecnología determinada, se utiliza una herramienta de generación de código. Si surgen cambios en la tecnología de implementación, la herramienta de generación de código puede adaptarse a los cambios, en vez de aplicarlos directamente sobre las implementaciones ya realizadas. Así, no es necesario modificar las aplicaciones desarrolladas ya que están abstractamente descritas por modelos.

- **Mantenimiento:** El mantenimiento del código escrito a mano es con frecuencia una tarea muy tediosa: un cambio muy simple en los requisitos de una aplicación requiere ingenieros para modificar una gran cantidad de artefactos software, tales como: interfaces de usuario, clases lógicas, tablas de la base de datos y documentación. Siguiendo una estrategia MDD, sólo los modelos deben ser modificados. Los cambios a nivel de aplicación se realizan automáticamente.

Mantenimiento

En la actividad de mantenimiento los grandes objetivos son los siguientes:

- Permitir a los usuarios personalizar su sistema cuando sus necesidades o preferencias cambian de una forma que sea familiar.

- Adaptar el sistema cuando las necesidades o preferencias del

usuario cambian. Por adaptación nos referimos a un cambio de una configuración a otra para satisfacer a un cambio en las necesidades del usuario.

Como hemos presentado anteriormente, técnicas centradas en el usuario o técnicas de programación *end-user* pueden ser utilizadas para incorporar la personalización del sistema desde etapas tempranas del proceso de desarrollo. Sin embargo, hemos observado que hay factores importantes que no se conocen hasta que el sistema se utiliza. Por ejemplo, durante el desarrollo de una sala de reuniones [25], el sistema se fija con unos valores iniciales pero cuando los usuarios finales utilizan el sistema, los usuarios cambian constantemente la configuración debido a que la configuración inicial no satisfacía sus necesidades o ellos necesitaban cambiarla.

Por tanto, comenzamos **analizando el nivel de adaptación del sistema** que es necesario para permitir que el sistema satisfaga adecuadamente las necesidades del usuario final [35]. Este análisis se ha basado en las siguientes ideas:

- La adaptación en casas inteligentes es impulsado por una influencia cíclica entre las necesidades del usuario y las capacidades del sistema [36].

- La adaptación intenta evitar el desajuste entre las necesidades del usuario y las capacidades del sistema [37].

- La adaptación de un sistema está limitada por un punto en el que tantos cambios se han realizado que en realidad estamos hablando de otro sistema [10].

Los problemas detectados en esta actividad de mantenimiento son: (1) proporionar a los usuarios finales los mecanismos para personalizar su sistema cuando sus necesidades cambien de una manera familiar para ellos, y (2) adaptar estas personalizaciones cuando el sistema está en tiempo de ejecución requiere un enorme esfuerzo para gestionar la complejidad de adaptación.

4.2 Introduciendo a los Usuarios en PervML

Esta sección plantea cómo introducir a los usuarios finales en las etapas tempranas del proceso de desarrollo en el método PervML presentado anteriormente (ver Sección 3.2). En concreto, esta sección plantea cómo los usuarios finales pueden participar en la definición de los modelos de PervML que describen el sistema.

4.2.1 Visión General. Esquema *Software Factory*

Un esquema *Software Factory* categoriza y resume los artefactos utilizados para construir y mantener el sistema. Contiene un gráfico de puntos de vista donde cada uno define una parte del sistema con un nivel específco de abstracción y con un punto específico en el ciclo de vida software [38; 39]. La Figura 4.3 muestra el esquema *Software Factory* propuesto para el desarrollo de sistemas pervasivos.

En la figura, puede observarse que el esquema define cinco puntos de vista diferentes para el sistema pervasivo. La vista superior representada en la figura, corresponde a la vista de usuarios finales y el resto corresponen a las vistas definidas previamente en PervML [23]: Analista (*Analyst*), Arquitecto (*Architect*), Código fuente (*Source code*) y, Despliegue (*Deployment*).

Nuestra propuesta consiste en identificar dónde pueden participar en el método los usuarios finales y cómo interactuan con el resto de vistas de PerVML.

En resumen, el usuario final, el analista y arquitecto describen el sistema mediante el lenguaje especifico de dominio PervML (ver Sección 3.2.1). En el caso de los usuarios finales, se les permitirá introducir su especificación mediante mecanismos y herramientas basadas en técnicas *End-user Development*. Las especificaciones realizadas en estos mecanismos y herramientas se mapean a los modelos de PervML (modelo de servicios, estructural e interacción) mientas que los analistas y arquitectos introducen su especificación mediante una herramienta. De esta descripción, un motor de transformación genera código. En paralelo, un desarrollador de software, crea software para hacer frente a

las dependencias tecnológicas que deben tenerse en cuenta para acceder a los dispositivos que implementan la funcionalidad del sistema. Por último, todo el código es compilado, empaquetado, y desplegado en un servidor con un *framework* de implementación.

A continuación, los puntos de vista representados en la Fig. 4.3 son brevemente introducidos. Una completa descripción de cada vista queda fuera del ámbito de este trabajo y se plantea describir en detalle el punto de vista end-user y cómo interactua con los otros puntos de vista de PervML en trabajos futuros.

- **System End-user view:** el usuario final describe el sistema pervasivo con un alto nivel de abstracción utilizando la metáfora de servicio que consiste en seleccionar elementos que agrupan funcionalidad y coordinan la interacción entre proveedores para llevar a cabo tareas específicas y, utilizando técnicas *End-user Development* tales como: (1) *Visual Programming* porque busca una forma de programación natural y que imite las prácticas más familiares sin que los usuarios sigan un conjunto de secuencias lógicas y acciones estrictas. (2) *Programming By example* porque permite especificar funcionalidad a partir de ejemplos en el sistema y (3) la metáfora de las piezas de puzle porque se basa en la familiaridad de ensamblar piezas de puzle juntas para especificar funcionalidad.

  Esta vista persigue que el usuario final describa la funcionalidad requerida en el sistema. Mediante las técnicas *End-user Development* se construirán tres modelos gráficos en el lenguaje PervML: el *Services Model,* el *Structural Model* y el *Interaction Model.* En estos modelos, el usuario final describe (1) los diferentes tipos de servicios, (2) la ubicación de cada servicio en el sistema y (3) cómo interactúan unos servicios con otros. El usuario puede reutilizar las descripciones de los tipos de servicios de proyectos anteriores, ya que probablemente sean similares de proyecto a proyecto.

- **Pervasive System Analyst view:** el analista describe el sistema pervasivo con un nivel alto de abstracción utilizando la metáfora de servicio. Esta vista persigue que el analista complete

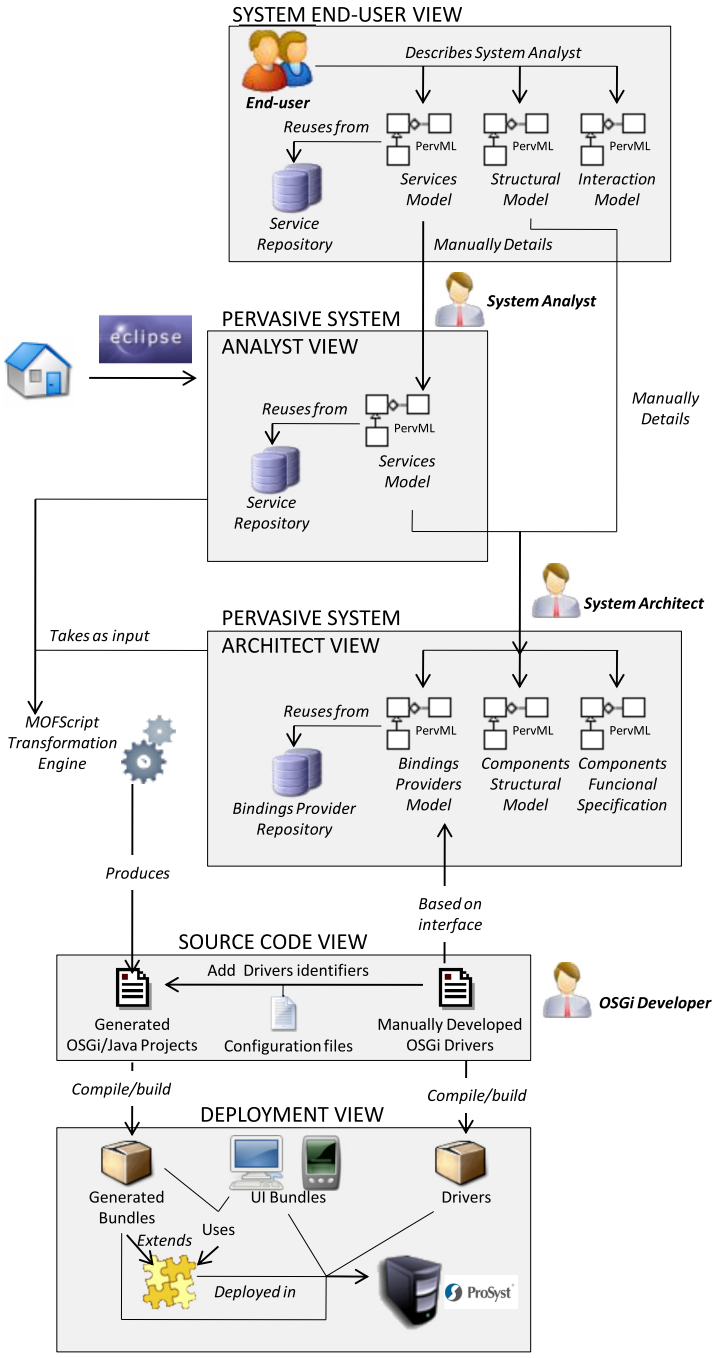


Fig. 4.3: Introducción usuarios finales en PervML

mediante detalles manuales la especificación de la funcionalidad requerida por los usuarios finales en el modelo de servicios. En este modelo, el analista describe (1) las pre y post condiciones de cada tipo de servicio, (2) la *Protocol state machine* por cada tipo de servicio para especificar qué operaciones de servicios son invocadas en un momento concreto. Los analistas pueden reutilizar las descripciones de los tipos de servicios de proyectos anteriores, ya que probablemente sean similares de proyecto a proyecto.

- **Pervasive System Architect view:** el arquitecto selecciona el tipo y número de dispositivos o sistemas software que son más adecuados para implementar los servicios especificados por el usuario final y el analista. La selección podría tener en cuenta razones económicas o limitaciones en el entorno físico del sistema. En PervML se denomina *Binding Provider* a los elementos que son responsables de enlazar el sistema físico con el entorno lógico. Por ejemplo, un sensor de iluminación está a cargo de la medición de la temperatura ambiente del entorno físico, mientras que un servidor de correo electrónico permite el envío de información a los agentes que están fuera del alcance del sistema (entorno lógico). El arquitecto construye otros tres tipos de especificaciones utilizando PervML: el *Binding Providers Model,* el *Component Structural Specification* y el *Components Functional Specification.* Para lograr este objetivo, el arquitecto describe (1) cada tipo de *Binding Provider* (sus interfaces y sus relaciones), (2) los *Binding Provider* que son utilizados por cada servicio del sistema y, (3) cuyas acciones deben ser ejecutadas cuando una operación de un componente se invoca.

- **Source Code view:** el código fuente maneja tres tipos de artefactos: (1) Proyectos OSGi/Java de Eclipse que son automáticamente generados como resultado de aplicar el motor de transformación a la especificación PervML. (2) *Drivers* para gestionar los dispositivos o sistemas software que son seleccionados por el arquitecto son implementeados por el desarrollador OSGi. Estos *drivers* proporcionan acceso al *framework* basado en OSGi para los dispositivos o sistemas de software externos. Deben

de ser desarrollados a mano porque su desarrollo depende de la dependencias tecnológicas. Si cualquier dispositivo o sistema de software externo ha sido utilizado previamente en el sistema, el mismo *driver* puede ser reutilizado. (3) Archivos de configuración que configuran los proyectos OSGi/Java para utilizar los *drivers* seleccionados. Esta configuración sólo implica indicar los identificadores de los *drivers*.

- **Deployment view:** por último, los proyectos de código fuente son compilados y ejecutados en bundles (archivos JAR). Estos bundles son desplegados en un servidor OSGi con un *framework* de implementación y los *drivers*. El *framework* implementa la estrategia de ejecución de PervML que proporciona clases abstractas a las primitivas conceptuales de PervML (*Service, Trigger, Interaction, etc.*) para facilitar el proceso de traducción. *Bundles* adicionales pueden proporcionar interfaces de usuario que tienen que ser desplegados en el servidor OSGi.

4.2.2 Ámbito

Existen muchos tipos de sistemas que son considerados sistemas pervasivos. Asistentes móviles a los visitantes de los museos [40], campus virtuales que vinculan la realidad con datos digitales [41] o entornos comerciales que sugieren listas de compra [42].

Para aplicar el esquema *Software Factory* presentado en la sección anterior, se debe describir el tipo de sistemas a desarrollar [23]. En resumen,

> la aproximación metodológica introducida en esta Sección 4.2, asistirá a los usuarios finales en el desarrollo de sistemas que proporcionan servicios heterogeneos a los usuarios en un entorno físico.

Estos servicios serán implementados con dispositivos que estarán localizados en el entorno físico y por componentes software embebidos en unidades de computación. Mediante estos elementos, el sistema proveerá la funcionalidad requerida por los usuarios en el entorno físico

que viven. Esta funcionalidad tiene que ser soportada por una petición explicita del usuario y por una reacción implícita de las acciones de los usuarios o cambios en el entorno (*context-awareness*).

En [23] se describe una lista no exhaustiva de los sistemas soportados por PervML. Son los siguientes: una casa inteligente, una sala de reuniones, un parking, un hospital y una fábrica. **En esta tesis de máster el desarrollo de sistemas pervasivos se centra en el ámbito de la casa inteligente**. La casa inteligente es el sistema que proporciona servicios para controlar y vigilar las luces de la casa, persianas, TV, etc. También se notifican por varios canales (SMS, mensajería instantánea, correo electrónico, etc) intrusiones, inundaciones, etc. El entorno puede configurarse cuando la familia está viendo una película (luces suaves, persianas bajas, etc.).

Para desarrollar este tipo de sistemas, PervML se basa en una idea clave [23]:

> **Los usuarios de este tipo de sistemas pervasivos requieren servicios y ellos no quieren preocuparse de cómo los servicios se proporcionan.**

Por tanto, **el sistema debe soportar mecanismos de abstracción que expongan la funcionalidad del sistema como es esperada por los usuarios, independientemente de cómo esta funcionalidad esté internamente implementada.**

De esta idea clave, se puede concluir en que debe existir un software integrador que integre los elementos externos (dispositivos o sistemas software) con el fin de prestar los servicios que requieren los usuarios y ofrecer mecanismos de abstracción que permitan a los usuarios definir su sistema de la forma que ellos esperan. Además, el desarrollo de software puede implicar la utilización de distintas tecnologías (control, multimedia y redes de datos). Entonces, debe existir una tecnología que las integre. Sin embargo, un sistema pervasivo no es sólo un sistema integrador. También debe proporcionar una interfaz de usuario, que permita adaptar cambios de contexto.

4.2.3 Estrategia de Introducción

Una vez que el esquema *Software Factory* y el ámbito del sistema han sido introducidos, es importante especificar las carácteristicas que van a ser soportadas por el método y cómo los usuarios finales participan en la especificación de las mismas. En la Sección 3.1, Sistemas Pervasivos se presentan características en diferentes áreas del ámbito de los sistemas pervasivos como pueden ser: sensibilidad al contexto, interfaces multimodales y descubrimiento dinámico de servicios. Estos son sólo unos pocos ejemplos de características de ámbito de los sistemas pervasivos.

Para seleccionar una estrategia de introducción de los usuarios finales en PervML, nos vamos a centrar ofrecer a los usuarios finales mecanismos y técnicas que les permitan especificar las descripciones de su sistema (cómo se ha definido anteriormente) y utilizar el método PervML para la generación del sistema mediante los modelos especificados.

4.3 Introduciendo a los Usuarios en MDDSPL

Esta sección presenta cómo introducir a los usuarios finales en las etapas tempranas del proceso de desarrollo en el método MDDSPL presentado anteriormente (ver Sección 3.4). En concreto, esta sección plantea cómo los usuarios finales participan especificando la información que describe su sistema pervasivo reconfigurable.

4.3.1 Visión General. Esquema *Software Factory*

La Figura 4.4 presenta el esquema *Software Factory* propuesto para el desarrollo de sistemas pervasivos.

