ICT-UNPA-117-2015
ISSN: 1852-4516
Aprobado por Resolución N° 0662/15-R-UNPA

ANÁLISIS DE DESARROLLO DE SOFTWARE ORIENTADO A FEATURE - LÍNEA DE PRODUCTO DE SOFTWARE PARA APLICACIONES DE TVDI

Lic. Mirtha Fabiana Miranda, GISP, ITA, mirfamir@gmail.com
Dra. Sandra Isabel Casas, GISP, ITA, scasas@unpa.com.ar
Dra. Claudia Andrea Marcos, ISISTAN, UNICEN, cmarcos@exa.unicen.edu.ar
Universidad Nacional de la Patagonia Austral, Unidad Académica Río Gallegos
Río Gallegos, Abril 2015

Resumen. El Desarrollo de Software Orientado a Features (FOSD) consiste en la construcción de sistemas a partir de un conjunto de características, cada feature o característica es una unidad funcional que satisface un requisito de software. Las fases son análisis del dominio, diseño e implementación del dominio, configuración y generación del producto de software, obteniendo así una Línea de Productos de Software (SPL). El presente trabajo realiza una comparación de herramientas para su implementación y, se analiza un conjunto de aplicaciones que modelan features. Resultando un espacio de estudio abierto para modelar aplicaciones para la Televisión Digital Interactiva (TVDi). Además se han encontrado diversos métodos, artefactos y patrones de diseño para el modelado de una aplicación de TVDi en particular, reflejando un estudio para el modelado de variabilidad. El objetivo principal es estudiar SPL / FOSD, empleando los Patrones de Diseño de Interacción (PDI) para el dominio de las aplicaciones de TVDi.

Palabras claves: Desarrollo de Software Orientada a Feature, Modelo de Features, Línea de Producto de Software, Patrones de Diseño de Interacción, Televisión Digital Interactiva.

1. INTRODUCCION

El Desarrollo de Software Orientado a Feature, FOSD, es un paradigma para la construcción de sistemas, a partir de las características propias. Una característica (feature) es una unidad funcional de un sistema de software que satisface un requisito, representa una decisión de diseño, y puede generar una opción de implementación. Se utilizará Modelos de Features (FM) para describir las funcionalidades comunes y las variantes de una familia de productos de software. Los FM se utilizan para describir las propiedades o funcionalidades (features) obligatorias, opcionales y alternativas dentro de un dominio. La idea básica de FOSD es modularizar el software en módulos de features que representan propiedades comunes. Diferentes sistemas pueden compartir features comunes y diferir en otras features, este conjunto de features se denomina Línea de Productos de Software (SPL). Una SPL es una familia de sistemas relacionados a un dominio en particular, cuyos artefactos de implementación son compartidos. Para el desarrollo de una SPL, primero se analiza el dominio e identifican las diferencias y similitudes, entre las features o funciones de ese dominio [Som05] [AK09] [Gon07].

Una SPL sugiere una nueva forma de desarrollo de software, en la que ya no se desarrolla un software para cada producto diferente, sino una línea de productos. De esta manera, los productos que comparten algunas de sus features pueden ser desarrolladas a partir de artefactos comunes sin la necesidad de empezar desde cero. La ingeniería de una SPL presenta diferentes métodos de modelado que expresan las features comunes y la variabilidad entre los productos de una línea, denominándose Modelos de Variabilidad (VM). Los VM generan una gran cantidad de features y combinaciones entre ellas. Por lo tanto, es imposible gestionar grandes modelos sin la ayuda de herramientas. El análisis automático de los VM permite extraer automáticamente datos importantes de los MF, las cuales pueden ser útiles a los analistas, diseñadores, programadores y gestores de SPL [BFGR13].

En la Sección 4 se analizan diversas aplicaciones que modelan features, resultando un espacio de estudio abierto para modelar aplicaciones para la Televisión Digital Interactiva (TVDi). La TV Digital es una tecnología que está transformando la televisión, presenta imagen, sonido en alta definición e interactividad, esto posible gracias al pasaje de la transmisión analógica a la digital, la cual permite enviar datos, video, audio, aplicaciones, etc. a través de los canales de transmisión. Estas señales digitales son más eficientes que las analógicas y tienen como principal ventaja la posibilidad de enviar servicios a través del mismo canal, permitiendo un uso eficiente del espectro de transmisiones. Para tener acceso a la TV Digital (terrestre), cada televidente deberá contar con un equipo receptor o puede estar integrado en los televisores [OHM+13] [BZ13]. Con la llegada de la TVD en Argentina se creó del Sistema Argentino de Televisión Digital Terrestre (SATVD-T), que comenzó oficialmente sus transmisiones en 2010 con una cantidad reducida de antenas transmisoras que se planea ir ampliando junto con redes de fibra óptica para llegar a todo el país y cuyo control y monitoreo está a cargo de la empresa estatal ARSAT [ARSAT] [OHC14]. Lo mismo ocurre con la oferta de canales, señales de noticias y otros. A partir de este proceso se estima que el llamado “apagón analógico” en Argentina se producirá en el año 2020, cuando la televisión pase a ser exclusivamente digital. Las políticas impulsadas por el Gobierno argentino para la implementación y puesta en marcha del SATVD-T, fomenta una excelente oportunidad para sentar un precedente en materia de desarrollo de software de TVDi [NM13]. En la Sección 6.1 se detallan la arquitectura, plataforma y lenguaje de programación requeridas para de este tipo de aplicaciones.

Para el desarrollo de aplicaciones de TVDi se han empleado distintos métodos y artefactos, entre ellos notación UML, modelo NCM, métodos ágiles como Scrum, y otras herramientas para la codificación. Los artefactos mencionados se explayan en detalle en la Sección 6.2.
Otro concepto a estudiar son los Patrones de Diseño de Interacción (PDI), los patrones existen desde hace tiempo a finales de los 60, más tarde en los 90 comenzó a adaptarse a la Ingeniería del Software. Los patrones son vistos como formatos o ejemplos de solución, cada patrón describe un problema que ocurre una y otra vez en un entorno y, después, describe la solución a ese problema de tal forma que se puede utilizar esa solución varias veces [DMA05]. En la Sección 6.3 se presentan PDI para el dominio de las aplicaciones de TVDi [Kur09].
El objetivo general que se pretende alcanzar es analizar la potencialidad de los enfoques SPL /FOSD, empleando los PDI para el dominio de aplicaciones de TVDi. Se estudia las aplicaciones existentes que emplean modelos de features y herramientas para su implementación.
La estructura del presente documento se organiza de la siguiente forma, a saber: en la Sección 2, se estudia el concepto de Línea de Producto de Software. En la Sección 3 se describe el Modelado de Feature; FOSD y las fases del proceso. En la Sección 4 se mencionan herramientas que implementan los MF. En la Sección 5 se presentan aplicaciones que emplean features en distintos dominios. En la Sección 6, se presenta el análisis preliminar de SPL - FOSD para el dominio de aplicaciones para TVDi y por último; en la Sección 7 se muestra la conclusión del trabajo desarrollado.

2. LINEA DE PRODUCTOS DE SOFTWARE

La primera aplicación de una SPL surge en 1985, la empresa sueca Celsius Tech System AB, implementaba sistemas para el comando y control de navíos, comprometidos a plazos acordados y nuevos pedidos de clientes; la solución se basó en arquitectura de referencia, técnicas de reutilización en distintas áreas junto con una reorganización de la empresa, la experiencia fue exitosa, pudiendo cumplir con los plazos de ambos sistemas y desarrollar nuevos sistemas. A partir de ese momento se comenzó a construir productos de software con más requisitos en plazos de tiempo cortos, mejorando los tiempos de diseño y el mantenimiento de estos. La reducción de los costos en el desarrollo de proyectos similares gestionados en paralelo, es uno de los grandes beneficios de la SPL [Rod07] [RGC07] [MP14] [CBT+14].
La Ingeniería de Software se ha centrado en el desarrollo de software individual, un sistema a la vez. Un proceso de desarrollo típico comienza con el análisis de los requisitos de un cliente. Después de varias etapas de desarrollo (especificación, diseño, implementación, prueba y despliegue), y como resultado un solo producto de software. La Ingeniería de Líneas de Productos de Software se centra en el desarrollo de múltiples sistemas informáticos similares en un dominio a partir de una base de código común. El proceso de desarrollar una línea completa de productos de software en lugar de una sola aplicación se denomina ingeniería de dominio [KA11]. En cambio la Ingeniería de Producto incluye el desarrollo de los productos para clientes concretos. La figura 1 muestra gráficamente el proceso de ingeniería de producto y su relación con ingeniería de dominio [Dia10].
Una SPL consiste en un conjunto de aplicaciones con una arquitectura común específica de dichas aplicaciones. Cada aplicación contará con su propia especificación de componentes, configuración de componentes adicionales y su propia implementación. El núcleo común de la familia se reutiliza cada vez que se requiere una nueva aplicación [NJ13].

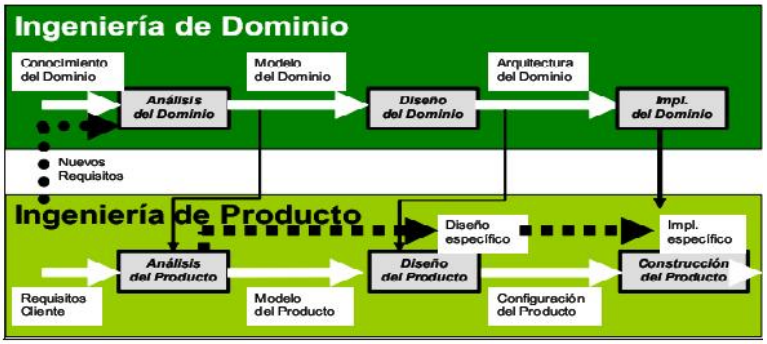

Figura 1. Procesos de una Línea de Producto de Software

Se puede generar distintas versiones de la aplicación según el tipo de especialización de SPL, clasificándose según la plataforma, entorno, funcionalidad y proceso. Estos tipos se definen como a) tipo según la plataforma: se desarrollan versiones para diversas plataformas; b) tipo del entorno: se crean versiones para gestionar el entorno del sistema operativo y dispositivos periféricos; c) tipo según la funcionalidad: se crean versiones para clientes concretos que tienen diferentes requisitos y; d) tipo según el proceso: adaptar la aplicación para tratar con procesos de negocios concretos.

Una SPL permite la reconfiguración, en la cual se pueden agregar o eliminar componentes del sistema, definir parámetros y restricciones para los componentes, e incluir conocimientos de los procesos de negocio. Esta configuración pueden realizarse en dos puntos del proceso de desarrollo, a saber: a) durante el despliegue o; b) durante el diseño. La configuración durante el despliegue es la utilización en paquetes de software verticales que son diseñados para una aplicación en particular; se pueden mencionar los sistemas de Planificación de Recursos de Empresas (ERP), estos consisten en sistemas integrados y diseñados para soportar procesos de negocios.