En la figura puede observarse que el esquema define seis puntos de vista diferentes para el sistema pervasivo. Las vistas superiores representadas en la figura, corresponden a las vistas de usuarios finales y el resto corresponen a las vistas definidas previamente en PervML [23]: Analista (*Analyst*), Arquitecto (*Architect*), Código fuente (*Source code*) y, Despliegue (*Deployment*).

Nuestra propuesta consiste en identificar dónde los usuarios finales pueden participar en el método MDDSPL y cómo interactuan con el resto de vistas.

En resumen, el analista y el arquitecto describen el sistema mediante el lenguaje especifico de dominio PervML (ver Sección 3.2.1) mientras que los usuarios finales describen las características de su sistema seleccionandolas en un catálogo de configuraciones predefinido por el analista mediante mecanismos y técnicas *End-user Development*. El analista describe el catálogo de configuraciones de acuerdo a las especificaciones de los modelos de PervML. Las selecciones de configuraciones que realizan los usuarios finales en el catálogo, se almacenan en el modelo de decisión. Cuando el usuario ha terminado de describir su sistema, el modelo de realización mapea los elementos del modelo de decision seleccionados a selecciones en los modelos de PervML (modelo de servicios, estructural e interacción) mientas que los analistas y arquitectos añaden detalles manuales a la especificación mediante una herramienta. De esta descripción, un motor de transformación genera código. En paralelo, un desarrollador de software, crea software para hacer frente a las dependencias tecnológicas que deben tenerse en cuenta para acceder a los dispositivos que implementan la funcionalidad del sistema. Por último, todo el código es compilado, empaquetado, y desplegado en un servidor con un *framework* de implementación.

A continuación, los puntos de vista representados en la Fig. 4.4 se introducen brevemente. Del mismo modo que en la Subsección 4.2.1, una completa descripción de todas las vistas queda fuera del ámbito de este trabajo. Esta tesis se centra en ofrecer una aproximación que soporte las vistas de los usuarios finales y cómo estos describen su sistema pervasivo (esta aproximación se describe en detalle en el Capítulo 5).

- **System End-user view:** el usuario final describe la caracterización de su entorno físico y selecciona características del catalogo de configuraciones disponibles con un alto nivel de abstracción utilizando la metáfora de servicio que consiste en seleccionar elementos que agrupan funcionalidad y coordinan la interacción entre proveedores para llevar a cabo tareas específicas

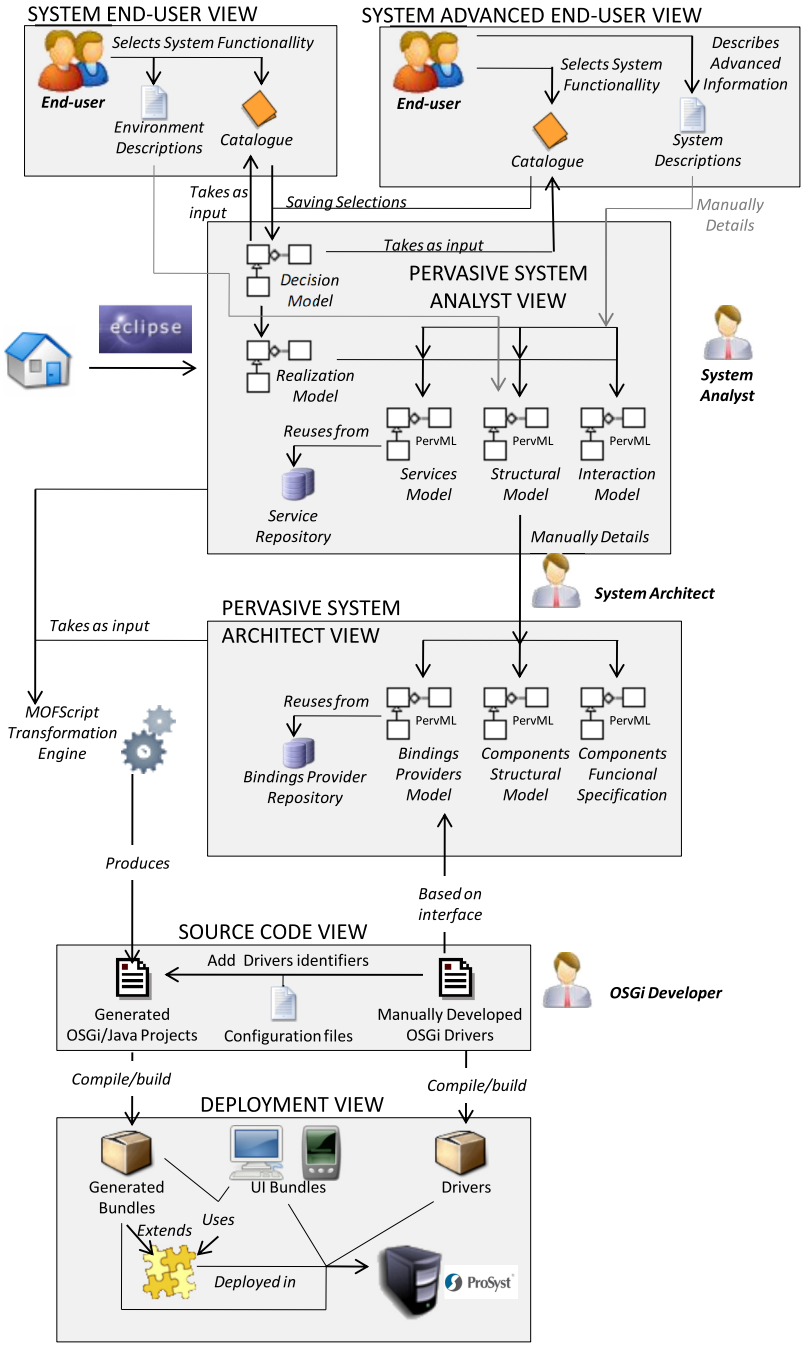


Fig. 4.4: Introducción usuarios finales en MDDSPL

y, utilizando técnicas *End-user Development* tales como: (1) *Visual Programming* porque busca una forma de programación natural y que imite las prácticas más familiares sin que los usuarios sigan un conjunto de secuencias lógicas y acciones estrictas. (2)Natural Programming porque tiene como objetivo hacer posible que los usuarios expresen sus ideas de la misma forma que las piensan y con una representación que les resulte familiar. (3) *Programming By example* porque permite especificar funcionalidad a partir de ejemplos en el sistema y (4) la metáfora de las piezas de puzle porque se basa en la familiaridad de ensamblar piezas de puzle juntas para especificar funcionalidad.

- Esta vista persigue que el usuario final describa la funcionalidad requerida del sistema en su entorno físico de forma que resulte familiar para él y sin necesidad de estar familiarizado con el dominio del hogar inteligente o primitivas del lenguaje PervML mediante representaciones visuales. Después, las configuraciones que realiza el usuario final se amacenan en el modelo de decisión y las descripciones de su entorno físico son incorporadas al modelo estructural mediante detalles manuales del analista.

- **System Advanced End-user view:** el usuario final avanzado describe la caracterización de su entorno físico, selecciona características del catalogo de configuraciones disponibles y puede describir nuevos servicios que no están soportados en el catálogo con un alto nivel de abstracción utilizando las metáforas y técnicas *End-user Development* descritas en la vista anterior: la metáfora de servicio, *Visual Programming, Natural Programming, Programming By example* y la metáfora de las piezas de puzle. Esta vista persigue que el usuario final avanzado describa la funcionalidad requerida en el sistema en su entorno físico de forma que resulte familiar para él. Después, las configuraciones que realiza el usuario final son amacenadas en el modelo de decisión y, las descripciones de su entorno físico así como la definición de nuevos servicios que no estén soportados en el catalogo, son incorporados a los tres modelos gráficos del lenguaje PervML: *the Services Model, the Structural Model and the Interaction Model*

mediante detalles manuales del analista.

- **Pervasive System Analyst view:** el analista describe el sistema pervasivo con un alto nivel de abstracción utilizando la metáfora de servicio. Esta vista persigue que el analista complete mediante detalles manuales la especificación de la funcionalidad requerida tanto por el usuario final como por el usuario final avanzado en los tres modelos gráficos en el lenguaje PervML: *the Services Model, the Structural Model and the Interaction Model.* Los analistas pueden reutilizar las descripciones de los tipos de servicios de proyectos anteriores, ya que probablemente sean similares de proyecto a proyecto. A partir de estas especificaciones en los modelos, los analistas definen el catálogo de configuraciones disponibles para los usuarios finales en el modelo de decisión y definen el modelo de realización.

El resto de vistas: Pervasive System Architect, Source Code y Deployment, se describen del mismo modo que en la Subsección 4.2.1.

4.3.2 Ámbito

El ámbito para introducir a los usuarios finales en el método MDDSPL es el mismo que el descrito en la Subsección 4.2.2. Esta aproximación se centra en el desarrollo de sistemas pervasivos para el ámbito de la **casa inteligente**.

Del mismo modo del ámbito para introducir a los usuarios finales en PervML, para introducirlos en el método MDDSPL **el sistema debe soportar mecanismos de abstracción que expongan la funcionalidad del sistema como es esperada por los usuarios, independientemente de cómo esta funcionalidad este internamente implementada.**

4.3.3 Estrategia de Introducción

Una vez que el esquema *Software Factory* y el ámbito del sistema ha sido introducido, es importante especificar las carácteristicas que van a

ser soportadas por el método y cómo los usuarios finales participan en la especificación de las mismas.

El Capítulo 5 describe en detalle la aproximación definida en esta tesis de máster para introducir a los usuarios finales en el método MDDSPL en las fases tempranas del proceso de desarrollo y sean ellos mismos quienes describan su sistema. Concretamente, describe en detalle los perfiles que soportan cada vista del esquema y el proceso que deben de seguir para especificar la descripción de su sistema.

4.4 Conclusiones

En este capítulo, se han analizado los objetivos para introducir a los usuarios finales en el desarrollo de sistemas pervasivos tanto para introducirlos en el método PervML como en el método MDDSPL, descritos en el Capítulo 3. Además, se han analizado en detalle los objetivos para las distintas actividades del proceso de desarrollo software para el método MDDSPL (Sección 4.1).

Para abordar los objetivos planteados, se han definido dos aproximaciones una para introducir a los usuarios en PervML (Sección 4.2) y otra para introducirlos en el método MDDSPL (Sección 4.3). En esta tesis de máster, dado el tiempo y esfuerzo, sólo vamos a plantear cómo interactúan los usuarios finales en PervML proponiendo como trabajo futuro la aplicación de la propuesta presentada en este capítulo. En esta tesis de máster nos hemos centrado en definir una aproximación para introducir a los usuarios finales en el método MDDSPL por las conclusiones presentadas en la Subsección 4.1.2. Por tanto, la aproximación desarrollada se centra en permitir a los usuarios finales describir su sistema mediante el uso de técnicas *End-user Development* y que estas descripciones sean utilizadas en el método MDDSPL para desarrollar un sistema pervasivo que se adapte a las necesidades y expectativas del usuario.

En conclusión, **esta tesis de máster aborda los objetivos para introducir a los usuarios finales en el método MDDSPL en las fases tempranas del proceso de desarrollo.** El siguiente capítulo describe en detalle cómo han sido abordados los objetivos presentados

en este capítulo.

Capítulo 5

Aproximación MDDSPL para Introducir a los Usuarios Finales en las Etapas Tempranas del Proceso de Desarrollo

Este capítulo presenta en detalle la aproximación propuesta para introducir a los usuarios finales en el método MDDSPL para participar en las fases tempranas en el proceso de desarrollo. Esta aproximación sigue el esquema *Software Factory* presentado en la Subsección 4.2.1 que categoriza y resume los artefactos utilizados para construir el sistema.

El resto de este capítulo está estructurado en las siguientes secciones: la Sección 5.1 presenta una vista general de la aproximación presentada. La Sección 5.2 presenta las características que deben definir los usuarios finales para describir su sistema pervasivo reconfigurable. La Sección 5.3 presenta los dos prefiles que son soportados en la aproximación: los

usuarios finales y los usuarios finales avanzados. La Sección 5.4 describe las interfaces que dan soporte las descripciones del usuario final y del usuario final avanzado. La Sección 5.5 presenta el proceso definido para guiar a los usuarios en la descripción de su sistema y definir cómo interactúan los usuarios finales con los analistas durante el proceso. Por último, la Sección 5.6 concluye el capítulo.

5.1 Vista General

La aproximación propuesta introduce a los usuarios finales en el método MDDSPL descrito en la Subsección 3.4. La Fig. 5.1 presenta la vista general de la aproximación. El *Realization Model* es el modelo que mapea las características seleccionadas en el *Decision Model* en elementos de los modelos de PervML. El *Decision Model* contiene el catálogo de configuraciones disponibles. El *End-user front-end* representa las interfaces que permiten a los usuarios finales obtener la descripción del sistema.

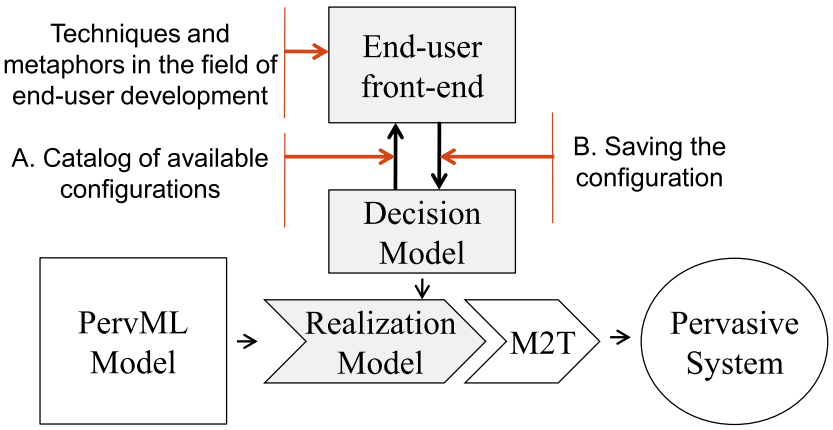


Fig. 5.1: Vista general aproximación

Las siguientes subsecciones se centran en describir el *End-user front-end*: qué información del sistema se debe obtener, las interfaces de usuario basadas en técnicas y metáforas *End-user Development*, los perfiles que soporta y, el proceso definido que permite a los usuarios finales crear la descripción del sistema.

5.2 Descripción del Sistema

Para cumplir con las características de los sistemas pervasivos [22] descritas en la Sección 3.1, los usuarios finales tienen que ser capaces de describir la siguiente información:

- El **entorno físico** donde se despliega el sistema así como, las **localizaciones** que forman el entorno.
- Los **usuarios** que pueden utilizar y personalizar el sistema y, un conjunto de **políticas** para cada usuario. Una política determina el conjunto de servicios o configuraciones disponibles para un usuario.
- Los **servicios** que el sistema tiene que proporcionar en las diferentes localizaciones del entorno.
- Los **dispositivos** que están disponibles en cada localización. Los dispositivos pueden ser utilizados para proporcionar funcionalidad de servicios. La información que describe que dispositivos son utilizados para proporcionar un servicio, se conoce como la configuración de un servicio.

Para especificar esta información, los usuarios finales deberán especificar alguna información desde cero (por ejemplo, la descripción del entorno, las localizaciones, los usuarios y las políticas). Para almacenar esta información utilizamos un repositorio. Para almacenar la configuración de servicios y dispositivos requeridos por el usuario en el entorno, proporcionamos un catálogo de configuraciones disponibles.

El catálogo de configuraciones disponibles surge ante la necesidad de ofrecer a los usuarios mecanismos para que puedan especificar la información de su sistema de forma que les resulte familiar. Como hemos explicado en secciones anteriores, los usuarios finales y los usuarios finales avanzados no tienen porqué estar familiarizados con el dominio de la casa inteligente ni con las primitivas de este dominio (por ejemplo, dispositivo y *binding providers*). Así, el catalogo de configuraciones disponibles ofrece a los usuarios finales los servicios,

dispositivos y una configuración por defecto para ellos. **El catálogo de configuraciones disponibles está definido en el modelo de decision.**

Por tanto, los **objetivos del modelo de decisión** son dos: (1) Ofrecer a los usuarios finales las configuraciones disponibles de su hogar inteligente y, (2) Guardar la configuración que el usuario final requiere mediante la *end-user front-end*.

Para ofrecer a los usuarios finales una visualización con las configuraciones disponibles de su hogar, estudios descritos en [43] recomiendan utilizar un árbol cuando el número de grupos es elevado. Además, recomiendan que cada opción esté explicada para que los usuarios conozcan las consecuencias. Así, en la representación visual del catálogo de configuraciones disponibles, presentaremos una imagen representativa de cada servicio o dispositivo y una breve descripción.

Además, para **minimizar los cambios en el catálogo de configuraciones**, se deben especificar **familias de disposivos** en lugar de dispositivos porque existe una gran variedad de dispositivos compatibles. Por ejemplo, una familia de dispositivos *Volumetric Detector* tiene un catálogo de dispositivos asociados que pueden ser los siguientes detectores: *Volumetric 360 degree* y *Volumetric 160 degree*. Así, cuando un nuevo dispositivo se soporte en el sistema, sólo es necesario actualizar el catálogo de la familia de dispositivos en lugar de modificar el catalogo de configuraciones disponibles.

Una vez que los usuarios finales han definido la información necesaria para describir su sistema y, crear así, una descripción del mismo, se ha estudiado la población de usuarios finales para identificar distintos perfiles y ofrecer mecanismos que se adapten mejor a ellos.

5.3 Perfiles de Usuarios Finales

La población de usuarios finales es muy diversa porque los usuarios finales están presentes en muchos dominios y tienen necesidades distintas. Por eso, los usuarios finales pueden ser categorizados de diferentes formas. Por ejemplo, los usuarios finales pueden ser categorizados por sus actividades o por el uso que realizan del software.

Hemos estudiado las categorizaciones de usuarios finales descritas en [44] y hemos observado dos **perfiles** comunes:

1. **Usuarios finales:** no están familiarizados con aplicaciones informáticas, con el dominio de la casa inteligente ni con las primitivas del dominio.
2. **Usuarios finales avanzados:** no son expertos en aplicaciones informáticas pero les resultan familiares las aplicaciones informáticas porque ellos las utilizan en casa o trabajan con ordenadores (por ejemplo, doctores, arquitectos, contables, etc.) y pueden estar familiarizados con el dominio de la casa inteligente.

Así, para definir un proceso que capture la información necesaria del sistema, hemos definido tres roles de usuarios de acuerdo a los perfiles presentados anteriormente. Los roles son los siguientes:

- ***End-user:*** representa a los usuarios que no están familiarizados con aplicaciones o sistemas informáticos.
- ***Advanced end-user:*** representa a usuarios finales que están familiarizados con aplicaciones informáticas pero no son expertos en estas aplicaciones informáticas.
- **Analista:** reprenta a un experto profesional en aplicaciones y/o sistemas informáticos. En este caso, se trata de un experto en el modelado de sistemas pervasivos en el método MDDSPL y en el dominio del hogar inteligente. El Analista tiene que dar soporte a los roles de los usuarios finales durante el proceso definido para que describan su sistema y, tienen que validar las descripciones del usuario de la descripción del sistema para que contenga toda la información necesaria para su desarrollo.

5.4 Soportando los distintos Perfiles. Interfaces

Técnicas de visualización han sido frecuentemente utilizadas para guiar a los ingenieros de software durante el desarrollo software. Por ejemplo,

diseño [45], depuración y pruebas [46; 47] y, mantenimiento [48; 49]. Una visualización natural puede constituir un mecanismo valioso para la participación activa de los usuarios finales en las etapas tempranas del proceso de desarrollo. Para permitir esto, utilizamos interfaces de usuario.

Nielsen [50] recomienda que las interfaces deben "hablar el lenguaje del usuario". Esto incluye tener buenos *mappings* entre el modelo conceptual de información del usuario y la interfaz definida para ello. En conclusión, se ha demostrado en [51; 52] que **el uso de una representación visual parece ser la mejor opción porque resulta más intuitivo para los usuarios finales que utilizar otras opciones como pueden ser representaciones textuales**.

Para seleccionar una representación visual adecuada, nos hemos inspirado en técnicas bien aceptadas en el campo de *End-user Development* (ver Sección 2.1) tales como: *Natural Programming* [13], *Programming By Example* [15], *Visual Programming* [53] y la metáfora de las piezas de puzle, con el objetivo de mejorar la usabilidad de los usuarios en las interfaces. La ideas aplicadas en esta aproximación de cada técnica son las siguientes:

- ***Natural Programming:*** ofrecer a los usuarios finales una representación natural de su entorno.
- ***Visual Programming:*** ofrecer a los usuarios finales representaciones visuales de los elementos de su entorno para que les resulte familiar.
- ***Programming by example:*** permitir a los usuarios finales especificar nuevas configuraciones seleccionando la funcionalidad deseada en su entorno físico o elementos que los representen.
- **Metáfora piezas de puzle:** permitir a los usuarios personalizar las configuraciones de los servicios gestionando piezas de puzle.

La representación visual que proponemos para permitir a los usuarios finales para describir su sistema es visualización inmediata natural (*Natural Immediate Visualization*). Natural es un concepto utilizado en *End-user Development* que es definido como [13]:

"representar fielmente la naturaleza o la vida", lo que implica que se trabaja de acuerdo a la forma en que la gente espera. Visualización [54] se define como la formación de una imagen mental que no está necesariamente relacionada con algo en el campo visual [55]. Visualización de la información se refiere a la visualización de una gran cantidad de datos y su representación en un modo comprensivo y natural para usuarios finales. Por tanto, se define la **visualización inmediata como ofrecer a los usuarios finales una visualización natural de las necesidades que acaban de describir por su cuenta**.

Esta visualización inmediata soporta los perfiles de usuarios finales presentados en la sección anterior (Usuarios Finales y Usuarios Finales Avanzados) mediante dos tipos de interfaces: *Closed-option* y *Open-option*.

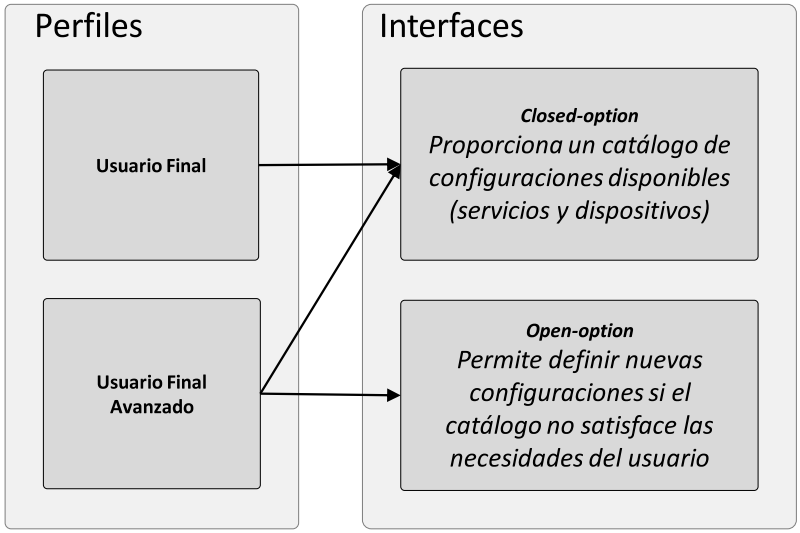


Fig. 5.2: Interfaces de la aproximación

La Fig. 5.2 representa las interfaces plantedas y los perfiles a los que dan soporte. Los Usuarios Finales son soportados por interfaces *Closed-option* mientras que los Usuarios Finales Avanzados están soportados por ambas interfaces *Closed-option* y *Open-option*. Las interfaces *Closed-option* proporcionan a los usuarios finales el catálogo de configuraciones disponibles presentado en la sección anterior. Las interfaces *Open-option* permiten a los Usuarios

Finales Avanzados definir nuevas configuraciones si el catalogo de configuraciones disponibles no satisface sus necesidades.

En conclusión, nuestro objetivo principal es soportar estos perfiles de usuarios e interfaces en nuestra aproximación. Entonces, para permitir a los usuarios finales especficar las características del sistema descritas en la sección anterior (entorno físico, localizaciones, usuarios, políticas, servicios y dispositivos), hemos definido un proceso que guía a los usuarios finales durante la descripción de su sistema indicando las fases y los pasos que tienen que llevar a cabo. Además, este proceso describe cómo interactúan los usuarios finales con el Analista.

5.5 Proceso para Crear la Descripción del Sistema

Para abordar la descripción del sistema por parte de los usuarios finales nos hemos basado en las siguientes ideas:

1. **Los usuarios finales participan en el proceso de diseño** [56].
2. **Los Analistas cooperan con los usuarios finales** [57].
3. **El diseño fomenta la participación** [1].

Para soportar estas ideas y los perfiles e interfaces definidas en secciones anteriores, hemos definido un proceso encargado de crear una descripción del sistema y que está sea utilizada para generar el sistema pervasivo utilizando la aproximación MDDSPL planteada en la Sección 4.3. Además, el proceso define cómo interactúan los usuarios finales con el Analista.

La Fig. 5.3 presenta el proceso propuesto. Los tres roles soportados en el proceso se representan verticalmente mientras que las fases del proceso se representan horizontalmente. Las fases se llaman: (1) *Context Scope*, (2) *System Specification*, (3) *Advanced System* y (4) *Validation*. A continuación se describe cada una de ellas:

1. ***Context Scope:*** el Analista determina el perfil del usuario final (avanzado o no) y el dominio del sistema a ser desarrollado

(por ejemplo, una casa inteligente, un sistema pervasivo para un coche, un edificio o una fábrica). Después, el Analista adapta la representación visual a las características del usuario final y del dominio. En esta tesis de máster, se ha decidido que el dominio del sistema sean las casas inteligentes.

2. ***System Specification:*** los usuarios finales describen las principales características del sistema pervasivo.

3. ***Advanced System:*** las características descritas por los usuarios finales en la fase anterior, son refinidas incluyendo nuevas configuraciones que no están disponibles en el catálogo de configuraciones.

4. ***Validation:*** el Analista valida junto con los usuarios finales las características definidas por los usuarios finales con el obejtivo de obtener la descripción del sistema. Si en esta fase se detectan ambiguedades o errores, los usuarios finales pueden repetir las diferentes fases del proceso de forma iterativa.

Las siguientes subsecciones describen en detalle las fases y pasos que forman el proceso. También, defininen la representación visual que soporta los pasos y roles definidos en el proceso.

5.5.1 *Context Scope*

El objetivo de esta fase es que el Analista determine las características del usuario final. Estas características son el dominio donde el sistema va a ser desarrollado (una casa inteligente) y el perfil del usuario (avanzado o no). El Analista deberá ajustar la información de la interfaz en esta fase mediante entrevistas tradicionales.

A partir de estas características, el Analista ajustará la interfaz para que los usuarios finales especifiquen el sistema. Si el Analista considerará que el perfil del usuario es: Usuario Final Avanzado, se preparará la representación visual para ser utilizada mediante interfaces *Open-option* (estas interfaces permiten a los usuarios definir información personalizada). Por el contrario, si el Analista determina que el perfil

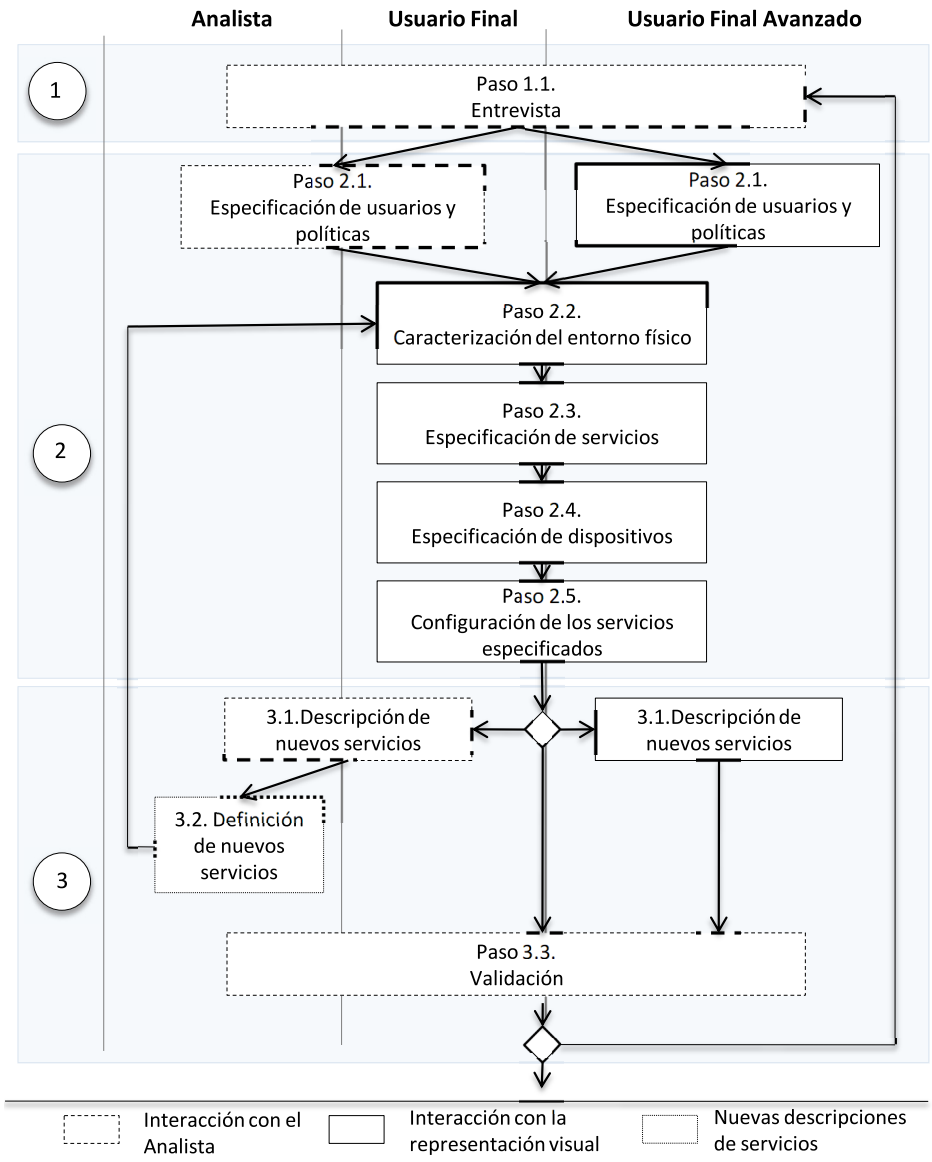


Fig. 5.3: Proceso para obtener la descripción del sistema

del usuario es Usuario Final, se preparará la representación visual para ser utilizada mediante interfaces *Closed-option*.

Por ejemplo, si el sistema de dominio es una casa inteligente, la representación visual se adaptará para presentar sólo conceptos relacionados con casas inteligentes (ej. localizaciones, servicios y

dispositivos).

5.5.2 *System Specification*

El objetivo de esta fase es que los usuarios finales describan las características de su sistema pervasivo. Para lograr esto, los usuarios finales tendrán que describir las principales características en cinco pasos. A continuación, se describe en detalle cada uno de ellos.

Paso 2.1: Especificación de usuarios y políticas

En este paso, los usuarios finales tienen que describir los usuarios que pueden utilizar el sistema, las políticas y, los vínculos entre un usuario y una política. Por ejemplo, dos usuarios pueden ser "Paul" and "Mary", mientras que dos políticas pueden pueden ser "Parents" y "Kids". En los siguientes pasos, los usuarios finales tienen que indicar los servicios que cada usuario de cada política puede activar. Por ejemplo, los usuarios de la política "Kids" no pueden activar el servicio de seguridad.

Este paso se lleva a cabo de diferente forma dependiendo del perfil del usuario final. El Usuario Final tiene que describir los usuarios y las políticas trabajando con el Analista mientras que, el Usuario Final Avanzado puede definir esta información el mismo utilizando una representación visual en una interfaz *Open-option*. Para definir esta representación visual, nos hemos inspirado en las técnicas *End-user Development* descritas anteriormente: *Natural Programming* and *Visual Programming* para representar de forma visual cada usuario definido, cada política y los vínculos entre ellos.

Paso 2.2: Caracterización del entorno físico

Tanto el perfil de Usuario Final como el Usuario Final Avanzado, tienen que caracterizar el entorno físico donde va a desplegarse su entorno. Además, deben caracterizar las localizaciones donde los servicios del sistema van a estar disponibles. Por ejemplo, el usuario final puede especificar que su sistema va a proporcionar servicios en cuatro localizaciones distintas de su hogar: el baño, una habitación , la sala de estar y la cocina.