Una SPL surge a partir de aplicaciones existentes, por ejemplo si la organización ya tiene una aplicación, se utiliza esta como base para generar una nueva, son instancias y especializaciones de arquitectura de aplicaciones más genérica. Un sistema genérico se adapta y específica para satisfacer requerimientos específicos de funcionalidades, plataforma de destino o configuración de funcionamiento [Som05] [SGB01] [Dia10] [MSC+14].

Todos los productos de una SPL proveen un conjunto de funcionalidades comunes, pero cada producto difiere de los demás en el conjunto de funcionalidades opcionales (variables) que implementa; entonces la variabilidad es la diferencia funcional entre productos de una SPL [GPA+07].

La Figura 2 ilustra la evolución en los costes entre una SPL y el desarrollo convencional. A mayor número de productos, mayor esfuerzo y coste. En un entorno convencional se realiza una copia de una aplicación, y con el tiempo la copia evoluciona en forma independiente. Aquí el parecido entre las dos aplicaciones sirve para agilizar el desarrollo, pero en ningún caso el mantenimiento. Cada aplicación se mantiene de forma independiente, incluso si este mantenimiento es similar también para ambas. La reutilización es oportunista, y la gestión de la semejanza (partes comunes y partes variables) es secundaria. Por el contrario, una SPL realiza inicialmente un estudio para analizar el dominio, determina las variaciones a soportar, y desarrolla la plataforma de SPL. Antes de comenzar con el primer producto, ya se tiene el coste de la inversión inicial. La intersección entre las dos líneas de la Figura 2 muestra el punto a partir del cual se empieza a rentabilizar la SPL. A partir de aquí, la generación de un producto es menos costosa con la SPL que con el enfoque tradicional. Sin embargo, según estudios realizados, se requiere de más de tres productos para compensar los gastos iniciales de desarrollo [Dia10].

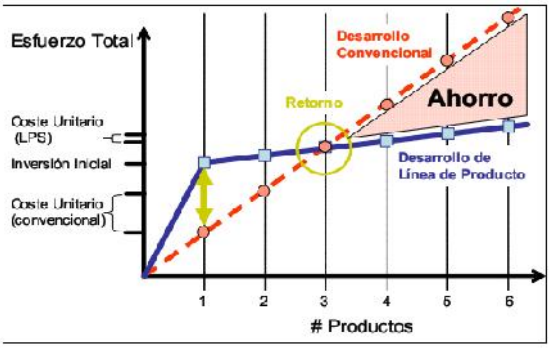

Figura 2. Línea de Producto de Software vs desarrollo convencional

La construcción de una SPL permite la reutilización de aplicaciones obteniendo alta calidad, más rápidos y, a menor costo [Pen09]. Los beneficios relativos a la calidad que una SPL aporta, se puede medir de dos formas; la primera mediante el grado de precisión con que cada producto se ajusta a las necesidades de cada cliente. Esta medida depende del grado de variabilidad de la SPL. A mayor variabilidad, más probabilidades de adaptar el producto a los gustos del cliente. Pero, normalmente, esta variabilidad tiene un costo, y el reto es encontrar el equilibrio entre costo y variabilidad. A diferencia de los enfoques tradicionales, en una SPL la variabilidad es un concepto clave. Todo el proceso de desarrollo está guiado por esta noción con el objetivo de abaratar los costes de la variabilidad, y así poder conseguir mayores cotas de variabilidad y, por tanto, de satisfacción de las peculiaridades del cliente. El segundo aspecto, es la tasa de defectos en los productos de la SPL, los beneficios se derivan de la reutilización de los elementos comunes. La continua utilización de estos elementos a lo largo del tiempo hace que finalmente estén muy depurados/probados. Además, los beneficios de encontrar y eliminar un defecto en elementos comunes no se limitan al producto en el lugar que se detecta el error, sino que se dispersa entre todos los productos de la SPL.
En términos de la variabilidad de una línea de productos, la ingeniería de dominio la define y la ingeniería de producto lo explota, seleccionando las features específicas durante la derivación de productos. En particular, la ingeniería de dominio es responsable de establecer el conjunto básico de elementos comunes y de definir la variabilidad de las aplicaciones resultantes de una SPL. Por otro lado, la ingeniería de producto deriva aplicaciones a partir de los artefactos identificados en el proceso anterior. En este proceso se establecen el conjunto de features (configuraciones) del producto final [MDGL13].
La similitud y variabilidad son dos particularidades fundamentales de una SPL, la variabilidad evoluciona constantemente con la extensión de alcances y los cambios en las aplicaciones [Pen09]. Debido a esta complejidad y a esta naturaleza extensiva e intensiva de SPL, la seguridad y la ingeniería de requisitos son mucho más importantes para la puesta en práctica del desarrollo basado en SPL, de lo que ya son para el desarrollo de un Sistema de Información, ya que una brecha de seguridad o vulnerabilidad en la línea puede provocar importantes problemas a largo plazo a todos los productos de la misma. Es por ello que la disciplina conocida como Ingeniería de Requisitos de Seguridad, es una parte muy importante en el proceso de desarrollo software y especialmente dada su complejidad para conseguir SPL seguras, ya que facilitan técnicas, métodos y normas para abordar esta tarea desde las primeras fases del desarrollo e implica el uso de procedimientos repetibles y sistemáticos para asegurar que el conjunto de requisitos obtenidos es completo, consistente y fácilmente comprensible y analizable por parte de los diferentes actores implicados en el desarrollo de la SPL y sus sistemas. Los requisitos de seguridad son importantes desde las primeras fases del ciclo de desarrollo y apoyándose en los estándares de seguridad internacionales más importantes (como ISO/IEC 15408 e ISO/IEC 17799:2005; ISO/IEC 27001), con el objeto de

aportar una perspectiva que permita mejorar la calidad, tanto en la SPL como en los productos de dicha línea, los cuales serán conformes a dichos estándares [MFP08].

3. MODELADO DE FEATURE

Los modelos de features (FM- Feature Model) representan el espacio de posibles configuraciones de todos los productos en una SPL. Permite la visualización de las features jerárquicas y sus relaciones. El primer modelo de features fue propuesto en 1990 [PSF+13].
El objetivo del FM es definir un conjunto de combinaciones válidas de features, también llamados configuraciones. Lo principal es que varios FM solicitados se pueden extraer de las mismas configuraciones de entrada, sin embargo, sólo unos pocos de ellos son significativos y mantenibles. Los FM es la forma más utilizada para la modelización y el razonamiento acerca de la variabilidad de un sistema. A partir de una fórmula y conocimiento, se puede sintetizar una FM precisa, significativa y única. Se ha demostrado la aplicabilidad del principio de ingeniería inversa, refactorización, fusión o despliegue de los FM [RGMS13] [ABH13].
En [BSR10] se presenta un análisis de una revisión de la literatura de FM que explica cómo estos modelos están madurando y la cantidad de herramientas que existen para la industria de Software.

3.1 Feature

Una feature es una unidad de funcionalidad de un sistema de software que satisface un requerimiento, representa una decisión de diseño ó proporciona una opción potencial de configuración. El concepto de feature es el centro de FOSD [KA11] [TC10].
Se estudia las features como funcionalidad o función. La matemática aporta simplicidad, claridad, fundamentación de principios para el diseño y las herramientas de software automatizados, por esta razón se estudiarán a las features como definición de función, desde el punto de vista desde su origen matemático.
De las diversas definiciones [AGMG14], acerca del concepto de función se puede decir que la mayoría describe a la función desde la perspectiva del espacio del problema, es decir, las features son las que describen el resultado esperado de un sistema. Por otra parte otras definiciones presentan una perspectiva del espacio de soluciones, es decir las features son las encargadas de implementar la funcionalidad deseada [Bat09]. Todas las funciones son funciones unarias F(x). Entonces se encuentra un conjunto de funciones:

$$F1, F2, F3, ... Fn$$

La función es un incremento en la funcionalidad del programa. Comenzar con nada (programa vacio), añadir una feature, agregar otra, para producir así el programa P, a saber:

$$P=F1, F2 , F3 , ... Fn$$

Un modelo de features es una representación de una fórmula proposicional, en la que se debe clasificar y/o seleccionar entre las funciones para lograr la SPL deseada, es decir identificar las features y sus combinaciones significativas:

$$P=F1 \cdot F3 \cdot F6 \cdot Ø$$

3.2 Desarrollo de Software Orientado a Features (FOSD)

El Desarrollo de Software Orientado a Features (Feature-oriented Software Development, FOSD) es un paradigma reciente para la construcción, personalización y síntesis de sistemas de software a gran escala, cuyo concepto central es la feature, que puede traducirse como función [KA11]. El software se descompone en términos de las features que provee, para construir software bien estructurado que pueda ser adaptado a las necesidades del usuario y al escenario de aplicación. A partir de un conjunto de features comunes se puede generar una variedad de sistemas de software diferentes que comparten features comunes y difieren en otras, de manera similar al enfoque de SPL [TC10] [LBL06].

3.2.1 Fases del proceso FOSD

En todas las fases se aplica el concepto de feature, FOSD presenta cuatro etapas: análisis, diseño, implementación, estas con respecto al dominio y la última etapa consiste en la configuración y generación del producto [AK09] [LBL06].
Como muestra la Figura 3, en la primera fase se analiza el dominio para identificar las features del dominio y sus relaciones, como parte del sistema, es decir que funciones son admitidas y cuáles no, esto incluye la determinación del alcance del dominio que define la dimensión del dominio. A continuación pero todavía en la etapa de ingeniería de dominio, las features son implementadas como módulos de features, se trata de la fase de diseño e implementación del dominio, el cual se crea un conjunto de artefactos de features relativas al producto. Finalmente, en la configuración y generación de productos se recogen los artefactos de features correspondientes y se crea un software eficiente y confiable [TC11].

Figura 3. Fases de proceso FOSD

A continuación se presentan las distintas fases de Proceso FOSD:

- **Análisis del dominio**

El análisis identifica y captura las variabilidades y puntos en común de un dominio. Cuanta más información se expone en el modelado, mejor se puede guiar al proceso de configuración y generación [AK09].
Un método que existe es el Análisis de Dominio Orientado a Features (FODA). FODA es un modelo compuesto de dos elementos: las features y las relaciones entre ellas. Las features están organizadas en una estructura jerárquica en forma de árbol [Gom11]. Una de las features del árbol es la raíz y representa el sistema como un todo. En esta aproximación las relaciones entre features pueden ser de dos tipos:

a) Relación jerárquica. La relación es definida entre una feature padre y sus features hijas. Una feature hija solamente puede hacer parte de los productos en los que la feature padre aparece. En FODA fueron propuestas tres tipos de relaciones jerárquicas:
   Obligatoria (Mandatory): indica que cuando la feature padre es parte de un producto particular, la feature hija también debe ser parte del producto.

Opcional (Optional): indica que cuando la feature padre es parte de un producto particular, la feature hija, puede o no, ser incluida en el producto .
Alternativa (Alternative): es la relación entre una feature padre y un conjunto de features hijas que indica que cuando la feature padre hace parte de un producto particular, sólo una de las features del grupo de hijas debe hacer parte del producto.

b) Relación no jerárquica.
Excluye (Excludes): Una feature X que excluye a la feature Y significa que si la feature X es incluida en el producto, la feature Y no debe ser incluida, y viceversa.
Requiere (Requires): Una feature X que requiere de una feature Y, significa que si la feature X es incluida en el producto, la feature Y también debe ser incluida, pero no viceversa.

La Figura 4 presenta un FM, los nodos en esta figura representan features y las líneas muestran las relaciones entre ellas. El nodo raíz denominado como A representa el concepto de dominio que se está modelando. Las features de un FM se clasifican en obligatorios, opcionales y alternativas. Las features opcionales como el nodo C representado con un círculo vacío puede ser o no parte de un producto. Las features obligatorias, como B, D y, E, representados por un círculo relleno son parte de todos los productos que contienen la SPL. Las features alternativas pueden ser exclusivo (XOR) o no exclusiva (OR). XOR indica que sólo una sub-feature puede ser seleccionada F o G; OR permite la selección de más de una opción para un producto, H o I o los dos nodos H e I [PSF+13].

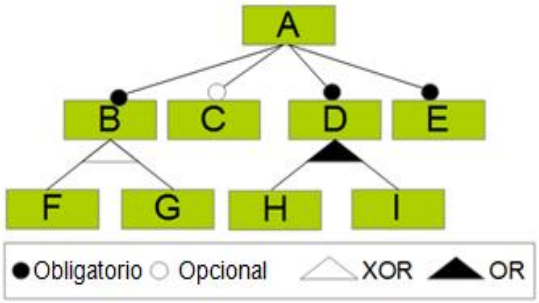

Figura 4. Modelado de dominio

Se encuentran otras extensiones de FODA como por ejemplo, a) Método de Reuso Orientado a Feature (FORM), b) Árbol de programación de feature generativas (GPFT), c) Diagrama de Feature basado en Van Gurp (VBFD), d) Diagrama de variabilidad de características (VFD), e) Modelado de Caso de Uso para Línea de Productos y la Ingeniería de Software (PLUSS), f) FeatuRSEB (combinación de FODA y Reutilizar-Driven Software Engineering Business), g) Modelado de Feature basados en UML (UML-BFM), h) Modelado de Feature integrados basado en modelado en UML (IFM-UML), entre otros [PSF+13].
También existe en la literatura de FOSD el método ciFeature, para la creación de features independientes del contexto, en el que el contexto pueden ser herramientas de desarrollo IDEs, lenguajes, o códigos heredados, permitiendo aumentar la reusabilidad [KPF11].
Además de la representación gráfica, la variabilidad en una SPL puede ser representado por los modelos basados en texto. Estos modelos utilizan texto estructurado para describir features y sus relaciones. Los ejemplos de los idiomas que se utilizan para escribir modelos basados en texto son los siguientes: Lenguaje Pruebas Variabilidad (TVL), Clafer, GUIDSL, y SXFM[1] [PSF+13].

[1] SXFM siglas de Simple Feature XML Modelo, un formato textual concisa para denotar modelos de características.

- **Diseño y especificación del dominio**

Es el proceso de definición de la arquitectura de una SPL. Las features se especifican mediante una especificación formal, es decir mediante un lenguaje de modelado.
El Desarrollo Dirigido por Modelo (MDD) es una metodología de la Ingeniería de Software que eleva el desarrollo de software a un mayor nivel de abstracción, es decir trata de unificar las funciones de las trasformaciones. Las features son las transformaciones verticales y las transformaciones de MDD son transformaciones horizontales. Ambas son funciones, que se utilizan en diferentes circunstancias.
También se puede utilizar el lenguaje de modelado unificado (UML), para modelar las features, provee una notación gráfica para el modelado de diferentes aspectos tanto en el ambiente académico como en desarrollos industriales [AK09] [Gon07].
El Diseño orientado a features consiste de dos pasos: a) Funciones de modelación por separado y, b) de razonamiento acerca de las combinaciones de features. El modelado de funciones por separado se trata en un diseño orientado a features que consiste en un conjunto de fragmentos de modelo, cada uno de los cuales corresponde a la parte del diseño que se refiere a una feature. El beneficio de los FM en unidades distintas son:
I. Separación de preocupaciones (Separation of concerns:): Un diseño orientado a features permite a un desarrollador estructurar un diseño de software a lo largo de features, lo que facilita la comprensión, capacidad de mantenimiento y capacidad de evolución.
II. Variabilidad: Las features individuales de un diseño pueden estar compuestas de diferentes combinaciones que producen diversos diseños.
III. Reutilización: Las features pueden ser reutilizados en diferentes variantes de diseño sin replicar la información, que no es posible con diseños monolíticos.
En el razonamiento de las combinaciones de las features, un conjunto de features que estructuran el diseño, el desarrollador puede razonar acerca de sus dependencias e interacciones. Las features pueden relacionarse con otras features de diferentes maneras. Estas son las dependencias o iteraciones que se pueden presentar en el diseño [ASLK10].
La teoría de FOSD involucra el concepto de feature interaction (FI) que representa una situación en la que dos o más features exhiben un comportamiento inesperado que no ocurre cuando se utilizan aisladamente. Este problema surgió y fue analizado formalmente en el dominio de los servicios de telecomunicaciones, denominado feature interaction problem (FIP). Sin embargo, FIP se presentará cada vez que los componentes desarrollados independientemente sean requeridos para trabajar de manera conjunta, como ocurre con los servicios Web, que desarrollaron diversas técnicas para detectar y manejar las interacciones, principalmente para especificar y resolver FI a nivel de diseño y especificación [AK09] [TC10] [ASLK10].
En la Figura 5 se muestra un diagrama de clases, cada clase con sus propiedades y métodos; el punto relleno representa la composición de features.
En sistemas distribuidos, la interacción entre features se realiza en tiempo de ejecución.

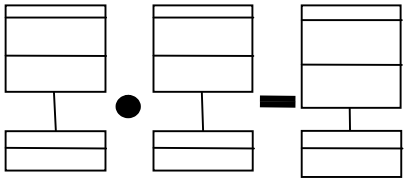

Figura 5. Fase de diseño de dominio

- **Implementación del dominio**

En esta fase se realiza la selección de las features para su posterior implementación en programas, como lo muestra la Figura 6.
La herramienta Jak es un lenguaje que extiende Java por mecanismos orientados a funciones. Se trata de un árbol de contenidos que presenta declaraciones de clases y refinamiento. Una declaración de refinamiento es una forma de aplicar los cambios de una feature sin cambiar el código del programa.
Se propusieron también extensiones de lenguaje orientados a ofrecer para C++, llamado FeatureC++, y para XML.
Una de las primeras herramientas para implementar features fue AHEAD (Diseño de Aplicaciones con Ecuaciones Jerárquicas Algebraicas), AHEAD es para la programación orientado a features. Esta herramienta permite introducir un nuevo lenguaje llamado Jak que es una extensión de Java [Thu08] [Thu13].
También esta FeatureHouse que consiste en un plugin que puede ser utilizado por muchos lenguajes como C, Haskell, JavaCC o XHTML.
Para la implementación del modelado y gestión de SPL, existen otras herramientas disponibles actualmente, tales como SPLOT, FeatureIDE, ClaferMOO, XFeature, FMP, y Pure [PSF+13] [BGF13]. Estas herramientas serán explicadas en detalle en la sección 4.

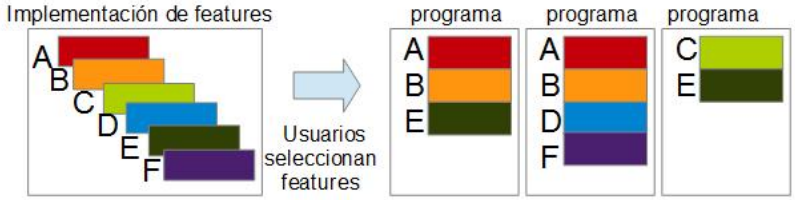

Figura 6. Implementación de programas

- **Configuración y generación del producto**

En esta etapa el FOSD difiere con el enfoque tradicional de dominio, dado que ahora el usuario es el encargado del montaje, adaptación e integración de las features reutilizables en la ID, es decir se encarga de una automatización completa.
Existen varias herramientas que se estudian en la sección 3.3, las cuales ayudan a los usuarios en la selección de features y relaciones entre ellas. El proceso de optimización de una selección de features es una forma de meta-programación arquitectónica. La idea básica es la creación de objetos y métodos que actualizan a otros objetos. Los objetos representan los diseños de sistemas y los métodos son las transformaciones que actualizan esos diseños. Una función de selección representa un diseño de sistema y la optimización es una transformación que se actualiza o cambia el diseño o la arquitectura.
Una vez que el usuario selecciona las features que necesita, se genera el software que desea. Posteriormente hay que verificar que el código generado sea correcto, para ello, se utilizan tres niveles de corrección: corrección sintáctica, tipo de corrección y corrección de comportamiento. La corrección sintáctica, es cuando, el software generado es correcto con respecto a la sintaxis del lenguaje empleado; el tipo de corrección, es cuando, el software generado está bien escrito con respecto al lenguaje; y la corrección de comportamiento, es cuando, el software generado se comporta correctamente de acuerdo con la especificación formal o informal. Además, es deseable para garantizar propiedades de corrección para toda la SPL en lugar de comprobar cada variante del producto de manera aislada. La razón es que de

generación todos los sistemas no son factibles debido al gran número de selección de features [AK09].