Paso 2.3: Especificación de Servicios

En este paso, los usuarios finales tienen que especificar los servicios

que necesitan en su entorno físico y dónde se localizan estos (por ejemplo, el servicio de Iluminación Automática en la sala de estar). Tanto los Usuarios Finales como los Usuarios Finales Avanzados tienen que especificar sus servicios mediante interfaces *Closed-option*. Para soportar esto, nos hemos inspirado en las técnicas *End-user Development* descritas anteriormente: *Natural Programming*, *Visual Programming*, y *Programming By Example*.

Para permitir a los usuarios finales especificar servicios, se mantiene la caracterización del entorno físico definida en el paso previo y se ofrece a los usuarios el catálogo de servicios disponibles (el catálogo de servicios predefinidos y configuraciones ha sido descrito en la Sección 5.2).

Cuando los usuarios finales especifiquen un servicio en una localización, se deberá mostrar si un servicio se ha podido resolver o no. Esto se podría mostrar utiliznado los colores: verde y rojo respectivamente. Por último, los servicios especificados en el entorno físico se almacenan en la descripción del sistema.

Paso 2.4: Especificación de Dispositivos

En este paso, los usuarios finales especifican los dispositivos que están disponibles en cada localización. Para hacer esto, la representación visual proporcionada es similar a la descrita en el paso anterior. Por un lado, se presenta el catálogo predefinido para permitir a los usuarios finales requerir dispositivos. Por otro lado, se representa el entorno físico definido en el paso 2.2. Así, los usuarios pueden especificar dispositivos en distintas localizaciones de su entorno.

Como se ha descrito anteriormente, para **minimizar los cambios en el catálogo de configuraciones**, especificamos **familias de disposivos** en lugar de dispositivos porque existe una gran variedad de dispositivos compatibles. Así, cuando un nuevo dispositivo sea soportado en el sistema, sólo es necesario actualizar el catálogo de la familia de dispositivos en lugar de modificar el catálogo de configuraciones disponibles. Si los usuarios finales no conocen que dispositivo a seleccionar de una familia de dispositivos, pueden dejar vacía está configuración. En este caso, el Analista tendrá que completar esta información en la última fase del proceso.

Paso 2.5: Configuración de los servicios especificados

En este paso, los usuarios finales pueden especificar una configuración personalizada para cada servicio especificado en el paso 2.3.

Para soportar esto, tanto el Usuario Final Avanzado como el Usuario Final definen estas configuraciones mediante una interfaz *Closed-option*. La representación visual de esta interfaz ha sido inspirada por la métafora de las piezas de puzle descrita anteriormente. Se ofrece una configuración por defecto para cada servicio en una serie de izquierda a derecha de piezas de puzle, de acuerdo al catálogo de configuraciones. Cada pieza puede tener propiedades definidas para permitir a los usuarios finales personalizar sus servicios. Estas propiedades tienen un valor por defecto definido. Por ejemplo, la Lámpara gradual tiene una propiedad llamada: Intensidad con un valor por defecto: 200.

Por tanto, para personalizar un servicio, los usuarios finales tienen que eliminar una pieza que represente el cambio y unir otra pieza compatible. Opcionalmente, pueden personalizar cada servicio cambiando el valor de las propiedades de cada pieza.

5.5.3 *Advanced System*

La definición de nuevas configuraciones no soportadas en el catálogo de configuraciones, implica definir nuevos servicios y su configuración requerida.

Los Usuarios Finales Avanzados pueden definir esta nueva información ellos mismos mediante interfaces *Open-option*. Los Usuarios Finales necesitan interactuar con el Analista para describir las nuevas configuraciones (nuevos servicios y configuraciones) que necesiten.

Una vez que las nuevas configuraciones han sido definidas, el Analista las incorpora en el catalogo predefinido de configuraciones. Entonces, los Usuarios Finales pueden incluirlas en la descripción de su sistema mediante interfaces *Closed-option*. La representación visual para especificar nuevas configuraciones y el paso que incluye esta fase, se presenta en detalle a continuación.

Paso 3.1: Descripción de nuevos servicios

En este paso, para reducir la complejidad para los Usuarios Finales, sólo los Usuarios Finales Avanzados describen la información necesaria

para un servicio nuevo. Esta información es: el nombre del servicio, su localización, qué dispositivos o servicios son necesarios para percibir información de contexto (*condition*) y, qué dispositivos o servicios son necesarios para activar este servicio (*action*). Por ejemplo, el usuario final puede necesitar un servicio de climatización que enciende el aire acondicionado cuando la ventana de la sala de estar está cerrada y la temperatura de la sala de estar es superior a 26ºC.

Para soportar esto, la representación visual de este paso ha sido inspirada por las siguientes técnicas *End-user Development* descritas anteriormente: *Natural Programming, Visual Programming,* y *Programming By Example.*

La interfaz *Open-option* proporcionada en este paso para los Usuarios Finales Avanzados ofrece la misma representación física del entorno especificada en el paso 2.2. Esta representación visual proporciona mecanismos que permiten al Usuario Final Avanzado describir la información necesaria de un nuevo servicio seleccionando elementos de su entorno físico. También, ofrece un mensaje de texto donde los Usuarios Finales Avanzados pueden leer y modificar el resultado de su descripción. En este mensaje de texto, la representación visual permite modificar alguna información. Por ejemplo, el valor de las acciones o condiciones (activo, inactivo o un valor numérico) y cómo el mensaje de texto tiene que unir dos o más acciones o condiciones (*and/or*). Esta información modificable se visualiza de diferente forma que la información que no es modificable.

Una vez que se ha especificado la información requirida para un nuevo servicio, esta información se almacena en la descripción del sistema.

5.5.4 *Validation*

El objetivo del único paso de esta fase es que el Analista junto con los usuarios finales validen la descripción del sistema mediante una entrevista tradicional.

Si en esta fase se detectan ambiguedades o errores, los usuarios finales pueden repetir las fases del proceso de forma iterativa. En

caso contrario, la descripción obtenida en el modelo de decisión y las descripciones del sistema, se utilizan como entrada para el modelo de realización y para detalles manuales del Analista, tal y como se ha descrito en la Sección 4.3.

5.6 Conclusiones

Este capítulo describe en detalle la aproximación propuesta en esta tesis de máster para introducir a los usuarios finales en el método MDDSPL en las fases tempranas del proceso de desarrollo.

Para ello, se ha identificado la información que deben proporcionar de su sistema los usuarios finales. Seguidamente, se ha estudiado la población de usuario finales y se han identificado dos perfiles comunes: el Usuario Final y el Usuario Final Avanzado. A partir de estos perfiles, se han identificado tres roles que deben ser soportados en la aproximación (*End-user, Advanced End-user* y Analista). Después, se han identificado dos tipos de interfaces para soportar los roles defnidos previamente (*Open-option* y *Closed-option*) con el objetivo de mejorar la usabilidad de los usuarios. Por último, se ha definido un proceso que describe las fases y pasos que deben seguir los usurios finales para especificar la infomación requerida y, cómo deben interactúar con los Analistas.

Una vez que se han realizado todas las fases definidas en el proceso, el Analista aporta detalles manuales a las descripciones del sistema en los modelos gráficos de la vista *Pervasive System Analyst* tal y como se ha descrito en el esquema *Software Factory* presentado en la Sección 4.3.

Capítulo 6

Herramienta

Para aplicar la aproximación descrita en el capítulo anterior, hemos implementado una herramienta prototipo. Esta herramienta permite a los usuarios finales describir los requisitos de su hogar inteligente. Para desarrollarla, nos hemos inspirado en conocidas técnicas *End-user Development* y patrones de interacción que mejoran la usabilidad de las interfaces.

El resto de este capítulo está estructurado en las siguientes secciones: la Sección 6.1 presenta una visión general de la herramienta. La Sección 6.2 describe la arquitectura de componentes. La Sección 6.3 describe cómo se aplica la aproximación descrita en el capítulo anterior. La Sección 6.4 describe el catálogo con las configuraciones disponibles del sistema y cómo se crea la descripción del sistema a partir de las configuraciones que especifica el usuario final. La Sección 6.5 presenta las técnologias utilizadas para el desarrollo de la herramienta.

6.1 Visión General

Esta sección ilustra una vista general de la herramienta que hemos desarrollado. La Fig. 6.1 instancia la vista general presentada en el capítulo anterior en los modelos utilizados en la herramienta end-user.

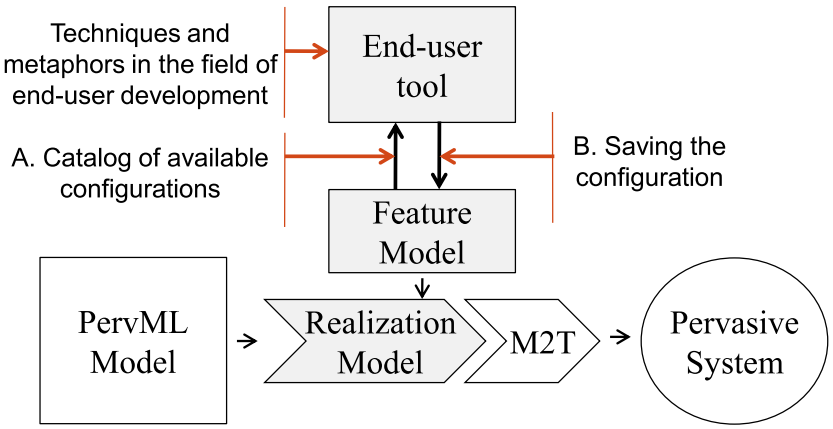


Fig. 6.1: Vista general herramienta

Las siguientes subsecciones describen en detalle cómo estos modelos participan en la herramienta end-user permitiendo que los usuarios finales participen en la descripción de su sistema. Después, el resto de secciones se centran en la herramienta end-user: su arquitectura de componentes, cómo soporta la herramienta las interfaces y el proceso que han sido presentados en el capítulo anterior.

6.1.1 El Modelo PervML

Como se ha descrito en la Sección 3.2, el lenguaje PervML es un Lenguaje Específico de Dominio (DSL) para describir sistemas pervasivos utilizando conceptos con un alto nivel de abstracción.

Este modelo (ver parte inferior de la Fig. 6.2) describe los bloques de construcción para el ensamblado del sistema pervasivo [24]. Los bloques grises implementan la funcionalidad de las características seleccionadas y los bloques blancos activan la funcionalidad alternativa del sistema. Los bloques (l), (m), (o) y (p) proporcionan adaptadores para nuevos

recursos disponibles.

6.1.2 El Modelo de Características

Los modelos de características se utilizan ampliamente para describir un conjunto de productos en una Línea de Productos Sofware en terminos de características. En estos modelos, las características están jerárquicamente unidas en una estructura de árbol y, opcionalmente, conectadas por restricciones. Existen muchas propuestas para tipos de relaciones o representaciones gráficas de modelos de características [58]. Para el desarrollo de esta herramienta hemos elegido el **Modelo de Características** [59] como lenguaje de modelado porque está orientado al análisis y tiene un buen soporte de herramienta [60].

Este modelo (ver parte superior de la Fig.6.2) determina las características iniciales y potenciales en la casa inteligente es decir, el catálogo de configuraciones disponibles. Las características representadas de color gris en la figura, son características seleccionadas mientras que las características de color blanco representan características potenciales. Inicialmente, la casa inteligente proporciona *Automated illumination, Presence simulation* y *Security*. *Security* depende de *In home detection* y una sirena de alarma (*Siren*). El sistema puede ser potencialmente actualizado con sensores volumétricos de presencia (*Volumetric Detection*) y más alarmas para mejorar la seguridad del hogar. En la sección 6.4 se describe este modelo en detalle.

6.1.3 El Modelo de Realización

El modelo de realización es una extensión incorporada a Atlas Model Weaving (AMW) [61] del método MDDSPL para relacionar las características del modelo de características con elementos de PervML.

AMW es un modelo para establecer relaciones entre modelos. La extensión aumenta la relación AMW (*AMW relationship*) con etiquetas *default* y *alternative*. Esta relación se aplica entre características y elementos de PervML (*Binding Providers* (BPs) y Servicios). En el contexto de un BP, la relación *default* significa que un BP ha sido seleccionado para la configuración inicial del sistema. La relación

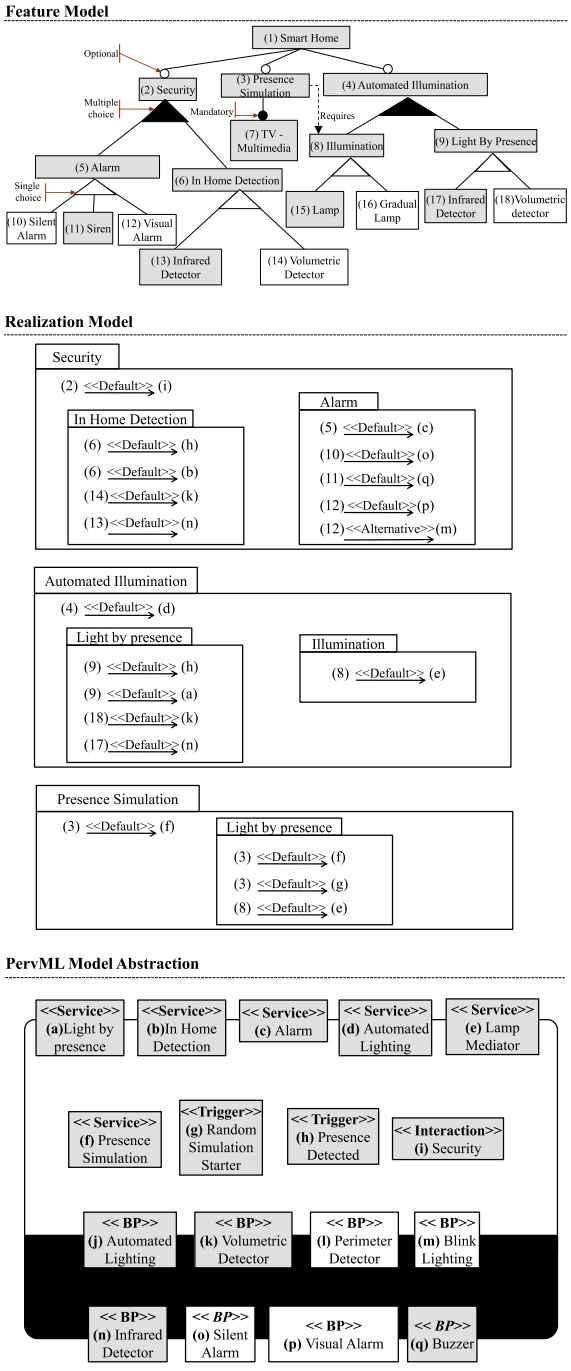


Fig. 6.2: Modelos herramienta

alternative significa que un BP se considera y no participa en la configuración inicial pero puede ser incorporado a la Línea de Producto Software. Los BPs inactivos proporcionan BP alternativos para reemplazar el BP por defecto en caso de fallo.

Este modelo (ver parte central de la Fig. 6.2) establece las relaciones entre las características y los elementos de PervML. Por ejemplo, la característica Alama Visual está relacionada con un BP (p) para alarmas visuales pero, alternativamente, puede ser reemplazado con un BP (m) que emula una alarma visual utilizando iluminación intermitente.

6.1.4 La Herramienta End-user

La herramienta end-user permite a los usuarios finales describir su sistema y configurar el modelo de características siguiendo la aproximación descrita en el capítulo anterior. Así, cuando los usuarios finales han finalizado de describir las características de su sistema, se obtiene el modelo de decisión y una descripción del sistema.

6.2 Arquitectura de componentes

Como se ha descrito en la sección anterior, en nuestra herramienta hemos representado el catálogo de configuraciones describe utilizando la técnica *Feature Modelling* [35].

El modelo de características de la Fig. 6.3, presenta un pequeño ejemplo de catálogo de configuraciones que soporta los servicios y dispositivos disponibles en una casa inteligente modelados como un modelo de características.

Además, la Fig. 6.3 presenta la arquitectura de componentes de nuestra herramienta. La arquitectura esta compuesta de:

- Una **interfaz** dónde los usuarios finales especifican sus requisitos mediante las interfaces y proceso definidos en el capítulo anterior.
- Un **controler** *(Controller)* utilizado para asistir a los usuarios finales en la descripción de sus necesidades proporcionándoles

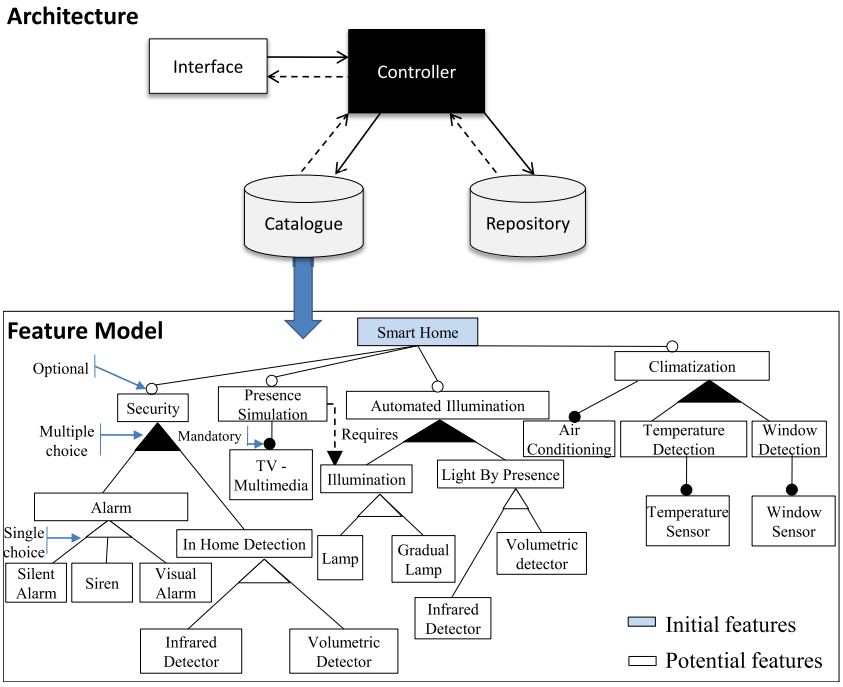


Fig. 6.3: Arquitectura de componentes

la interfaz adecuada o comprobando que sus descripciones son completas y sin errores. El *Controller* trabaja en cada paso del proceso. Un ejemplo de tarea llevada a cabo por el *Controller* es asegurar que las especificaciones de los servicios son completas (ej. un servicio seleccionado tiene todos los dispositivos requeridos configurados en la localización adecuada). El *Controller* se ha implementado utilizando *Eclipse Modelling Framework* (EMF) para recuperar información del modelo de características y, FAMA Framework [62] para llevar a cabo evaluaciones y comprobar errores.

- Un **Modelo de características** que representa el **catálogo** predefinido de configuraciones disponibles de la casa inteligente. El modelo de características determina las características iniciales y potenciales del hogar inteligente. Estas características representan los servicios y las familias de dispositivos. Las familias de dispositivos son las hojas del modelo de características mientras

que los servicios son los nodos que no son hojas. El modelo de características se representa en nuestra herramienta mediante *XML Metadata Interchange* (XMI).

- Un **repositorio** donde se almacenan las descripciones del sistema que realizan los usuarios finales. Estas descripciones son: usuarios, políticas, entorno físico, localizaciones y nuevas configuraciones. Para el repositorio también se ha utilizado XMI.

6.3 Soportando la Aproximación en la interfaz End-user

Para soportar la aproximación, esta herramienta tiene que soportar los roles, la representación visual y las interfaces descritas en el capítulo anterior. Además, la herramienta debe sorportar el proceso definido para permitir a los usuarios finales describir toda la información requerida sobre su hogar inteligente. Las próximas subsecciones describen cómo la herramienta end-user desarrollada soporta la aproximación.

6.3.1 Soportando las Descripciones

Como se ha descrito en el capítulo anterior, para diseñar una interfaz end-user nos hemos basado en técnicas aceptadas en el campo End-user Development (*Natural Programming, Programming By Example, Visual Programming* y metáforas). Además, hemos aplicado patrones de interacción y principios de diseño para ayudar a los usuarios finales ya que estos pueden no estar familiarizados con aplicaciones o sistemas informáticos. De acuerdo a los estudios [63; 64; 65], las principales decisiones que hemos aplicado para diseñar las interfaces son las siguientes:

- **Utilizar un *wizard*:** en nuestro proceso para lograr un objetivo único (la descripción de sistema) necesitan ser abordadas muchas decisiones antes de obtener el objetivo complemente (muchos pasos), que pueden no ser conocidos por los usuarios finales.

Utilizar un *wizard* se recomienda en [63] para los usuarios que desean obtener un objetivo pero no están interesados en los pasos que hay que llevar a cabo para obtenerlo.

- **Ofrecer botones de navegación:** utilizamos botones de navegación para sugerir a los usuarios finales que están siguiendo un camino con pasos. Esto es recomendado en [63] porque se mejora el aprendizaje y la memorización de la tarea de cada paso. Además, cuando se fuerza a los usuarios a seguir unos pasos en orden, son menos propensos a olvidar cosas importantes y a cometer menos errores.

- **Representar elementos en un *grid layout***: se recomienda en [63] que cualquier circunstancia donde se presentan muchos objetos de información sean presentados y organizados espacialmente en un área limitada. Esto mejora la presentación y minimiza el tiempo lectura y búsqueda de los objetos en pantalla.

- **Ofrecer opciones:** una conclusión interesante obtenida en [64] es la siguiente "*what people see is what they select from!*". El estudio afirma que las personas tienden a seleccionar de la lista completa de opciones que se presentó por primera vez. Rara vez realizan un esfuerzo para encontrar opciones adicionales a través de desplazamiento. Si se presentan once elementos, la elección es de estos once. Cuando las opciones deben ser comparadas entre sí, los controles de presentación con todas las opciones, obtienen los mejores resultados.

- **Selección en lugar de introducir texto:** los estudios presentados en [65] presentan las ventajas y las desventajas de utilizar campos de entrada o campos de selección para la recolección de datos. Si se trata de información que es menos conocida o sujeta a errores ortográficos o a errores, recomiendan elegir una técnica de selección.

Por tanto, hemos desarrollado una interfaz basada en las decisiones presentadas anteriormente que permiten especificar los servicios y dispositivos que los usuarios finales necesitan. La Fig. 6.4 presenta

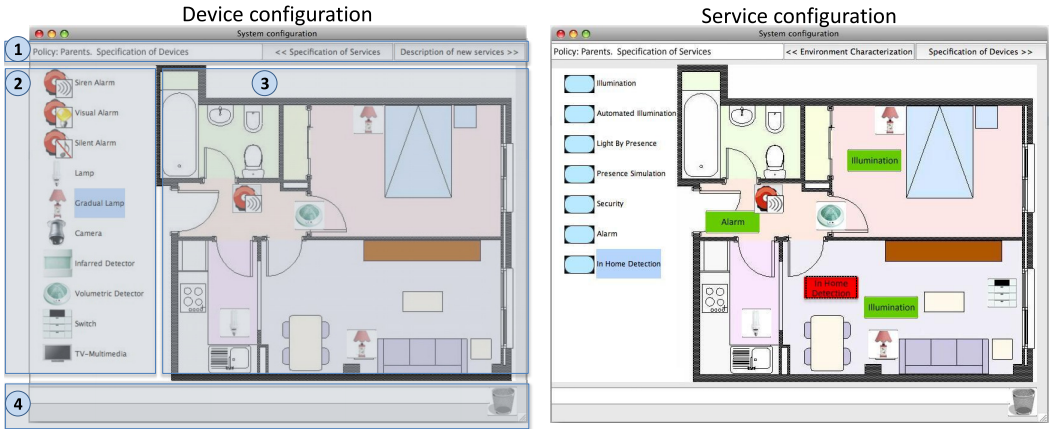


Fig. 6.4: Herramienta End-user. Captura de pantalla

una captura de pantalla cuando los usuarios finales están configurando servicios y dispositivos en el entorno físico de su hogar inteligente. Cada interfaz está divida en 4 areas:

1. Título y botones de navegación.
2. Catálogo de configuraciones disponibles.
3. Representación del entorno físico.
4. Información dónde la herramienta puede advertir a los usuarios finales o asistirlos.

En concreto, la parte izquierda de la figura presenta cómo el usuario final ha seleccionado algunos dispositivos para diferentes localizaciones de su casa inteligente (por ejemplo, una sirena y un detector volumétrico para el pasillo). La parte derecha de la figura, presenta los servicios que han seleccionado los usuarios finales para diferentes localizaciones (por ejemplo, el servicio de alarma para el pasillo).

Como puede observarse en la Fig.6.4 muestra, hemos aplicado los patrones de interacción descritos anteriormente. Se ha aplicado el patrón *grid layout* para dividir la interfaz en 4 áreas. El patrón *wizard* se utiliza para guiar a los usuarios a través del proceso

definido preguntando progresivamente a los usuarios la información requerida (servicios, dispositivos, etc.). Además, se han utilizado los botones de navegación en el área (1) para permitir a los usuarios finales navegar entre las distintas interfaces que solicitan información. Los patrones de ofrecer opciones y selección en lugar de introducir texto, son aplicados en el área (2) ofreciendo los dispositivos/servicios disponibles como opciones y permitiendo a los usuarios arrastrar estos dispositivos/servicios en el entorno físico del usuario representado en el área (3).

6.3.2 Soportando el Proceso

Esta subsección describe las interfaces que soportan los pasos definidos en las fases 2 y 3 del proceso, dónde los usuarios finales interactúan con la representación visual.

Paso 2.1: Especificación de políticas y usuarios

La representación visual de este paso consiste en dos áreas que los usuarios pueden gestionar: *User* y *Policy*. Cuando los Usuarios Finales Avanzados definen un usuario en el sistema, se requiere una imagen y un nombre. Cuando se define una política sólo se requiere el nombre. Los nombres de los usuarios y los nombres de las políticas deben ser únicos y almenos, los Usuarios Finales Avanzados tienen que especificar un usuario y una política en su sistema.

Además, el Usuario Final Avanzado puede vincular los usuarios definidos con una o más políticas. Para hacer esto, el usuario arrastra la imagen que aperece junto al nombre del usuario junto al nombre de la política deseada. La Fig. 6.5 presenta una especificación de usuarios y políticas utlizando nuestra herramienta. En primer lugar, el Usuario Final Avanzado crea los usuarios *Paul* y *Mary* especificando sus nombres y seleccionando una imagen representativa para ellos (ver parte izquierda de la figura). En segundo lugar, los Usuarios Finales Avanzados crean las políticas *Parents* y *Kids* (ver parte derecha de la figura). Por último, cuando un Usuario Final Avanzado vincula un usuario con una política, la interfaz presenta la imagen representativa del usuario cerca del nombre de la política.

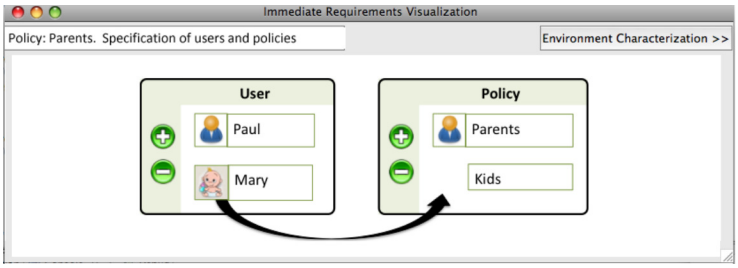


Fig. 6.5: Especificación de usuarios y políticas

Paso 2.2: Caracterización del entorno físico

En la herramienta permitimos que los usuarios finales puedan definir su entorno físico y las distintas localizaciones utilizando una imagen representativa. Esta imagen puede ser un plano o incluso una imagen. Después, los usuarios finales pueden describir las localizaciones de su entorno físico sobre la imagen (por ejemplo, habitaciones, cocina, sala de estar, etc.).

Se permite que los usuarios utilicen un plano o incluso una imagen representativa de su casa porque esta representación está basada en el conocimiento y el entorno familiar del usuario final. La Fig. 6.4 presenta una captura de pantalla de la herramienta dónde las diferentes localizaciones de la casa se han identificado sobre el plano (un pasillo, un baño, una habitación, una cocina y una sala de estar).

Paso 2.3: Especificación de servicios

La herramienta proporciona una interfaz que está divida en dos frames: el frame de la izquierda ofrece una lista de servicios disponibles y el frame de la derecha ofrece una visualización del entorno físico. Para definir un servicio en una localización, los usuarios finales tienen que seleccionar un servicio del frame izquierdo y arrastrarlo en la localización adecuada.

Inmediatamente, una imagen representativa será mostrada en la localización adecuada del entorno físico en dos colores:

- **Rojo**, si la configuración del servicio no resuelve sus dependencias de acuerdo al catálogo predefinido (servicios/dispositivos que el servicio necesita no han sido definidas aun).

- **Verde**, si la configuración del servicio ha resuelto las dependencias de acuerdo al catálogo predefinido.

Después, el servicio requerido será seleccionado como configuración inicial en el modelo de caraterísticas.

La Fig. 6.6 presenta la especificación del servicio de iluminación automática (*Automated Illumination*) en la sala de estar. Cuando el usuario final arrastra este servicio, las familias de dispositivos requeridas son añadidas automáticamente en la sala de estar (en este caso, el sensor Volumétrico *Volumetric Detector* y una lámpara *Lamp*).

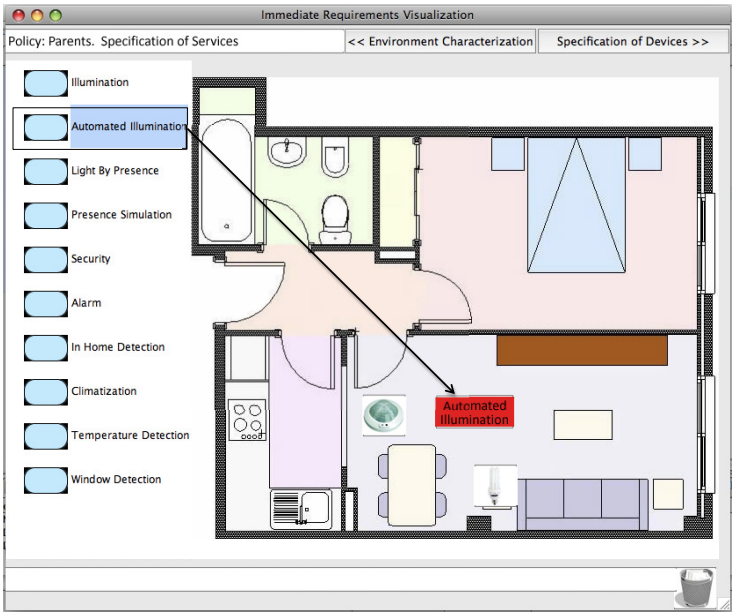


Fig. 6.6: Especificación de servicios

Paso 2.4: Especificación de dispositivos

En este paso los usuarios finales especifican los dispositivos disponibles en cada localización. Para hacer esto, la herramienta proporciona una interfaz similar a la descrita en el paso anterior. La interfaz está divida en dos frames: el frame izquierdo presenta la lista de familias de dispositivos que están disponibles en el catálogo de acuerdo al modelo de características mientras que, el frame izquierdo representa el entorno físico (ver Fig. 6.7). Además, para especificar un dispositivo

en una localización, los usuarios finales sólo necesitan seleccionarlo en el frame izquierdo y arrastarlo a la localización deseada.

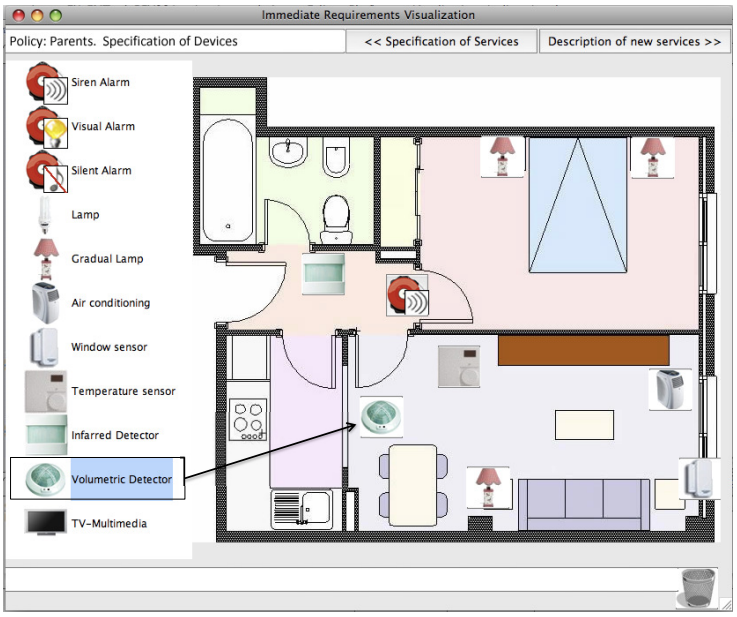


Fig. 6.7: Especificación de dispositivos

Por cada familia de dispositivos representada en el entorno físico, los usuarios finales deben seleccionar el dispositivo específico. Para ello, los usuarios finales tienen que seleccionar la familia de dispositivos en el entorno físico (frame derecho) y seleccionar una opción disponible del catálogo para esa familia (frame izquierdo). Por ejemplo, la Fig. 6.8 presenta los contenidos del frame izquierdo cuando un usuario ha seleccionado la familia de dispositivos *Volumetric Detection*. En este caso los contenidos a seleccionar son la marca y el modelo.