3.3 Programación Orientada a Features

La programación orientada a features (FOP) es una técnica de programación que permite la representación de las similitudes y las variaciones de un sistema de software. El concepto principal de esta técnica son las features. A partir de un conjunto de características, muchos sistemas de software diferentes pueden ser generados compartiendo características comunes y difiriendo en otras [AK09].
La FOP surge como respuesta a las limitaciones que el diseño y la Programación Orientada a Objeto (POO) posee en relación a la implementación de una SPL [Per11]. La FOP aparece estrechamente ligada al concepto de SPL dado que la primera es un excelente mecanismo para implementar las segundas. Usando FOP, muchas de las variabilidades existentes en una línea SPL pueden implementarse como features. A la hora de construir un producto específico dentro de una SPL, bastaría con seleccionar las features correspondientes a las variaciones seleccionadas y componerlas. No obstante, no todas las variabilidades de una SPL se implementan adecuadamente mediante el uso de las features. En ocasiones, como variaciones que suponen incrementos muy pequeños de funcionalidad (ej. variar el color de una interfaz gráfica), resultan más adecuadas otras técnicas, como la parametrización o la herencia tradicional de la POO [KA11].
Para la programación se encuentran los siguientes artefactos, a saber:
FeatureC++ es una extensión de lenguaje C++ para apoyar FOP. FeatureC++ permite al programador expresar features en forma modular. Soporta fácil derivación y composición de los miembros de la familia del programa basado en especificaciones declaradas en forma de ecuaciones algebraicas. FeatureC++ también adopta los conceptos del POA para aumentar aún más la modularidad de los temas transversales. Actualmente, soporta capas Aspectual Mixin (AML). AML se combinan las ventajas de los Mixins y aspectos para aumentar la modularidad transversal.
FeatureHouse es un enfoque general para la composición de artefactos de software. FeatureHouse es independiente del lenguaje en que los artefactos de software escritos en varios idiomas puede estar compuesto, por ejemplo, el código fuente, casos de prueba, modelos, documentación y archivos MAKE. Artefactos de software se representan como la estructura de árbol de features, que captura la esencia de la estructura modular de un artefacto en forma de un árbol. Como paradigma de la composición, FeatureHouse admite dos enfoques: FSTComposer, FSTMerge, y un plug-in FSTGenerator. FeatureHouse se puede utilizar con el entorno visual de desarrollo FeatureIDE. Actualmente, FeatureHouse proporciona soporte para la superposición de artefactos de software escrito en Java, C #, C, Haskell, JavaCC, aleación y UML.

3.4. Modelado de Variabilidad

La variabilidad se define como la habilidad de cambio o de personalización de un sistema [AHC+13]. Existen dos grandes grupos en los que se dividen los sistemas con variabilidad: los sistemas personalizables, cuya variabilidad se debe principalmente a la selección del usuario de la parte que le interesa; y las familias de productos, en los cuales una serie de productos más o menos similares se unen para permitir la reutilización de la parte común [GLS04] [KA11] [RS08].

Se pueden distinguir distintos tipos de variabilidad. Una de estas es la variabilidad en el tiempo. Esta variabilidad se define como la existencia de diferentes versiones de un artefacto que son válidos en distintos momentos. Esta variabilidad responde a un proceso natural en los productos de software, la evolución de los sistemas. Un ejemplo de variabilidad en el tiempo puede ser el auto, el sistema de abrir y bloquear puertas. Actualmente este punto de variabilidad puede tener las variantes de llaves o tarjetas magnéticas y con una tecnología más avanzada, puede tener lectores de huella digital, reconocimiento de voz, etc.
Otra tipo de variabilidad es la variabilidad en el espacio; su concepto es la existencia de un artefacto en diferentes formas en un mismo espacio de tiempo. Un ejemplo de esto puede ser la manera de ingresar texto a un editor de texto. Se puede hacer mediante teclado, Voz, etc. Generalmente, este es el tipo de variabilidad que más se estudia, debido a que los productos generados desde una familia de software se comercializan al mismo tiempo.
También se puede definir la variabilidad dependiendo del origen de esta, ya que estas pueden ser definidas de manera externa o interna. La variabilidad externa obedece a los cambios percibidos como necesarios por el cliente; se trata de la variabilidad de artefactos de dominios que es visible a los clientes. La causa de la variabilidad externa generalmente proviene desde necesidades específicas del cliente, pero a veces puede resultar exigible dado el marco legal de la aplicación. Un ejemplo de variabilidad externa es el artefacto por el cual se controla el acceso a una puerta. Esta pueda ser un teclado numérico o/y un lector biométrico, dependiendo de la seguridad que exija el cliente.
La variabilidad interna consiste en la variabilidad de artefactos de dominio que permanecen escondidos al cliente. La causa de la variabilidad interna surge de las variabilidades externas. Esto a su vez genera más refinamiento, cada vez de más bajo nivel. Como al cliente le interesan los aspectos del sistema de más alto nivel, es necesario esconder ciertos puntos que no ayudan a la utilización ni comercialización del producto, ya que lo hacen más complejo. Un ejemplo de esto puede ser dado un lector de huellas digitales, saber qué tipo de almacenamiento y algoritmos de compresión utiliza el sistema para almacenar las imágenes [Urroz12].
Existen patrones que ayudan a lograr la variabilidad en las distintas fases del ciclo de vida en un sistema en tiempo de ejecución, desde el diseño arquitectónico, diseño detallado, implementación, compilación y vinculación e incluso después de la presentación [SGB01].
Otra propiedad importante son los mecanismos de patrones recurrentes. Los mecanismos de variabilidad trabajan en diferentes entidades, algunos de estos mecanismos pueden ser muy similares.
En algún momento se debe elegir entre las variantes y una única variante será seleccionada y ejecutada. Los lugares en los que se espera variación pueden ser:
Antes de la Entrega:

- Arquitectura Producto Derivación: la unión de los puntos de variación es lo que genera un producto en particular.
- Recopilación: La finalización del código fuente se hace durante la compilación, pero puede extenderse con comportamiento adicional.
- Vinculación: esta fase depende de la programación y el entorno de ejecución que se utiliza. En algunos casos se realiza después de la compilación y en otros al inicio del sistema.

Después de la Entrega:

- Comienzo del tiempo de la actividad y del tiempo de ejecución: algunas variaciones pueden ocurrir en el sitio del cliente, por lo que se deben usar parámetros en la puesta en marcha, por ejemplo librerías dinámicas.

- Tiempo de ejecución por llamada (Adaptación al entorno de ejecución): esta variabilidad hace que una aplicación sea interactiva, generalmente se da en los lenguajes orientados a objetos, conocidos como los plugins.

4. HERRAMIENTAS PARA EL MODELADO CON FEATURE

Las distintas herramientas presentadas en la literatura permiten la creación y el manejo de los FM, desde prototipos académicos hasta sistemas comerciales[2] [KA11].
FeatureIDE es una herramienta open source, que permite generar directamente el código (Java o C + +), así como una representación gráfica de los FM. Se trata de un plugin de Eclipse, que soporta todas las fases del desarrollo de software y/o ciclo de vida de una SPL. FeatureIDE es un desarrollo Integrado de desarrollo para AHEAD en todas las fases. Otras implementación que permite es FeatureC++ y FeatureHourse. El uso de aspectos (AspectJ) permite el grado de disponibilidad AJDT como potencial en la industria. FeatureIDE permite crear FM creados con SPLOT.
Otros entorno como JAk, no estaban disponibles para varios lenguajes de programación, por lo que FeatureIDE al aceptar varios lenguajes ya sea en formato java, C++, C, C#, XML e interpreta el formato SXFM.
Después de tantos prototipos con AHEAD y sus limitaciones (formato SXFM), se diseñaron muchas extensiones más de FeatureIDE con representaciones gráficas [AK09].

Otras Herramientas encontradas [Fel13]:

- FaMa está desarrollado por un equipo de la Universidad de Sevilla. Permite el análisis automatizado de los FM integrando algunos de los resolutores más comúnmente propuestos (BDD, SAT y CSP), siendo ésta una feature poco común. Dispone de un plugin de Eclipse gráfico desarrollado en EMF bajo licencia Open-Source.
- MFM es un método desarrollado en el Centro de Investigación en Métodos de Desarrollo de Software (ProS). Allí se ha desarrollado MOSKitt Feature Modeler (MFM), una herramienta Open-Source basada en Eclipse y desarrollada utilizando EMF, GMF y ATL. MOSKitt soporta edición gráfica de modelos, soporte de persistencia, transformación de modelos, traceability y sincronización, documentación y otros.
- S2T2 es el resultado de la investigación de las líneas de productos software por Lero (centro de investigación de Ingeniería del Software de Irlanda). Es una aplicación de Java Open-Source independiente que permite configuraciones y comprobación de restricciones básicas. No dispone de features clonables, referencias, atributos o restricciones avanzadas.

En la Tabla 1 se han analizado las funcionalidades de las herramientas disponibles relacionadas con los FM, resaltando los aspectos más importantes.
De todas las herramientas analizadas excepto una (SPLOT), presentan una facilidad de uso media, baja medida y consisten en plugin para ser utilizados en Eclipse. SPLOT en particular tiene una complejidad más alta porque su objetivo principal es poner en práctica las SPL a través de online enfocada a usos académicos y a maestros del área. Esta herramienta contiene implementados eficientes sistemas de razonamiento y heurísticas de resolución de problemas SAT[3], lo que permite contar configuraciones válidas y calcular el grado de variabilidad de las características de los productos [Urroz12].

[2] http://www.fosd.de/
[3] SAT: Problema de satisfacibilidad booleana. Es un tipo de problema de complejidad Np-completo

Descripción	FeatureIDE[4]	SPLOT[5]	CIDE[6]	XFeature[7]	pure::Variants[8]	FMP[9]	Fuji[10]
Open Source	Si	Si	Si	Si	Copyright	Si	Si
Creador	University of Magdeburg de Alemania	Universidad de Waterloo de Canadá	University of Magdeburg Alemania	Emp. Soft. P&P y Laboratorio ETH-Zurich	pure:systems	Universidad de Waterloo, Canadá	Universidad de Passau, Alemania
Plataforma	Java (plugin eclipse)	Web	Java (plugin eclipse)	Java (plugin eclipse)	Java (plugin eclipse)	Java (plugin eclipse)	JastAdd Extensible Compilador Java
Facilidad de uso	Media	Alta	Media	Baja	Media	Media	Media
Editor FM	Si	Si	Si	Si	-	Si	Si
Análisis Automático FM	Sí	Sí	Si	Si	-	Si	Si
Config. prod. interactivos	Sí	Sí	Si	-	-	-	-
Notación FM	Árbol y Diagrama	Árbol	Árbol	-	-	Si	Si
Integración con código	Sí	-	Si	-	Si	Integrado con RSM y RSA	-
Disponible Online	Si	Sí	Si	-	-	-	Si
Repositorio FM	-	Sí	Si	-	-	-	Si
Configuración del Flujo de Trabajo	-	Si	SI	-	-	-	-
Modelo FM	XML	SXML	Si	XML	-	-	-
Feature Clonables	No	No	No	Si	-	Si	-
FOP	AHEAD FeatureC++ FeatureHourse AspectJ, Antenna FM, Murge	-	AspectJ	-	-	-	AHEAD FeatureHouse
Otros FOPL	C, C++,	-	antlr, C, C++ , C #, ECMAScript, gCIDE, Haskell, JavaCC, y Python	-	-	-	-
Reconoce otras extensiones o herramienta	SPLOT(SXFM)		GUIDSL FeatureIDE	Se centra en modelo y meta-modelo	-	-	-
Utilizado por la Industria	Muy Maduro	Muy Maduro	Prototipo Académico	Prototipo Académico - Etapa de prueba	-	Prototipo Académico - proyecto descontinuado	-
Referencias	[BGF13][PSF+13] [KTG+09] [TM13][TKB+12]	BGF13 PSF+13	[Kas07] [KA11]		[Fel13]	[PSF+13]	

Tabla 1 Funcionalidades de las Herramientas para FM

[4] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/
[5] http://www.splot-research.org/
[6] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/cide/
[7] http://www.pnp-software.com/XFeature/
[8] http://www.pure-systems.com/pure_variants.49.0.html
[9] http://gsd.uwaterloo.ca/fmp
[10] http://www.infosun.fim.uni-passau.de/spl/apel/fuji/

5. APLICACIONES CON FEATURE

En la literatura se pueden encontrar varios casos de estudios o trabajos relacionados que demuestran el método FOSD para el entorno Web o servicios móviles; en la Tabla 2 se listan distintas aplicaciones encontradas.