Paso 2.5: Configuración de los Servicios Especificados

En este paso, los usuarios finales son capaces de definir la configuración de cada servicio especificado en el paso 2.3.

La interfaz de la herramienta ofrece una configuración por defecto de cada servicio en una serie de izquierda a derecha de piezas de puzle de acuerdo al catálogo de configuraciones. Cada pieza representa una característica del modelo de características y, cada característica puede tener definidas propiedades para permitir a los usuarios finales

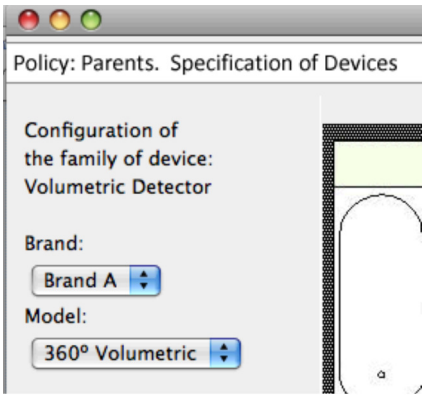


Fig. 6.8: Especificación de características de un dispositivo

personalizar sus servicios. Estas propiedades tienen un valor por defecto definido. Por ejemplo, la lámpara gradual (*Gradual Lamp*) tiene una propiedad llamada: *Intensity* con un valor por defecto: 200.

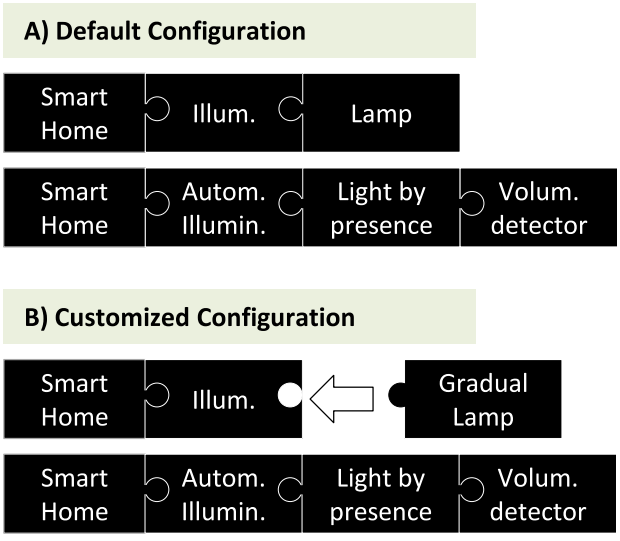


Fig. 6.9: Configuración del servicio de Iluminación Automática

Por tanto, para personalizar un servicio, los usuarios finales tienen que eliminar la pieza que representa el cambio y unirla con otra pieza compatible. Opcionalmente, pueden personalizar cada servicio modificando los valores de las propiedades de cada pieza. Por ejemplo,

la Fig. 6.9(A) presenta la configuración por defecto del servicios *Automated Illumination* definido en la sala de estar. La interfaz ofrece dos series de piezas porque este servicio requiere dos dispositivos: un detector volumétrico y una lámpara. Si los usuarios finales necesitan personalizar la configuración de este servicio para utilizar una lámpara gradual en lugar de una lámpara, simplemente tienen que eliminar la pieza que representa la lámpara (*Lamp*) y unir la pieza que representa la lámpara gradual (*Gradual Lamp*) (ver Fig. 6.9(B)). También, el usuario final puede modificar el valor de la intensidad de la lámpara gradual (por ejemplo a 600).

Paso 3.1: Descripción de nuevos servicios

En este paso, se captura la información relacionada con la descripción de nuevos servicios. Como se ha presentado en el capítulo anterior, para reducir la complejidad de los usuarios finales, sólo el Usuario Final Avanzado puede llevar a cabo este paso utilizando una interfaz *Open-option*.

La interfaz *Open-option* proporcionada en este paso para los Usuarios Finales Avanzados, ofrece la misma representación del entorno físico introducida en los pasos anteriores. Esta representación visual es de nuevo complementada con un frame izquierdo que proporciona a los Usuarios Finales Avanzados mecanismos para definir la información requerida para un nuevo servicio. La Fig. 6.10 presenta una descripción para un nuevo servicio de climatización para la sala de estar utilizando la interfaz *Open-option* definida.

El frame izquierdo de la figura presenta la información requerida: nombre, localización, condicion y acción. Para introducir esta información, los usuarios sólo deben seleccionar opción correspondiente en el frame izquierdo y entonces, seleccionar el elemento del entorno físico en el frame derecho. Por ejemplo, para especificar la localización el usuario selecciona localización en el frame izquierdo y después selecciona la localización del entorno en el frame derecho.

La representación visual muestra qué información ha sido seleccionada en el entorno y también, ofrece un mensaje de texto donde los usuarios pueden leer modificar el resultado de su descripción (frame inferior). En el mensaje de texto se permite a los usuarios modificar

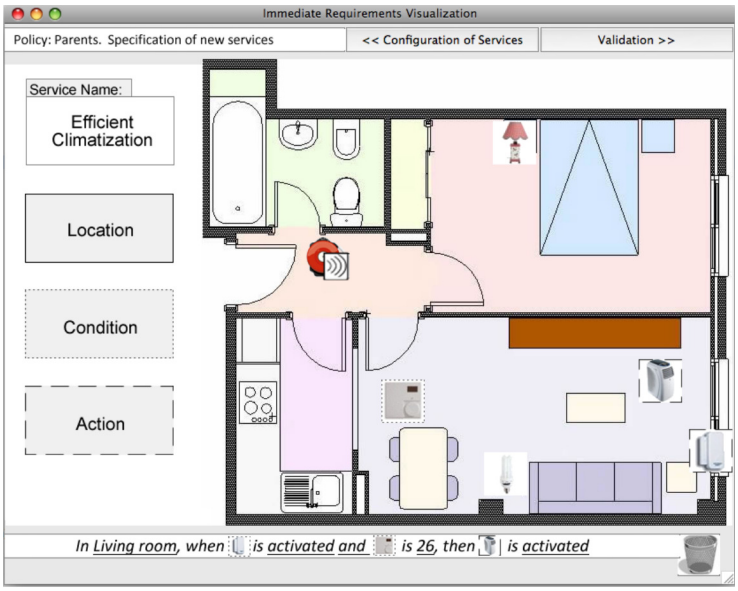


Fig. 6.10: Descripción de un nuevo servicio

alguna información. Por ejemplo, el valor de la acción o condiciones (activo, inactivo o un valor numérico) y cómo el mensaje de texto tiene que unir dos o más acciones o condiciones (*and/or*). Esta información modificable se visualiza subrayada en el mensaje.

6.4 Catálogo de Configuraciones Disponibles

Como se ha presentado anteriormente, el catalogo de configuraciones disponibles ha sido representado en la herramienta utilizando un modelo de características. Las siguientes subsecciones, describen en detalle este modelo, cómo esta información se presenta en la interfaz de los usuarios finales y, cómo se guardan las configuraciones especificadas por los usuarios.

6.4.1 Modelo de Características para Hogares Inteligentes

En el dominio de la casa inteligente, las configuración del sistema que tienen que seleccionar los usuarios finales está formada por los servicios

y los dispositivos requeridos en cada localización del entorno físico.

El modelo de características también determina cómo las características se relacionan entre si. Como el modelo de características presentado en la Fig.6.2 estas relaciones son las siguientes:

- ***Optional***. Una característica es *Optional* cuando puede estar seleccionada o no si su característica padre está seleccionada. Gráficamente, se representa con un pequeño círculo blanco en la parte superior de la característica.

- ***Mandatory***. Una característica es *Mandatory* cuando tiene que estar seleccionada cuando su característica padre está seleccionada. Gráficamente se representa con un pequeño círculo de color negro en la parte superior de la característica.

- ***Or-relationship***. Un conjunto de características pueden tener una relación *or* con su característica padre cuando una o más de sus características hijas pueden estar seleccionadas simultáneamente. Gráficamente se representa con un triángulo negro.

- ***Alternative***. Un conjunto de características pueden tener una relación *Alternative* con su característica padre cuando sólo una de sus características hija puede estar seleccionada simultáneamente. Gráficamente se representa con un triángulo blanco.

Adicionalmente, la técnica *Feature Modelling* incorpora dos relaciones para expresar restricciones:

- ***Requires***. Una relación *A Requires B* significa que cuando A está seleccionada en la configuración del sistema, B también está seleccionada. Gráficamente está representada con una flecha discontinua.

- ***Excludes***. Una relación *A Excludes B* significa que cuando A está seleccionda en la configuración del sistema, B no puede estar seleccionada. Gráficamante está representada con una flecha doble discontinua.

También proponemos incluir en el modelo de características ***Variation Points*** (puntos de variación). Por puntos de variación nos referimos a una característica que no ha sido seleccionada para la configuración actual pero, puede ser utilizada para definir configuraciones adicionales. Las características de color gris son las características seleccionadas en la configuración de la casa inteligente, mientras que las características de color blanco representan los puntos de variación.

La parte superior de la Fig. 6.2 presenta un modelo de caraterísticas que determina las características iniciales y potenciales de la casa inteligente. Estas características representan servicios o familias de dispositivos. Como se ha descrito en secciones anteriores, la herramienta presenta a los usuarios finales las opciones de acuerdo al modelo de características. La Fig. 6.11 presenta un ejemplo de opciones de servicios y dispositivos de acuerdo al modelo de características. Es importante tener en cuenta que las opciones de familias de dispositivos coinciden con los nodos hojas del modelo de características presentado en la figura (*Siren Alarm, Visual Alarm, Siren Alarm, Infrared Detector* y *Volumetric Detector*) y, las opciones de servicios coinciden con los nodos que no son hojas del modelo de características *(Security, Alarm* y *In Home Detection*).

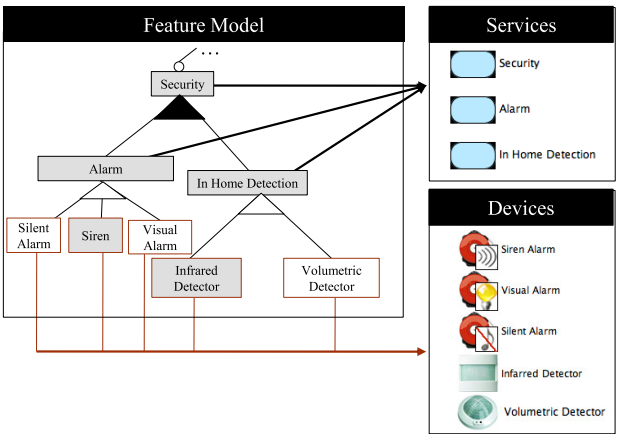


Fig. 6.11: El catálogo de configuraciones disponibles

Las opciones disponibles son mostradas en la interfaz con un árbol,

una imagen representativa de cada opción y una breve descripción de cada una de ellas para permitir a los usuarios conocer las consecuencias de sus selecciones tal y como recomiendan en [63]. La Fig. 6.4 presenta la lista de servicios y dispositivos disponibles para los usuarios finales según el modelo de características presentado en la parte superior de la Fig. 6.2.

6.4.2 Guardando los Requisitos del Usuario Final

Una vez que el catálogo de configuraciones ha sido mostrado, los usuarios finales pueden seleccionar los servicios o dispositivos para cada localización en la herramienta end-user. Cuando esto ocurre, la herramienta end-user establece a activo/inactivo características del modelo de características. Así, cuando los usuarios finales terminen de especificar su sistema, el modelo de características tendrá activadas las características de acuerdo a la configuración especificada por el usuario final.

Como se ha descrito en secciones anteriores, para que los usuarios finales establezcan una configuración, tienen que seleccionar los servicios/dispositivos y arrastrarlos a la localización deseada. Después, una imagen representativa del servicio/dispositivo se presenta en la representación del entorno físico. El servicio/dispositivo puede ser mostrado en dos colores diferentes: (1) **rojo** con una línea discontinua si la configuración del servicio/dispositivo no cumple sus restricciones (servicios/dispositivos que son necesarios) o, (2) **verde** si la configuración del servicio cumple las restricciones. La Fig. 6.4 presenta la especificación de dispositivos (ver la parte izquierda de la figura) y los servicios (ver la parte derecha de la figura).

Para diseñar estas interfaces nos hemos basado en los siguientes principios end-user y patrones de interacción [63; 65]:

- Utilizar **autocompletar**: el estudio presentado en [65] expone que la entrada asistida, también conocida como *autocompletion*, se prefiere frente a métodos de entrada no asistidos. Además, también minimiza el esfuerzo del usuario, reduciendo el tiempo de entrada y las pulsaciones de teclado.

- Utilizar una **advertencia** (*warning*): es recomendado en situaciones donde los usuarios llevan a cabo acciones que pueden conducir sin intención a un problema [63] y el sistema no puede o no debe resolver automáticamente esta situación sin que se consulte el usuario. La advertencia también podría incluir una descripción más detallada de la situación para ayudar al usuario a tomar la decisión apropiada, mediante almenos dos opciones.

- **Ofrecer todas las opciones:** esto es recomendado cuando el número de opciones no es largo y pueden ser presentadas sin **scroll** [65] ya que rara vez se realizan esfuerzos para encontrar opciones adicionales.

- **Ofrecer algunas opciones:** es recomendado [65] cuando el número de opciones es amplio y se necesita *scroll* para presentarlas porque mejora la velocidad, el rendimiento y la satisfacción de los usuarios.

De acuerdo a los patrones de interacción presentados anteriormente, hemos definido un conjunto de *mapings* entre la herramienta y los patrones de interacción dependiendo de la información disponible en el modelo de características. A continuación, presentamos los patrones de interacción utilizados por cada relación del modelo de características:

- Si existe una característica ***Mandatory***, nosotros aplicamos el patrón de interacción **autocompletar**. Cuando una característica A está relacionada con otra característica B con una relación *Mandatory*, si A está seleccionada B también tiene que estar. En la herramienta, las características están representadas por servicios/dispositivos. Así, cuando el usuario final selecciona un servicio que representa la característica A, con una relación *Mandatory* con un elemento B, el servicio que representa característica B se añade automáticamente a la misma ubicación de servicio A. Por ejemplo, cuando un usuario final selecciona el servicio de simulación de presencia (*Presence simulation*) para la sala de estar, ocurre lo siguiente: (1) el dispositivo *TV-Multimedia* se añade automáticamente a la misma localización (2)

como la relación es *Mandatory* entre las características *Presence Simulation* y *TV-Multimedia*, (3) el modelo de características se actualiza activando ambas características.

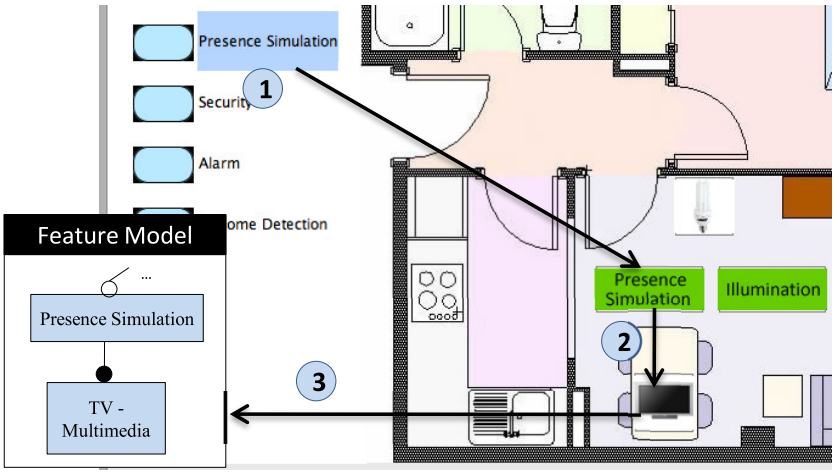


Fig. 6.12: Aplicando patrones en una relación *Madatory*

- Si existe una relación ***Requires*** o ***Excludes***, utilizamos el patrón de interacción **advertencia** (*Warning*). Cuando una característica A tiene una relación requiere con B, si se selecciona A la característica B tiene que estar seleccionada también. De forma similar ocurre cuando A tiene una relación excluyes con B. Cuando A está seleccionada B no tiene que estarlo. Cuando en la herramienta se selecciona un servicio/dispositivo que representa una característica con una relación requiere o excluye, la herramienta advierte a los usuarios finales mostrando una advertencia. La Fig. 6.13 presenta cuando un usuario final selecciona el servicio de simulación de presencia para la sala de estar (1). Como la característica que represta este servicio tiene una relación requiere con la característica Illumination, la herramienta muestra una advertencia (2). Entonces, el usuario final añade el servicio requerido a la misma localización (3) y, el modelo de características se actualiza activando las características *Illumination* y *Presence Simulation* (4).