La naturaleza de los sistemas basados en la web ha sido impactada por una diversidad cada vez mayor de dispositivos, navegadores y plataformas, y el conjunto de diferentes comportamientos de usuarios. El comportamiento dinámico de las features de los sistemas Web debe ser manejado correctamente [MSC+14] [PJ05].

Las aplicaciones móviles difieren de las aplicaciones de escritorio estándar, en el sentido de que estas imponen varios desafíos para el desarrollo de software. Deben ser ejecutadas en dispositivos heterogéneos, con recursos limitados y conexiones transitorias. Los dispositivos móviles, son usados en escenarios complejos como: exploración visual, monitoreo ambiental, manejo de tráfico en vías libres, operaciones militares estratégicas, situaciones de daños y desastres naturales. Este conjunto de desafíos, al cual se le denomina Prism (Programming In the Small and Many), hace referencia al desarrollo de software altamente distribuido, dinámico, móvil, generalmente inalámbrico y con procesamientos heterogéneos sobre un gran número de plataformas restringidas por recursos escasos.

Las aplicaciones móviles sensibles al contexto, en particular, deben adaptarse tanto a las capacidades de acceso en diversos dispositivos como a diferentes contextos, conservando consistencia y utilidad. Se define al contexto como cualquier información que puede ser usada para caracterizar la situación de entidades (usuario, lugar u objeto) que son consideradas relevantes en la interacción entre un usuario y una aplicación, incluyendo al usuario y a la aplicación. La información está clasificada en tres ambientes: a) computacional (procesadores, dispositivos de entrada y salida, capacidad de red, recursos de interacción, conectividad); b) usuario (localización, preferencias/perfil del usuario, situación social) y; c) físico (iluminación, nivel de ruido). Por lo tanto una aplicación adaptable o adaptativa se reconfigura por si misma dinámicamente de acuerdo a su contexto de uso.

Para desarrollar este tipo de aplicaciones es preciso definir los requisitos, en este caso requisitos funcionales; como por ejemplo, la facilidad de compartir datos en un ambiente colaborativo, que implica la disponibilidad de dichos datos en un ambiente con conexiones no siempre disponibles, ni seguras; por lo tanto para garantizar la disponibilidad de los datos y de las conexiones implica considerar la disponibilidad como propiedad de calidad asociada al requisito funcional. Un requisito no funcional es por ejemplo, la minimización del recurso respecto al consumo de la batería, el cual es imperativo en el dominio de las aplicaciones móviles, en particular la propiedad de calidad eficiencia está asociada a este requisito. La movilidad se refiere a tener los datos, los dispositivos y las aplicaciones en cualquier momento y lugar.

El desarrollo de aplicaciones móviles conlleva una variedad de consideraciones de acuerdo al propósito y escenario para el que van a ser utilizadas. Entre estas consideraciones se pueden mencionar: el tipo de aplicación, los sistemas operativos y las plataformas disponibles, las features de los dispositivos (pantalla, memoria, procesador, audio, batería, navegación, impresión entre otros), limitaciones en la conectividad incluso en el lenguaje de los navegadores [Gom11].

Otros aplicaciones con Features son trabajos como [EBA+11] han propuesto modelos orientados a objetivos (MO) para representar a los requisitos y a los objetivos de un sistema de software. Esto es importante en el ciclo de vida del desarrollo de SPL. En la fase de ingeniería de dominio, los MO pueden guiar y justificar la variabilidad en la fase de ingeniería de aplicaciones.

Proyecto	Descripción	Plataforma	Artefacto empleados	Artefacto resultantes	Referencia
Objetos de Aprendizaje MDC	Basado en 42 features, y 3 productos para el caso de estudio	Web	FeatureJS (Javascript y HTML) Anotaciones Composición	FeatureJS (plugin eclipse)	[MSC+14]
Sistemas de intercambio de bicicletas (BSS)	Uso de herramientas para el modelado y análisis de SPL de bicicletas.	Web	FeatureIDE SPLOT ClaferMOOVisualizer FMT (XSFM a XML) Splot2Clafer (XFM a CFR)	CAS: Análisis cuantitativo de BSS visto como sistemas adaptativos colectivos	[BGF13]
Enfoque ligero y reactivo para apoyar una SPL Portal Web.	Desarrollar rápidamente nuevos Portales Web del Portal genérico	Web	XML ASP, HTML Técnica de reutilización XVCL	ST Electronics	[PJ05]
Creación de una SPL- Animación de rostros humanos para lograr refinamiento de las técnicas de simulación.	La base son dos aplicaciones “Face animator” y “Tree grow simulator” Empleó 46 features.	(Una solo pantalla)	SPLOT	Generación de un archivo de configuración y un ejecutable, interfaz gráfica “Meshing Tool Generator”	[Urroz12]
Aprendizaje electrónico de idiomas asistido por móviles (MALL)	Estudio sobre un dominio (ID) particular de la aplicaciones móviles, basado en DSPL.	Móvil	InDOCas	Modelo arquitectural para aplicaciones móviles	[Gom11]
Describir la variabilidad en los FM	Representación de un FM por medio de las redes Petri, se empleó 7 features	Móvil	SPLOT (para FM)	FM2PN: emplea el JLex[11] y el CUP[12], basados en java. PIPEv2[13]	[MDGL13]
Desarrollo de SPL para automatización de hogares	Desarrollar producto a partir de una infraestructura SPL mediante .NET	Escritorio	lenguaje C# CaesarJ DSL VisualStudio2010	Una SPL para Hogar Inteligente	[Per11]

Tabla 2 Aplicaciones con Features

La variabilidad está representada por los FM. Existen relaciones de correspondencia complejas entre los dos modelos, dado que un objetivo puede ser realizado por varias features y a su vez una de las mismas se puede utilizar para la realización de varios objetivos. Ambos modelos proporcionan diferentes perspectivas de variabilidad. Los MO representan la variabilidad intencional del usuario para llegar a sus objetivos. Por otro lado, los FM son comúnmente utilizados para ilustrar la variabilidad entre los distintos sistemas.
En [MDGL13] se propone una herramienta basado en redes Petri para representar los FM, para prevenir errores en la elaboración inicial del FM por medio de reglas de inclusión.

6. FOSD - SPL PARA EL DESARROLLO DE APLICACIONES TVD: ANALISIS PRELIMINAR

En esta sección se introducirán conceptos y características de la Televisión Digital, aplicaciones existentes, lenguaje de programación, entorno y plataformas requeridas para la ejecución, métodos y artefactos empleadas en la actualidad en la República Argentina.

[11] Generador de analizador léxico
[12] Generador de parser
[13] diseño de grafo, para detectar problemas de invariabiliad del FM

6.1 Televisión Digital

La Televisión Digital (TVD) es el conjunto de tecnologías de transmisión y recepción de imágenes a través de señales digitales. En contraste con la televisión analógica la televisión digital codifica sus señales de forma binaria, permitiendo crear vías de retorno entre consumidor y productor de contenidos, haciendo posible la creación de aplicaciones interactivas. Además permite obtener imágenes más claras, mejor sonido y mayor diversidad de canales [Rey11] [BZ13].
Estas señales digitales se envían mediante ondas terrestres hasta llegar a todos los televisores en los hogares, conocida como Televisión Digital Terrestre (TDT) o como Televisión por Aire. La TDT consiste en una antena en forma de parrilla ubicada en los hogares que capta una señal de VHF (Very High Frecuency) que el televisor reproduce.
Principalmente la TDT tiene dos categorías: la televisión estándar SDTV (Standard Definition Television), que tiene una resolución de 576 líneas horizontales y 720 verticales y la televisión de alta definición HDTV (High Definition Television), con una resolución superior que puede llegar hasta las 720 líneas en modo progresivo o 1080 en modo entrelazado y se reducen las posibilidades de interferencia. Con la alta definición HDTV se mejora notablemente la calidad de la imagen [Vaz12].
Hasta hace poco, en la República Argentina, este servicio estaba disponible a través de proveedores privados, por satélites o por cables, actualmente se cuenta con la Televisión Digital Abierta (TDA[14]) que es pública y gratuita.
La TVD Interactiva (TVDi) propone aumentar la experiencia del televidente, permitiéndole acceder a más información, a un diálogo entre disciplinas y multipantallas o realizar acciones que antes no eran posibles, un televidente puede ver un partido de fútbol y al mismo tiempo utilizar el control remoto para revisar las formaciones de cada equipo en cualquier momento del partido [BZ13].
La interactividad es muy sencilla, se requiere un televisor común y un decodificador o un TV con decodificador integrado. El decodificador denominado indistintamente deco, receptor hogareño, sintonizador o set top box, en adelante STB tiene como función permitir que las señales emitidas en la Planta Típica de Transmisión tomadas a modo de radiofrecuencia sean captadas y se conviertan en audio y video para que las señales de TV emitidas puedan ser observadas en cualquier televisor hogareño existente [ARSAT].
El STB es un elemento central para ser utilizado por un televisor analógico y/o digital de un estándar diferente al del sistema de broadcasting o radiodifusión implementado en cada país. Para soportar características de interactividad los STB son dotados de un middleware, como MHP, ARIB, DASE, GEM o Ginga.
Los componentes físicos que constituyen un STB son los siguientes: placa del sistema; sintonizador, modulador/demodulador, demultiplexador, decodificador, procesador gráfico, CPU (Central Processing unit), memoria, disco, interfaces físicas. Para los diferentes estándares de TVD en el mundo (ISDB-T, DBV, ATSC, etc), las arquitecturas de hardware de los STB se diferencian principalmente por el tipo de demodulador y el decodificador MPEG-2/MPEG-4 .
La Figura 7 muestra la arquitectura en capas de un STB genérico. En la capa superior, se localizan los servicios y contenidos que pueden ser producidos en una transmisión de TVD. Ejemplos de estos servicios son las guías de programación electrónica (EPG), juegos on-line, programas interactivos, etc.
En la segunda capa, se ubican las aplicaciones. Estas aplicaciones son responsables por promover el tipo de servicio de la capa superior.

[14] Televisión Digital Abierta http://www.tda.gob.ar/

En la tercera capa, el middleware actúa como interfaz entre el hardware del STB y las aplicaciones. Por lo tanto las aplicaciones pueden ejecutarse de forma transparente sin la preocupación de acceso al hardware de un STB específico. De esta manera el desarrollo y portabilidad de las aplicaciones se vuelve más simple.
En la cuarta capa, se tienen los componentes multimedia de decodificación y codificación, así como los otros módulos multimedia.
En la quinta capa, el sistema operacional, es responsable por el funcionamiento del hardware, la cual provee una capa de abstracción al hardware del STB.
En la última capa, se encuentra el hardware de un STB, que es constituido por una CPU, dispositivos de entrada y salida, almacenamiento, decodificación, sintonización, etc [VV10].

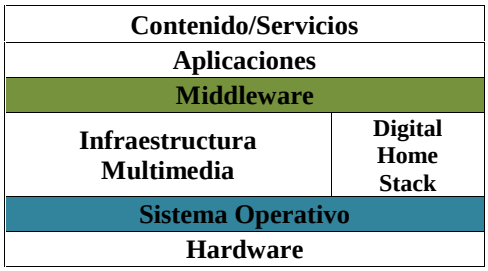

Figura 7. Arquitectura General del hardware de un STB

En la actualidad en Argentina se destacan dos situaciones con el middleware a utilizar, por un lado existen en el mercado STB de distintos fabricantes, y por otro lado estos pueden variar la plataforma de hardware o software. Estas dos situaciones conllevan a que el middleware permita ejecutar aplicaciones sin importar el STB que se disponga.
Para la transmisión de TDT también es necesaria la elección de un sistema o norma como ocurre con las transmisiones analógicas. En el mundo existen tres estándares de TDT cada uno con características diferentes en la codificación de la señal: el estadounidense ATSC, el europeo DVB y el japonés ISDBT. La principal diferencia de la norma brasilera japonesa respecto a sus antecesoras estadounidense y europea es que facilitan el acceso de televisión digital libre y gratuita en los equipos portátiles y móviles, como teléfonos celulares preparados para la recepción de televisión [CNM10].
La Transmisión Digital de Servicios Integrados o Integrated Service Data Broadcasting (ISDB-Tb) consiste en un conjunto de normas japonesas (la b indica que es la Versión Brasileña), donde se definen formas para transmitir contenidos digitales por aire. Este modelo se basa en la existencia de: un conjunto reducido de emisores de contenido (canales de televisión) y un conjunto suficientemente grande de televidentes [Pis10] [Abr13] [CNM10].
Algunas de las principales características ISDB-Tb son: transmisión de un canal HDTV, reservado para transmisiones de TV analógicas; seleccionar la transmisión de 2 y/o 3 canales SDTV (Definición estándar) en lugar de un HDTV; proporcionar servicios interactivos con transmisión de datos, como juegos o compras, vía línea telefónica o Internet de banda ancha; suministrar EPG (Electronic Program Guide, o guía electrónica de programas); puede recibirse con una simple antena interior; proporciona robustez a la interferencia multiruta, causante de los denominados "fantasmas" de la televisión analógica y a la interferencia de canal adyacente de la televisión análoga; permitir mayor inmunidad en la banda UHF (Ultra High Frecuency) a las señales transitorias; entre otros [CNM10].
El Sistema Argentino de Televisión Digital Terrestre (SATVD-T) está basado en la norma ISDB-Tb. En este modelo, los STB no tienen control directo sobre el contenido emitido sino que lo tienen sobre la sintonización de un programa dado. Esto implica que el modelo puede

implementarse sin un canal de retorno que conecte a los televidentes a los emisores u otros servicios.
En la Figura 8 se visualizan los distintos módulos BTS (Broadcast Transport Stream) de SATVD-T: creación del Flujo de Transporte; recepción del Flujo de Transporte y; ejecución de aplicaciones NCL. El Flujo de Transporte o Transport Stream (TS) es una abstracción que encapsula todo aquello que se puede transmitir en una frecuencia UHF. El TS según la norma ISDB permite transmitir varios programas simultáneamente. A su vez, cada programa está dividido en diferentes servicios: vídeo, audio, teletexto, y carrusel de datos, el TS principal (MPEG2 TS), el canal de meta información (ISDB-Tb SI) parámetros de transmisión (Parámetros TMCC). Dentro del TS principal, se encuentran dos programas o subcanales (PMT1y PMT2). En el diagrama, el PMT1 agrupa un vídeo de alta definición (VideoHD), un canal de audio, y el servicio de teletexto (CC).
La recepción del TS es cuando un TS es transmitido en una frecuencia UHF. Los STB sintonizan las frecuencias UHF y procesan estos TS. De tal manera que cuando el televidente cambia de canal, en realidad el STB solo debe mostrar el siguiente sub-canal definido con otro PMT PAT (Program Association Table).
Una aplicación TVD podrá ejecutarse en pantalla completa o con el vídeo principal mientras recibe eventos del control remoto. La ejecución de aplicaciones se basa en la existencia de un carrousel de objetos que contiene las aplicaciones a ejecutar y los archivos de datos o por medio de la señalización de eventos que llegan por el TS; los eventos le indican al STB como deben ejecutar la aplicación TVD [BZ13].

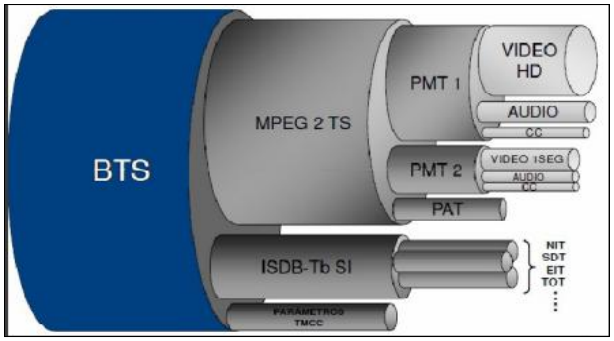

Figura 8. Modelo de SATVD-T

Una aplicación interactiva de TVD es un software de multimedia o programa informático[15] que es ejecutado por el STB, a través del cual el televidente puede interactuar vía control remoto [OHM+13]. Una aplicación se transmite por aire al igual que los programas televisivos y se ejecutan al mismo tiempo que se mira el programa de televisión.
Las aplicaciones interactivas de la TVD se clasifican de la siguiente manera: aplicaciones que no tienen relación con el programa que se está transmitiendo, podría ser el estado del clima en la transmisión de una película; aplicaciones que tengan relación con el programa en que se está transmitiendo, puede ser la formación de los equipos durante la transmisión de un partido de fútbol, recetas en un programa de cocina, biografías en un programa de Historia, entre otros; y por último están las aplicaciones que involucran al usuario, estas podrían ser las aplicaciones de entretenimientos, cuestionarios, entre otros.
Para que los STB puedan ejecutar aplicaciones interactivas de TVD deben tener instalado Ginga. Ginga[16] fue desarrollada por el Telemidia Laboratorio de la PUC de Río de Janeiro (Brasil), es un software open source o abierto y estándar. En primer lugar, que Ginga sea

[15] http://www.polocentroeste.edu.ar
[16] http://tvd.lifia.info.unlp.edu.ar/ginga.ar/

abierto significa que cualquier programador disponga de la información necesaria para utilizarlo sin tener que pagar licencia alguna. Además escribir una aplicación y que se pueda ejecutar en cualquier STB o TV con sintonizador digital es esencial para que todos los usuarios puedan rápidamente utilizar las aplicaciones interactivas siempre y cuando éstos tengan incluidos el software Ginga en el producto [BZ13].
Ginga tiene dos subsistemas, uno se llama Ginga-NCL, que es utilizado para aplicaciones declarativas que soporta los lenguajes NCL (Nexted Context Languaje) y Lua; y por otro lado Ginga-J para las aplicaciones procedurales o imperativas que da soporte a lenguaje Java. En Argentina se implementa el middware Ginga-NCL.
En Argentina el STB del SATVD-T implementa el middleware Ginga ar, el cual realiza todas las operaciones necesarias para que las señales recibidas a modo de radio frecuencia se transformen en video y audio: captación de las señales, decodificación, descompresión, etc.
Ginga.ar es una implementación derivada de PUC Río que presenta un estándar Ginga-NCL, desarrollada por el Laboratorio de Investigación y Formación en Informática Avanzada, Facultad de Informática de la Universidad Nacional de La Plata - LIFIA[17]-UNLP. Ginga.ar es software libre, actualmente licenciado como GPLv2, es la única implementación libre de Ginga con calidad industrial tal que permite que el mismo sea embebido en dispositivos con muy bajos recursos de hardware. Ginga ar fue portado a la arquitectura ST e instalado en STBs comerciales diseñados y producidos en Argentina, las versiones presentada por LIFIA son 1.0.1, 1.1.0, 1.1.2, 1.1.3 y la última versión es ginga ar 2.0. El porcentaje de código C/C++ original utilizado se redujo del 95% en la versión 1.1.0 al 17% en la versión 2.0 de Ginga.ar, esto permite mantener la compatibilidad con el estándar, tener las librerías directamente con el lenguaje NCL y poder integrar fácilmente futuras versiones. [OHM+13] [MOH+12] [Zam12].
Respecto de NCL, el mismo es un lenguaje declarativo basado en el modelo conceptual NCM (Nested Context Model), básicamente es un documento XML. El lenguaje define claramente cómo los objetos media (elementos de contenido multimedial, es decir, los elementos a visualizar como por ejemplo videos, imágenes, sonidos, etc.) son estructurados y relacionados, en tiempo y en espacio. El lenguaje Lua, es un lenguaje de programación imperativa, estructurada y bastante ligero; diseñado como lenguaje de script con una semántica extensible, para ello se requiere NCL. Ginga puede hacer ejecutar aplicaciones interactivas solo en lenguaje NCL o compuestas por NCL y Lua.