- Si existe una característica con una relación ***Optional*** o ***Single***

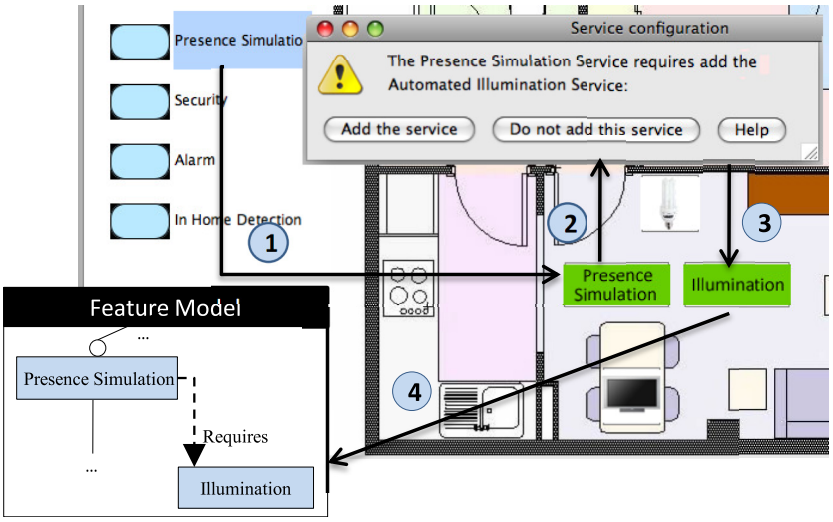


Fig. 6.13: Aplicando patrones en una relación *Requires*

choice, aplicamos los patrones de interacción: **ofrecer todas las opciones y autocompletar**. Cuando una característica A tiene una de estas relaciones con otras características, una de ellas tiene que ser seleccionada. En la herramienta, cuando un usuario final selecciona un servicio que representa una característica con una de estas relaciones, la herramienta presenta un dialogo con todos los servicios/dispositivos que representan las características relacionadas. Así, cuando el usuario final selecciona una de ellas, la herramienta añade la característica seleccionada a la misma localización. Finalmente, el modelo de características es actualizado. La Fig.6.14 presenta un ejemplo representativo de cómo el usuario selecciona el servicio de Alarma (1). Este servicio representa una característica que tiene una relación *Single Choice*. Entonces, la herramienta presenta un dialogo con todos los dispositivos que representan las características relacionadas (2). Entonce el usuario final selecciona el dispositivo de sirena (*Siren*) y el modelo de características se actualiza activando las características: *Alarm* y *Siren* (3).

- Si existe una relación **Multiple choice**, utilizamos el patrón de interacción **ofrecer algunas opciones y autocompletar**.

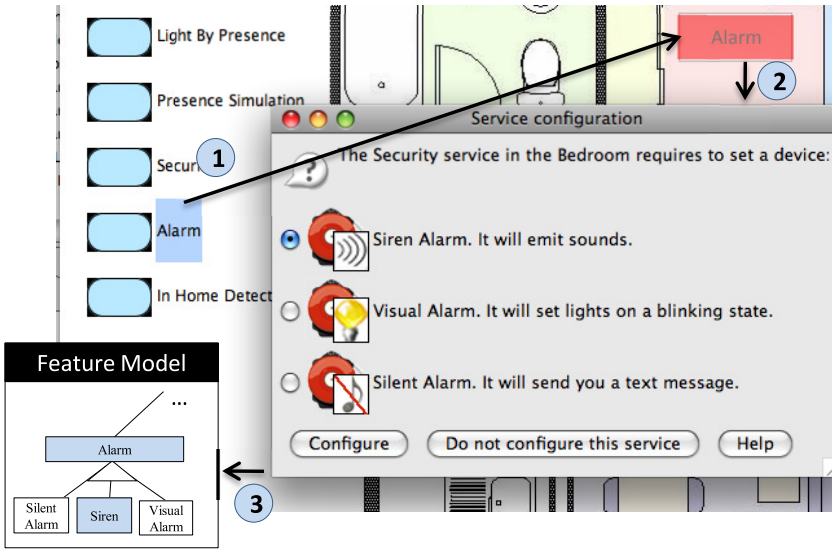


Fig. 6.14: Aplicando patrones en una relación *Opcional* o *Single Choice*

Cuando una característica A tiene una relación *Multiple Choice* con otras características, una o más tienen que ser seleccionadas. En la herramienta, cuando un usuario selecciona y representa una característica con esta relación, presenta un diálogo con algunos servicios/dispositivos que representan las características relacionadas si el número es elevado o todas las características. Entonces, el usuario final selecciona una o más características y la herramienta las añade a la localización correspondiente. Además, el modelo de características se actualiza de acuerdo a esta selección. La Fig. 6.15 presenta un ejemplo representativo de cómo el usuario final selecciona el servicio de seguridad (*Security*) (1). La característica que representa este servicio tiene una relación *Multiple Choice*. Entonces, la herramienta presenta un diálogo con las características relacionadas (2). Después, el usuario final selecciona el servicio *Alarm* y el servicio *In Home Detection*. Por último, el modelo de características se actualiza activando las características *Security, Alarm* y, *In Home Detection* (3).

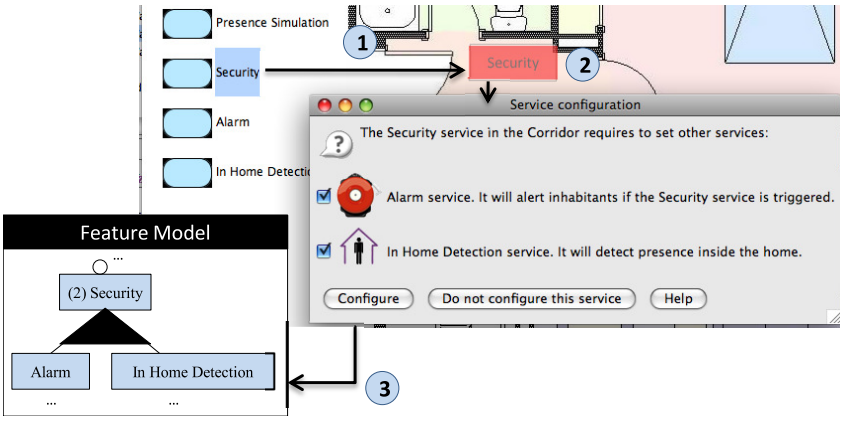


Fig. 6.15: Aplicando patrones en una relación *Multiple Choice*

6.4.3 Ejemplo. Descripción del Sistema

Un ejemplo de las descripciones del usuario final obtenidas utilizando la herramienta propuesta se muestra en la Fig. 6.16. En concreto, esta figura presenta parcialmente los requisitos de una sala de estar de una casa inteligente utilizada como caso de estudio. La Fig. 6.16(A) presenta el entorno físico donde están ilustrados los diferentes servicios requeridos en la sala de estar. La 6.16(B) presenta la visualización de la configuración definida por cada servicio como piezas de puzle. La Fig. 6.16(C) presenta el modelo de características con las características activas de acuerdo a la especificación del usuario final (una descripción del sistema análoga es obtenida en el Repositorio con las información del sistema: entorno físico, localizaciones, usuarios y políticas definidas en los primeros pasos del proceso).

6.5 Tecnologías Utilizadas

Para especificar el modelo de características utilizado en esta herramienta, hemos utilizado el editor *MOSkitt Feature Modeller* (MFM) [66] Este editor utiliza la tecnología proporcionada por la *Eclipse Modelling Platform* [67].

La Fig. 6.17 presenta una captura de pantalla del editor MFM. La

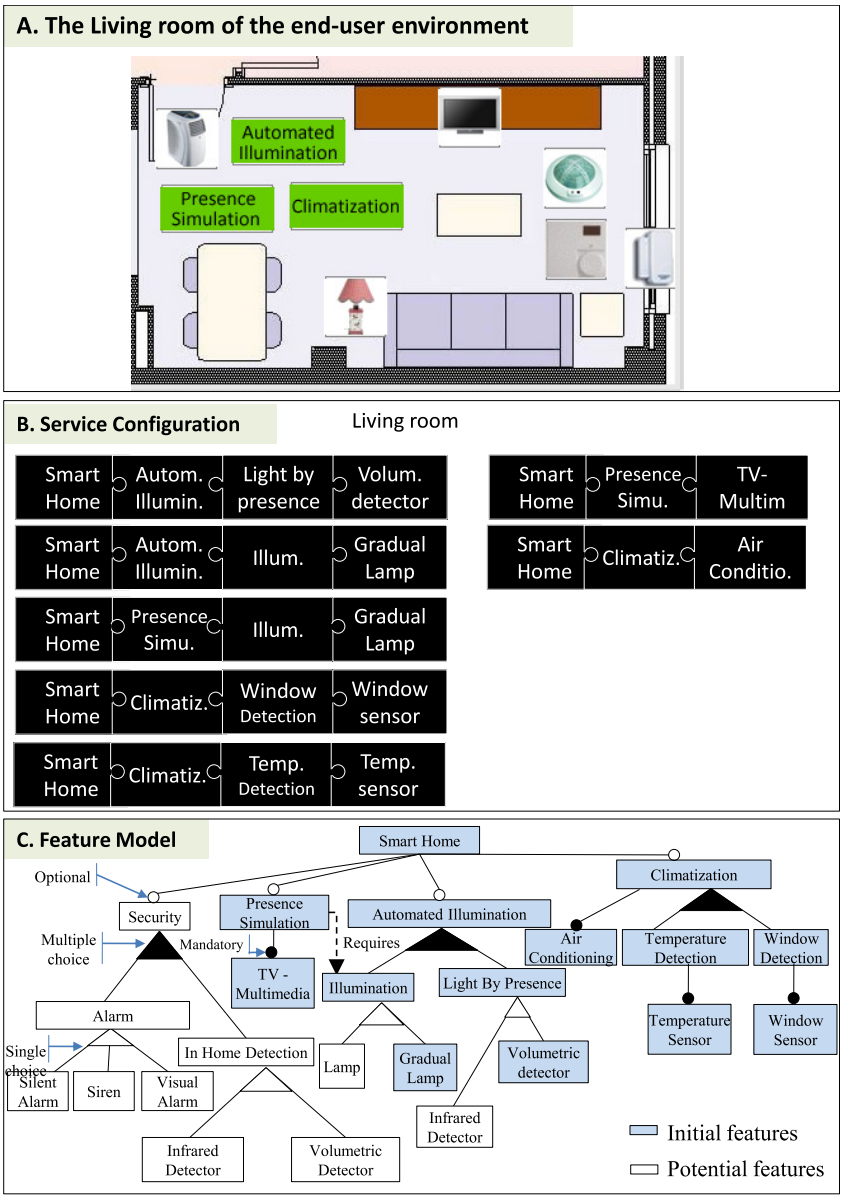


Fig. 6.16: Ejemplo. Requisitos para una casa inteligente

parte superior de la figura representa un modelo de características donde las características de color naranja representan las características activas (estas han sido requeridas por el usuario final en la herramienta). La parte inferior de la figura, presenta las propiedades de la característica seleccionada. Seleccionar una caracterítica determina si el usuario final

la requiere o no.

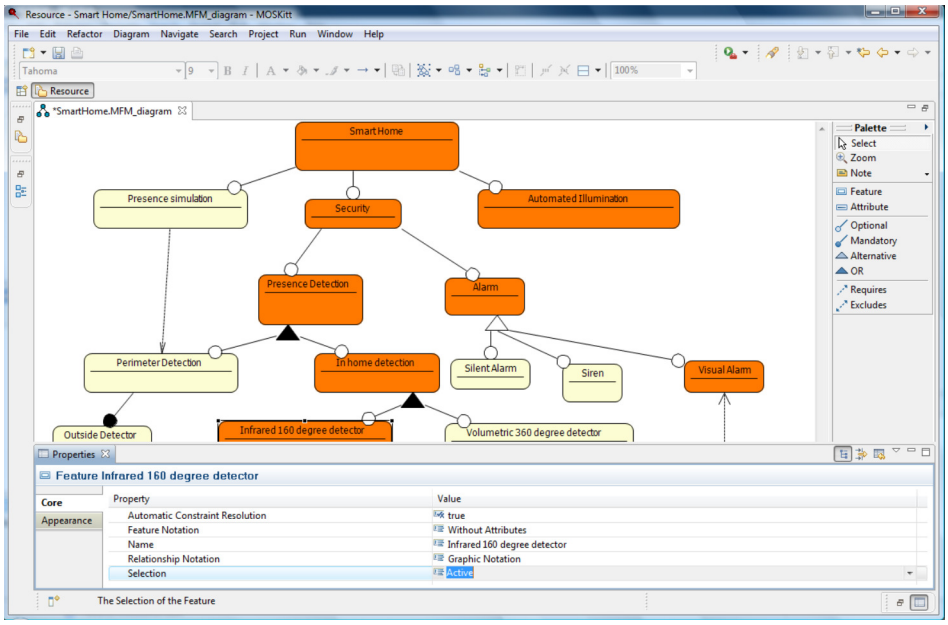


Fig. 6.17: Example. Feature model snapshot using MFM

Para conectar la herramienta con el modelo de características, hemos utilizado *EMF Model Query framework* [68]. *EMF Query* proporciona una API para construir y ejecutar instrucciones de consulta (*query*). Estas instrucciones de consulta pueden ser utilizadas para descubrir y modificar elementos.

En *EMF Query* existen dos instrucciones de consulta disponibles: *SELECT* y *UPDATE*. La instrucción SELECT ofrece consulta sin modificaciones, mientras que la instrucción UPDATE ofrece consultas con modificación. La instrucción SELECT requiere de dos cláusulas, una "FROM" y un "WHERE". La cláusula FROM describe la fuente de los elementos del modelo donde SELECT puede iterar a fin de obtener resultados. La cláusula WHERE describe los criterios para que un elemento del modelo que coincida. La condición establecida para en la cláusula WHERE cae bajo una condición especializada llamada *EObjectCondition* que está especialmente diseñada para evaluar los elementos del modelo.

Hemos implementado los patrones de interacción descritos en la

Subsección 6.4.2 utilizando *EMF Model Query*. Por ejemplo, cuando un usuario final selecciona un servicio de Alarma, la herramienta verifica el modelo de características para la característica seleccionada. También verifica las relaciones con otras características. En este caso, el servicio de Alarma está relacionado con una relación *Single choice* con tres características *(Silent Alarm, Siren* y *Visual Alarm)*. Por tanto, los patrones de interacción aplicados son (ver sección previa): (1) Ofrecer todas las opciones y (2) Autocompletar. Por tanto, necesitamos SELECT para obtener las características hijas relacionadas y, UPDATE para actualizar tanto la característica seleccionada como las características relacionadas.

A continuación se presenta la *query* que hemos implementado para obtener las características hijas de una relación *single choice* utilizando EMF:

```
1 SELECT statement =
2  new SELECT(
3   new FROM(currentFeature.getContents()),
4   new WHERE(new EObjectReferenceValueCondition(
5   new EObjectTypeRelationCondition(
6           FeatureModelPackagePackage.eINSTANCE
7                .getFeatureRelationship())),
8   FeatureModelPackagePackage.eINSTANCE.
9                    getFeatureRelationship_From(),
10  new EObjectInstanceCondition(SingleChoice))
11   )
12 );
```

Dada una característica (*currentFeature*) la instrucción de consulta SELECT recupera todas las características relacionadas con la *currentFeature* con una relación *Single choice (EObjectInstanceCondition(SingleChoice))*. Entonces, estas características son mostradas como servicios en el dialogo de la Fig. 6.14.

Una vez que el usuario final selecciona una de las opciones presentadas, el estado de la característica seleccionada y su característica relacionada es actualizada en el modelo de características de desactivadas a activadas:

```
1 UPDATE statement =
```

```
2  new UPDATE(
3     new FROM(featureModel.getContents()),
4     new WHERE(new EObjectAttributeValueCondition(
5        FeatureModelPackagePackage.eINSTANCE
6                    .getNamedElement_Name(),
7        new StringValue(featureID))),
8        new SetFeatureToState(active)
9  );
```

La declaración anterior actualiza el Modelo de características de la siguiente forma: teniendo en cuenta un identificador *featureID* como entrada, la instrucción de actualización busca una característica con el mismo *featureID (Siren Alarm)*. Entonces, el estado de la característica se establece a activado. Utilizamos esta declaración para cada característica a actualizar su estado. Por el contrario, si el usuario final arrastra una característica a la papelera, su estado se actualiza de activada a desactivada.

Capítulo 7

Conclusiones

En esta tesis de máster se ha introducido a los usuarios finales en las etapas tempranas del proceso de desarrollo de sistemas pervasivos. En concreto, se ha propuesto una aproximación metodológica inspirada en técnicas y metáforas *End-user Development*, que permite a los usuarios finales describir su hogar inteligente. Estas descripciones se han integrado en un método de Desarrollo Dirigido por Modelos.

Este capítulo resume las contribuciones obtenidas en este trabajo, publicaciones relaciones y trabajos futuros.

7.1 Contribuciones

El trabajo de esta tesis está principalmente relacionado con: (1) ***End-user Development*** porque ha introducido a los usuarios finales en las etapas tempranas del proceso de desarrollo de su hogar inteligente mediante técnicas y metáforas conocidas en este campo. (2) **Desarrollo Dirigido por Modelos** (MDD) porque se ha planteado una aproximación metodológica para introducir a los usuarios finales en PervML un método MDD desarrollado previamente. (3) **Línea**

de Productos Software porque se ha planteado una aproximación metodológica para introducir a los usuarios en el método MDDSPL, una Línea de Productos Software guiada por Modelos para el desarrollo de sistemas reconfigurables desarrollado previamente.

A continuación, se describen las contribuciones del trabajo realizado en esta tesis de máster. En el Capítulo 4 se ha analizado cómo introducir a los usuarios finales en los métodos PervML y MDDSPL para que participen en las etapas tempranas del proceso de desarrollo ya que actualmente no pueden participar en estos métodos. Para ello, hemos definido una aproximación metodológica para cada método que plantea dónde participan los usuarios finales. En esta tesis de máster, dado el tiempo y esfuerzo de desarrollar las aproximaciones definidas, nos hemos centrado en la aproximación metodológica que extiende el método MDDSPL, planteando como trabajo futuro desarrollar la aproximación metodológica que extiende el método PervML. En el Capítulo 5, se ha descrito en detalle la aproximación MDDSPL propuesta. En este capítulo, se presenta un proceso que hemos definido para describir qué fases y pasos deben seguir los usuarios finales para crear la descripción del sistema, cómo interactúan los usuarios finales con los analistas y cómo es la representación visual que permite a los usuarios finales describir su hogar inteligente. En el Capítulo 6 se ha presentado la herramienta que hemos desarrollado para soportar las descripciones de los usuarios finales y la aproximación propuesta.

7.2 Publicaciones

Se han presentado parte de los resultados presentados en esta tesis en distintos foros con revisiones por pares. A continuación, se presentan las publicaciones relacionadas con esta tesis:

1. **Francisca Pérez**, Carlos Cetina, Pedro Valderas and Joan Fons. *Towards End-User Development of Smart Homes by means of Variability Engineering.* Third International Workshop on Variability Modelling of Software-intensive Systems. Sevilla, Spain. 2009.

2. **Francisca Pérez**, Pedro Valderas and Joan Fons. *Enabling End-users Participation in an MDD-SPL Approach.* 1st International Workshop on Model-driven Approaches in Software Product Line Engineering (MAPLE 2009). San Francisco, California, USA. 2009.

3. **Francisca Pérez** and Pedro Valderas. *Allowing End-users to Actively Participate within the Elicitation of Pervasive System Requirements through Immediate Visualization.* Fourth International Workshop on Requirements Engineering Visualization (REV'09). Atlanta, Georgia, USA. 2009.

4. **Francisca Pérez** and Pedro Valderas. *A Tool-supported Natural Requirements Elicitation Technique for Pervasive Systems centred on End-users.* XIV Jornadas de Ingeniería del Software y Bases de Datos. San Sebastián, Spain. 2009.

7.3 Trabajos Futuros

Es necesario proporcionar a los usuarios finales mecanismos y herramientas que les permitan describir y personalizar su sistema sin perder funcionalidad ni expresividad de los métodos MDD que les dan soporte.

Para cumplir con esto, proponemos los siguientes trabajos futuros:

- Investigar más técnicas *End-user Development* que permitan mapear a más conceptos de modelado y ofrecer así, más expresividad a los usuarios finales e independencia respecto a los analistas.

- Evaluar mediante experimentos con usuarios finales si las técnicas *End-user Development* aplicadas mejoran la usabilidad y son más intuitivas para los usuarios finales que los métodos PervML y MDDSPL.

- Extender la aproximación metodológica propuesta para que utilizando técnicas *End-user Development*, los usuarios finales

puedan participar en la actividad de mantenimiento del proceso de desarrollo. Así, ellos mismos podrían especificar nuevas necesidades e integrar estos cambios con el método MDDSPL para el desarrollo de sistemas reconfigurables y evolucionar ellos mismos el sistema.

- Describir en detalle y desarrollar el punto de vista del usuario final propuesto en la aproximación metodológica para introducir a los usuarios finales en PervML.

Bibliografía

[1] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):94–104, 1999.

[2] Gregory D. Abowd and Elizabeth D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, 7(1):29–58, 2000.

[3] Albrecht Schmidt and Lucia Terrenghi. Methods and guidelines for the design and development of domestic ubiquitous computing applications. *percom*, 00:97–107, 2007.

[4] Rob Hague and Peter Robinson. End-user programming of reconfigurable systems. *Softw., Pract. Exper.*, 36(11-12):1285–1306, 2006.

[5] Mark W. Newman, Jana Z. Sedivy, Christine M. Neuwirth, W. Keith Edwards, Jason I. Hong, Shahram Izadi, Karen Marcelo, and Trevor F. Smith. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In *DIS '02: Proceedings of the 4th conference on Designing interactive systems*, pages 147–156, New York, NY, USA, 2002. ACM.

[6] Jan Humble, Andy Crabtree, Terry Hemmings, Karl-Petter Ãkesson, Boriana Koleva, Tom Rodden, and Pr Hansson. Playing with the bits - user-configuration of ubiquitous domestic

environments. In *Proceedings of Fifth Annual Conference on Ubiquitous Computing, UbiComp 2003*, Seattle, Washington, USA, 2003.

[7] *End user empowerment in human centered pervasive computing*, 2002.

[8] M. K. Lee, S. Davidoff, J. Zimmerman, and A. K. Dey. Smart homes, families and control. In *Proceedings of Design & Emotion 2006*, 2006.

[9] Javier Muñoz and Vicente Pelechano. Building a software factory for pervasive systems development. In *CAiSE*, pages 342–356, 2005.

[10] Henry Lieberman, Fabio Paternò, and Volker Wulf. *End User Development*. Springer, 2006.

[11] Brad A. Myers, Andrew J. Ko, Sun Y. Park, Jeffrey Stylos, Thomas D. Latoza, and Jack Beaton. More natural end-user software engineering. In *WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering*, pages 30–34, New York, NY, USA, 2008. ACM.

[12] The natural programming project, 1 2009.

[13] Brad A. Myers, John F. Pane, and Andy Ko. Natural programming languages and environments. *Commun. ACM*, 47(9):47–52, September 2004.

[14] Margaret Burnett, Curtis Cook, and Gregg Rothermel. End-user software engineering. *Commun. ACM*, 47(9):53–58, 2004.

[15] Henry Lieberman. Programming by example (introduction). *Commun. ACM*, 43(3):72–74, 2000.

[16] J.S.Y. Chin, V. Callaghan, and G. Clarke. A programming-by-example approach to customising digital homes. *Intelligent Environments, 2008 IET 4th International Conference on*, pages 1–8, July 2008.

[17] Chin, Callaghan, and Clarke. An end-user programming paradigm for pervasive computing applications. *International Conference on Pervasive Services*, 0:325–328, 2006.

[18] Alexander Repenning and Corrina Perrone. Programming by example: programming by analogous examples. *Commun. ACM*, 43(3):90–97, 2000.

[19] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. A cappella: programming by demonstration of context-aware applications. In *CHI '04*, pages 33–40, New York, USA, 2004.

[20] Khai N. Truong, Elaine M. Huang, and Gregory D. Abowd. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In *in Proceedings of Ubicomp 2004*, pages 143–160, 2004.

[21] The accord toolkit. http://www.sics.se/accord/toolkit.html, 1 2009.

[22] Lyubov Kolos-mazuryk. Requirements engineering for pervasive services. In RE05 Doctoral Consortium, 2005.

[23] Javier Muñoz. *Model Driven Development of Pervasive Systems. Building a Software Factory.* Tesis doctoral en informática, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2008.

[24] Javier Muñoz and Vicente Pelechano. Applying software factories to pervasive systems: A platform specific framework. In *ICEIS (3)*, pages 337–342, 2006.

[25] Javier Muñoz, Vicente Pelechano, and Carlos Cetina. Implementing a pervasive meeting room: A model driven approach. In *IWUC*, pages 13–20, 2006.

[26] Software Engineering Institute. Carnegie Mellon. Software product lines. http://www.sei.cmu.edu/productlines/, 2009.

[27] Carlos Cetina. Applying software product lines to build autonomic pervasive systems. Master's thesis, Universidad Politécnica de Valencia, 2008.

[28] *Software product lines: practices and patterns.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[29] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic computing through reuse of variability models at runtime: The case of smart homes. 42(10):37–43, October 2009.

[30] Michael G. Christel;Kyo C. Kang. Issues in requirements elicitation. Technical report, 09/1992 1992. Prepared for the SEI Joint Program Office, HQ ESC/AXS 5 Eglin Street, Hanscom AFB, MA 01731-2116. Sponsored by the U.S. Department of Defense.

[31] Maria Francesca Costabile, Piero Mussio, Loredana Parasiliti Provenza, and Antonio Piccinno. End users as unwitting software developers. In *WEUSE '08: Proceedings of the 4th international workshop on End-user software engineering*, pages 6–10, New York, NY, USA, 2008. ACM.

[32] Joseph A. Goguen and Charlotte Linde. Techniques for requirements elicitation. In Stephen Fickas and Anthony Finkelstein, editors, *Requirements Engineering '93*, pages 152–164. IEEE, 1993.

[33] Oscar Dieste Dante Carrizo and Natalia Juristo. Study of elicitation techniques adequacy. In *11th Workshop on Requeriments Engineering (WER)*, 2008.

[34] Hugh R. Beyer and Karen Holtzblatt. Apprenticing with the customer. *Commun. ACM*, 38(5):45–52, 1995.

[35] Francisca Pérez, Carlos Cetina, Pedro Valderas, and Joan Fons. Towards end-user development of smart homes by means of variability engineering. In David Benavides, Andreas Metzger, and Ulrich W. Eisenecker, editors, *VaMoS*, volume 29 of *ICB Research Report*, pages 103–110. Universität Duisburg-Essen, 2009.

[36] Stewart Brand and Penguin U. S. A. Paper. *How Buildings Learn: What Happens After They're Built.* Penguin Books, October 1995.

[37] Fahrmair, M., Sitou, W., and Spanfelner, B. Unwanted behavior and its impact on adaptive systems in ubiquitous computing. *ABIS 2006: 14th Workshop on Adaptivity and User Modeling in Interactive Systems*, October 2006.

[38] Andrés Wilson Brotto Furtado. Sharpludus: Improving game development experience through software factories and domain-specific languages. Master's thesis, Centro de Informática Federal de Pernambuco, 2006.

[39] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* Wiley, August 2004.

[40] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications with the tota middleware. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 263, Washington, DC, USA, 2004. IEEE Computer Society.

[41] Michael Rohs and Jürgen Bohn. Entry points into a smart campus environment " overview of the ethoc system. In *ICDCSW '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 260, Washington, DC, USA, 2003. IEEE Computer Society.

[42] George Roussos, Juha Tuominen, Leda Koukara, Olli Seppala, Panos Kourouthanasis, George Giaglis, and Jeroen Frissaer. A case study in pervasive retail. In *WMC '02: Proceedings of the 2nd international workshop on Mobile commerce*, pages 90–94, New York, NY, USA, 2002. ACM.

[43] Martijn van Welie, Hallvard Trætteberg, and Hallvard Trætteberg. Interaction patterns in user interfaces. In *Proc. Seventh Pattern Languages of Programs Conference: PLoP 2000*, pages 13–16, 2000.

[44] Christopher Scaffidi, Mary Shaw, and Brad Myers. The "55m end-user programmers" estimate revisited, 2005.

[45] S. Reiss. *Visualization for software engineering–programming environments,*. MIT Press, Cambridge, MA, 1998.

[46] Henry Lieberman and Christoper Fry. *ZStep 95: A reversible, animated source code stepper.* MIT Press, Cambridge, MA–London, 1998.

[47] Doug Kimelman, Bryan Rosenburg, and Tova Roth. *Visualization of Dynamics in Real World Software Systems.* MIT Press, Cambridge, MA, 1998.

[48] S. G. Eick, T. L. Graves, A. F. Karr, A. Mockus, and P. Schuster. Visualizing software changes. *IEEE Trans. Software Engineering*, 28(4):396–412, Apr 2002.

[49] S. Eick. Maintenance of large systems. pages 315–328, 1998.

[50] Jakob Nielsen. *Usability Engineering.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[51] John Steinmetz. *Computers and Squeak as Environments for Learning.* 2000.

[52] David Canfield Smith, Allen Cypher, and Jim Spohrer. Kidsim: programming agents without a programming language. *Commun. ACM*, 37(7):54–67, 1994.

[53] Andrew J. Ko and Brad A. Myers. Designing the whyline: a debugging interface for asking questions about program behavior. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 151–158, New York, NY, USA, 2004. ACM Press.

[54] David Sellier and Mike Mannion. Visualising product line requirement selection decision inter-dependencies. *Requirements Engineering Visualization, First International Workshop on*, 0:7, 2007.

[55] Blaine A. Price, Ronald M. Baecker, and Ian S. Small. An introduction to software visualization. In J. Stasko, J. Dominique, M. Brown, and B. Price, editors, *Software Visualization*, pages 4–26, London, England, 1998. MIT Press.

[56] Christopher Lueg. On the gap between vision and feasibility. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 45–57, London, UK, 2002. Springer-Verlag.

[57] Paul Dourish. *Where the Action Is : The Foundations of Embodied Interaction (Bradford Books)*. The MIT Press, September 2004.

[58] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Comput. Networks*, 51(2):456–479, 2007.

[59] D. Benavides, Ruiz A. Cortés, and P. Trinidad. Automated reasoning on feature models. *CAiSE 2005*, 3520:491–503, 2005.

[60] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez. Fama framework. In *SPLC*, 2008.

[61] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving models with the eclipse amw plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany*, 2006.

[62] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez. Fama framework. In *12th Software Product Lines Conference (SPLC)*, 2008.

[63] Martijn van Welie and Hallvard Trætteberg. Interaction patterns in user interfaces. In *PLoP 2000*, pages 13–16, 2000.

[64] Mick P. Couper, Roger Tourangeau, Frederick G. Conrad, and Scott D. Crawford. What they see is what we get: response options for web surveys. *Soc. Sci. Comput. Rev.*, 22(1):111–127, 2004.

[65] Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[66] Moskitt feature modeller. www.pros.upv.es/labs/projects/mfm.

[67] Eclipse modelling framework. http://www.eclipse.org/modeling/.

[68] Emf model query. http://www.eclipse.org/modeling/emf/?project=query.

www.pros.upv.es

Centro de Investigación en Métodos
de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n
Building 1F
46007 Valencia
Spain
Tel: (+34) 963877007 (Ext. 83530)
Fax: (+34) 963877359