6.2 Artefactos y Aplicaciones de TVD interactivas

Se han propuestos distintos artefactos y enfoques para el desarrollo de aplicaciones de TVD, entre ellos

a) Notación UML[18] (Unified Modeling Language) es un lenguaje de modelado de sistemas de software que permite especificar gráficamente métodos o procesos. Específicamente se utiliza para definir, detallar los artefactos, construir y documentar un sistema. Para describir las propiedades estructurales del sistema se utilizan diagramas de casos de uso y para describir el comportamiento del sistema se realiza mediante diagramas de secuencia y diagramas de actividad [Rey11].
   El trabajo [NM13] utiliza UML para comprender el ecosistema donde se desarrolla la aplicación, como así también las interacciones de los usuarios con las aplicaciones. Los diagramas implementados fueron los siguientes:

[17] Lifia TvDigital - http://tvd.lifia.info.unlp.edu.ar/doku.php.
[18] http://www.uml.org/

Diagrama de Componentes: brinda una comprensión general sobre cómo se encuentran interactuando los distintos módulos dentro de la aplicación.
Diagrama de Secuencia o Comunicación: permite entender los flujos de mensajes e interacción entre los usuarios y la aplicación.
Diagrama de Actividad: define cómo es el flujo de ejecución de una funcionalidad.

b) Modelo NCM que es un modelo conceptual centrado en la representación y tratamiento de documentos hipermedia. Un modelo conceptual hipermedia debe representar los conceptos de diseño de datos hipermedia, así como los eventos y las relaciones entre los datos. También debe definir las reglas de estructura y las operaciones en los datos de la manipulación y la actualización de las estructuras. La definición de los documentos hipermedia en NCM se basa en los conceptos de nodos y enlaces. Los nodos son piezas de información y los enlaces se usan para definir las relaciones entre los nodos. Sin embargo, los vínculos no son la única manera de definir las relaciones. El modelo distingue dos clases básicas de nodos, llamados nodos de contenidos y nodos compuestos [Rey11].
   Diagramas NCL, estos diagramas representan gráficamente la visión estructural de un documento NCL. En esencia es un grafo, en el cual los nodos representan medias y las aristas indican los eventos y acciones que los relacionan [OHM+12] [OHM+13].
c) Modelo gráfico de línea de tiempo para la representación de la vida de los elementos de aplicaciones NCL, la interactividad con el televidente y los eventos con sus acciones ocurridos a lo largo de la aplicación [CS11]
d) Herramientas para la codificación Composer y Emulator, creadas en Pontificia Universidad Católica de Río de Janeiro.
   Composer es un entorno de edición dedicado a la creación de programas declarativos en NCL para la TVDi. En esta herramienta, las abstracciones están definidas en los distintos puntos de vista: textual, estructural, layout y temporal. Composer es una aplicación escrita en Java que aún está en desarrollo, y como tal, tiene algunas inestabilidades.
   Emulator es una herramienta indispensable para la visualización de la aplicación NCL en el computador, ya que emula Ginga en un STB. Se utiliza para la etapa de pruebas [Rey11].
e) Métodos ágiles como Scrum, que aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Por ello, está especialmente indicado para desarrollos que tienen la necesidad de obtener resultados rápidos, con requisitos que son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales. En particular, Scrum resultó muy conveniente para las aplicaciones TVD, debido a que se presenta como un Framework que ofrece una estructura que permite crear procesos ligeros particulares para el desarrollo de nuevos productos.
   Entre los artefactos se mencionan la pizarra para el Workflow general y especifico del proyecto, pizarra para Storyboard General y específicos, pizarra para la división de tareas, plantillas de Especificación y Prueba, cuadro de escena, entre otros. En el Workflow General se presentan las tareas generales y secuencia de las mismas, donde se agrega una actividad propia de las aplicaciones TVDI denominada Storyboard, que representa a modo de guión gráfico una visión global del proyecto. Con el resultado del Storyboard se realiza la selección de las escenas [OHM+12] [NM13].

A continuación se mencionaran aplicaciones de TVDi desarrollados en Argentina que emplearon los métodos o artefactos descriptos anteriormente:

1. Aplicaciones desarrolladas por LIFIA: Partidos de Fútbol, Cocineros Argentinos, Cyclope Sokoban, Loading Animation, El Ahorcado, InfoSalud La Plata, Presupuesto Participativo, sin Mosquitos no hay Dengue, entre otros.
2. Prototipos desarrolladas por el Laboratorio de TV DIGITAL[19] de la Universidad Nacional de la Patagonia Austral: Campus y/o Noticiero Universitario, UpaAnimalitos, Unpalbox. Estas aplicaciones fueron modeladas mediante Scrum y los artefactos mencionados [MOH+12].
3. Aplicación desarrollada por ARSAT[20]: Precios Cuidados[21], está disponible en todas las áreas dentro de la cobertura terrestre de la TDA, la aplicación se transmite asociada a la señal televisiva (Canal Encuentro). En la Figura 9 muestra la aplicación donde los usuarios de TDA pueden acceder al listado actualizado de productos incorporados al acuerdo Precios Cuidados, seleccionando, a través del control remoto de su televisor, la provincia, cadena de supermercado y rubro que desean consultar. A través del botón rojo del control remoto, se abre la aplicación, que luego puede navegarse con las flechas arriba y abajo, seleccionando los distintos ítems con la fecha derecha, o con OK.

Figura 9. Aplicación de Precios Cuidados

6.3 Patrones de Diseño de Interacción para la TVDi

Los Patrones de Diseño de Interacción (PDI) para las aplicaciones de TVDi propuestos por [Kur09] se basan en el método de enfoque de grupo; este concepto permite el análisis de tareas y necesidades de los contenidos de los usuarios en aplicaciones de TVDi. Los resultados de los grupos sólo se refieren a tipos específicos de contenido, y se clasifican en tres clases: tareas de usuarios genéricas de TVDi, requisitos de contenido general y en requisitos generales de usabilidad.
Para cada patrón de cada grupo el enfoque de los PDI consiste en: usabilidad; identificación sistemática de los problemas de diseño recurrentes en base al análisis del contexto de uso, tareas y necesidades de los usuarios; integración mediante la vinculación de los patrones de las tareas del usuario genéricos; presentación de alternativas de diseño y discusión de sus ventajas y desventajas específicas; justificación de las soluciones de diseño que presenta resultados de las pruebas de usabilidad empírica y; evaluación iterativa de los expertos de usabilidad de la plataforma o contexto de uso en particular [FP12].
El marco genérico o framework desarrollado para los PDI consistió en los siguientes grupos de patrones:

[19] https://sites.google.com/site/laboratoriodetvdigital
[20] http://www.arsat.com.ar/
[21] www.precioscuidados.com

Grupo A - Diseño de pantalla: en este grupo de patrones se analiza y especifica el grupo de usuarios de la aplicación, junto con los requisitos funcionales y no funcionales de los usuarios, incluyendo sus objetivos y tareas. Según el contenido y funciones se clasifican en:

1. Elegir el Diseño correcto (A1): presenta tres formas de presentar el video y la aplicación: superposición, pantalla completa con video, pantalla completa sin video
2. Superposición (A2): el video se sigue transmitiendo en el fondo de la pantalla, mientras la aplicación se esa ejecutando sobre el video cubriendo una parte pequeña de la pantalla, dejando visible el video. En este patrón son adecuados para la visualización del video fondos semitransparentes, textos pequeños o funciones relacionados con el flujo del programa.
3. Pantalla completa con video (A3): el video ocupa ¼ de tamaño de la pantalla, mientras que la aplicación ocupa el resto. Se recomienda el video en la esquina superior derecha de la pantalla.
4. Pantalla completa sin video (A4): la aplicación cubre toda la pantalla, mientras que el video queda detrás de la aplicación, solo se escucha el audio.

Grupo B - Navegación, a partir del patrón A1 se selecciona la forma de navegar:

1. Múltiples formas de navegar (B1): se presenta los diferentes elementos de la interfaz de usuario para acceder a contenidos y funciones, los patrones son los siguientes: menú, menú, múltiples videos en pantallas, índice, número de página y tabs;
2. Menú (B2): un menú proporciona acceso a diversos contenidos y funciones, un elemento del menú lleva al submenú. Un menú proporciona acceso a los contenidos y funciones organizadas jerárquicamente.
3. Vídeo Multi-Pantalla (B3): una pantalla múltiple ofrece acceso a varios flujos de vídeo presentados simultáneamente. Ellos ayudan a los usuarios obtener una visión general de la secuencia de vídeos disponibles y de sus contenidos actuales, estos podrían ser los juegos olímpicos, torneos de tenis, fútbol o programas que ofrecen múltiples ángulos de cámara.
4. Índice (B4): un índice permite acceder a una visión general organizada alfabéticamente de los elementos de contenido y funciones.
5. Números de página (B5): los números de página proporcionan acceso directo a las páginas individuales. Al igual que en el teletexto analógico, determinados tipos de contenido tienen números de página consistentes a través de aplicaciones.
6. TABS (B6): Tabs facilitar el acceso a los elementos y funciones de contenido, similar al menú.

Grupo C - Teclas de control remoto, a partir del patrón A1 y B1 se elige la configuración de las teclas de uso:

1. Elegir las teclas correctas (C1): el patrón permite definir cuales teclas utilizará la aplicación: teclas de flechas, ok, de colores, numérica o especiales.
2. Teclas de flechas (C2): en un control remoto estándar son cuatro las teclas: arriba, abajo, izquierda y derecha
3. Teclas Ok-Select (C3): es la tecla OK, generalmente se encuentra en el centro de las cuatro teclas de flecha, pero no está etiquetado coherente en todos los controles remoto.
4. Teclas de color (C4): en un control remoto estándar son cuatro las teclas de colores: rojo, verde, amarillo y azul. No solo los colores están estandarizados, sino también el orden que se encuentran alineados en forma horizontal, ya conocido por muchos usuarios de la televisión analógica.
5. Teclas numéricas (C5): en un control remoto estándar son diez teclas: 0-9. En la mayoría de los controles remotos las teclas numéricas se colocan en una cuadrícula de 3 por 3 con el 0 centrada.

6. Teclas especiales (C6): además de teclas estándar algunos controles ofrecen teclas especiales, estos se emplean para algunas funciones en las aplicaciones, pueden ser: “text”, “interactive”.

Grupo D - funciones básicas, a partir de la elección de los patrones A1, B1 y C1 se clasifica en:

1. Llamada inicial a la interacción (D1): presenta formas de acceder a la aplicación. método de acceso, crear conciencia, indicador de navegación en pantalla, número del indicador, posición del indicador en pantalla y duración de indicador en pantalla
2. Inicio (D2): el inicio de la aplicación tiene que ser fácil para que el usuario no abandone la aplicación y esta tenga éxito.
3. Indicador de carga (D3): los usuarios deben ser informados que la aplicación se está cargando para que lo abandone la aplicación, esto es debido al tiempo de los STB.
4. Salida (D4):
5. Ocultar la aplicación (D5)
6. Subir un nivel (D6): siempre debe indicar en qué lugar se encuentra el usuario dentro de la aplicación.

Grupo E - Presentación de contenido, a partir de la elección de los patrones A1, B1 y C1 se clasifica en:

1. Diseño de texto (E1): en este patrón se tiene en cuenta el tipo, tamaño de fuente, color, interlineado, longitud y estructura del texto.
2. Caja de contenido (E2): presentación del tipo de media, diseño de página, transparencia, audio
3. Paginado (E3)
4. Barras de Desplazamiento (E4): para permitir desplazarse por la pantalla en forma vertical u horizontal
5. Switch entre ítems de contenido (E5)
6. Contenido sincronizado (E6)

Grupo F - Participación de usuario: le permite al usuario la votación y pregunta de opción múltiple, asignación de temas y completar texto.

1. Múltiples formas de participación (F1)
2. Votación y selección múltiples (F2)
3. Ubicación de ítems (F3)
4. Completado de texto (F4)
5. Aprobación para conectar (F5)

Grupo G: Entrada de textos

1. Múltiples formas de ingresar textos (G1)
2. Teclado en pantalla (G2)
3. Teclado de dispositivo móvil (G3)

Grupo H: Ayuda

1. Instrucciones en pantalla (H1)
2. Sección de ayuda (H2)

Grupo I: Accesibilidad y personalización

1. Accesibilidad (I1)
2. Personalización (I2)

Grupo J: Grupos de usuarios específicos
1. Niños (J1)

6.4 Motivaciones para el estudio de desarrollo de aplicaciones de TVDi mediante los enfoques SPL y FOSD

A partir de los enfoques estudiados para el desarrollo de aplicaciones de software de TVDi y las características de los STB existen diversas razones que permiten plantear que SPL y FODS son enfoques apropiados para la TVDi:

- En la Sección 6.1 se estudió el concepto y las características del middleware ginga ar; por lo cual las versiones de ginga ar no son compatibles entre sí, ginga ar 1.1 no es compatible con ginga ar 1.2; el resultado podría ocasionar falta de sonido, funcionalidades no esperadas, entre otras. A la vez, la misma versión de ginga ar instalada en distintos STB, dependiendo de su hardware, puede interpretarse de forma diferente. Frente al problema relacionado a la diversidad una SPL permite generar distintas versiones de la aplicación (Sección 2), según la plataforma, entorno, funcionalidad y proceso [BZ13]. El desarrollador mediante una SPL puede seleccionar el componente adecuado para generar el producto de TVDi según el fabricante del STB o el tipo de hardware que posea.

- Los métodos y artefactos presentados en la Sección 6.2 se aplican solo al desarrollo de una aplicación de software de TVD en particular. Si se analiza un conjunto de las aplicaciones mencionadas se verá que existen elementos o comportamientos comunes como, activar o poner en pausa un vídeo, mostrar u ocultar el nombre del programa; y por otro lado las distintas opciones de la aplicación en particular (variabilidad); un STB puede ejecutar la aplicación de manera inmediata; ejecutar la aplicación después de un cierto tiempo asociado con el vídeo principal o esperar que el televidente comience la aplicación [BZ13], entonces se puede plantear el desarrollo de una SPL. Existen elementos que fundamentan el desarrollo de SPL aplicando FM. En la literatura de SPL para el desarrollo de aplicaciones que utilizan MF se han encontrado para plataformas a la web, móvil y de escritorio; no así para aplicaciones de TVD. Es decir es un área de aplicación aun no abordada.

- El marco teórico estudiado en la Sección 6.3 presenta un framework para los PDI, así como un lenguaje de patrones de diseño para aplicaciones de TVDi que consta de 41 patrones. Cada grupo puede verse como una característica y cada patrón como una variante. Además se pueden establecer algunas relaciones de combinación en la aplicación de los patrones, pero no todas las combinaciones son válidas, con lo que se constata interacciones válidas e inválidas, o puede ocurrir que alguna interacción no este contemplada. Estas relaciones podrían ser analizadas mediante interacciones de features, encontramos cierta analogía entre los patrones de [Kun09] y SPL / FODS cuya comprobación permitirá realizar familias de aplicaciones usables, ya que cada patrón garantiza la usabilidad [FP12].
  La guía de patrones presenta una estructura del lenguaje de PDI para la identificación de un problema de TVDi en particular. Si se necesita otro producto similar en diseño y contenido, pero que varían en algunos requisitos; se podría crear un MF para modelar las características comunes y la variabilidad. Una vez desarrollados los componentes que desea el usuario, estos pueden armar sus propios productos de TVDi mediante una SPL.

7. CONCLUSIONES

El análisis realizado de aplicaciones existentes que utilizan modelos de features respecto a la web, móvil y de escritorio; refleja un espacio de estudio abierto en el modelado de aplicaciones para TVDi. Estas aplicaciones son altamente dinámicas con procesamientos heterogéneos sobre un gran número de plataformas que requieren adaptarse a diferentes contextos, conservando consistencia y usabilidad.

Debido a que los artefactos y métodos encontrados son para el desarrollo de una aplicación de TVDi en particular, resulta importante contar con metodologías rápidas que faciliten el desarrollo de software para este dominio. El marco de trabajo propuesto es el Desarrollo de Software Orientado a Features, esté método contempla todas las fases de desarrollo hasta llegar a un producto de software de TVDi, facilita la creación y gestión de familias de sistemas, obteniendo así una Línea de Producto de Software - SPL. Combinar ambas estrategias permitirá reducir los tiempos de desarrollo y alinear de forma correcta los requisitos permitiendo personalización y reutilización.

Por otro lado se estudiaron los Patrones de Diseño de Interacción para la TVDi, estos patrones representan modelos de features para el diseño de una aplicación. Empleando el enfoque SPL y los patrones mencionados se puede generar una gran cantidad de features y combinaciones entre ellas. Una SPL une el concepto de gestión de la variabilidad a través de modelos de features que capturan aspectos fijos y variables que se justificará para este tipo de dominio.

Además hay que tener en cuenta que gestionar los modelos de variabilidad de una SPL en forma manual es difícil, por ello se presentó una serie de herramientas para el modelado de features.

Como trabajo futuro se pretende modelar una SPL para TVDi basado en los Patrones de Diseño de Interacción y, mediante la demostración de varios escenarios que permita validar el marco de trabajo propuesto.

REFERENCIAS

[ABH13] Acher M, Baudry B, Heymans P, Cleve A y Hainaut J L 2013. Support for Reverse Engineering and Maintaining Feature Models. Seventh International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'13).

[Abr13] Abrutsky M 2013. Avances en el estudio de la Televisión Digital como plataforma educativa. III Jornadas de Enseñanza de la Ingeniería (JEIN 2013). Universidad Tecnológica Nacional, Secretaria de Ciencia y Tecnología y Posgrado. Programa de Tecnología Educativa y Enseñanza de la Ingeniería (TEyEI). ISSN 2313-9056, Año 3 Vol.2, pág 117. http://utn.edu.ar/

[AGMG14] Asadi M, Gröner G, Mohabbati B y Gaševi D 2014. Goal-oriented modeling and verification of feature-oriented product lines.

[AHC+13] Acher M, Heymans P, Cleve A, Hainaut J L y Baudry B 2013. Seventh International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'13), http://hal.inria.fr/docs/00/76/67/86/PDF/KSynthesis-VaMoS2013-CR.pdf

[AK09] Apel S y Kästner C 2009. An Overview of Feature-Oriented Software Development. Journal of Object Technology, Vol. 8 Nro. 5, pp. 49-84. http://jot.fm/issues/

[ARSAT] ARSAT 2010. Precios cuidados llego a la TDA con una aplicación de ARSAT. www.arsat.com.ar/

[ASLK10] Apel S, Scholz W, Lengauer C y Kästner C 2010. Detecting Dependences and Interactions in Feature-Oriented Design. 21st International Symposium on Software Reliability Engineering (IEEE).

[Bat09] Batory 2009. On The Importance and Chanllenges of FOSD. ftp://ftp.cs.utexas.edu/.snapshot/nightly.10/pub/predator/BatoryFOSDKeynote2.pdf

[BFGR13] Benavides D, Felfernig A, Galindo J A y Reinfrank F 2013. Automated Analysis in Feature Modelling and Product Configuration.

[BGF13] Beek M H, Gnesi S y Fantechi A 2013. Chaining available tools to support the modelling and analysis of a bike - sharing product line. http://blog.inf.ed.ac.uk/quanticol/files/2013/12/TRbssxvamos14.pdf

[BSR10] Benavides D, Segura Rueda S, y Ruiz Cortés A 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review, ISSN 0306-4379, Vol. 35, Nro. 6, pp. 615-636. http://dialnet.unirioja.es/servlet/articulo?codigo=3232475

[BZ13] Balaguer F y Zambrano A 2013. TV Digital Interactiva Argentina. TV DIGITAL: Un dialogo entre disciplinas y multipantallas, Universidad Nacional de La Plata, ISBN 978-950-34-1048-6, p. 33. http://correo0.perio.unlp.edu.ar/

[CBT+14] Capilla R, Bosch J, Trinidad P, Ruiz Cortés A y Hincheyd M 2014. An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. The Journal of Systems and Software 91, pp. 3–23.

[CNM10] Carboni O, Nuñez J L y Murolo N L 2010. Consideraciones iniciales sobre el surgimiento y la implementación de la TDT en Argentina: El rol del Estado a partir de la TV pública. En Proceedings of the fourth ACORN-REDECOM conference held in Brasilia, DF, Brazil. p. 14-15. www.cprlatam.org/papers/acornredecom2010carboni.pdf

[CS11] Cardozo S, Schwartz S 2011. Crea TV Digital (40JAIIO). ISSN 1850-2946, pp. 440-460.

[Dia10] Diaz O 2010. Línea de Producto de Software. Publicado en “Fábricas de Software: experiencias, tecnologías y organización” 2º edición, M. GPiattiniJGarzás (editores) Editorial Ra-Ma2010.

[DMA05] Diaz P, Montero S y Aedo I 2005. Ingeniería de la web y patrones de diseño. ISBN 84-205-4609-7, Pearson Prentice Hall.

[EBA+11] Ensan A, Bagheri E, Asadi M, Gasevic D, y Biletskiy Y 2011. Goal-oriented test case selection and prioritization for product line feature models. In Information Technology: New Generations (ITNG), Eighth International Conference on IEEE, pp. 291-298.

[Fel13] Felice L 2013. Integración de Técnicas de Análisis de Dominio con Especificaciones RSL.http://sedici.unlp.edu.ar/bitstream/handle/10915/28972/Documento_completo__.pdf?sequence=1

[Gon07] González A 2007. Máquina de Estados con Variablidad. IX Workshop de Investigadores en Ciencias de la Computación (WICC 2007). ISBN: 978-950-763-075-0, pp. 425-429.

[GLS04] Gonzales Baixauli B, Laguna M A y Sampaio do Prado Leite J C 2004. Análisis de Variabilidad con Modelos de Objetivos. http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER04/Bruno_Baixauli.pdf

[Gom11] E.P. Gomez Vargas. Modelo Arquitectual para aplicaciones móviles usando el enfoque de líneas de producción dinámica de software. 2011.

[GPA+07] Garces K, Parra C, Arbolera H, Yie A y Casallas R 2007. Administración de Variabilidad en una línea de producto basada en modelos. Proceedings of the Congreso Colombiano de Computación, Bogotá, Colombia.

[KA11] Kästner C y Apel S 2011. Feature-Oriented Software Development: A Short Tutorial on Feature-Oriented Programming, Virtual Separation of Concerns, and Variability-Aware

Analysis. In GTTSE Summer School: Generative & Transformational Techniques in Software Engineering, volume 7680 of Lecture Notes in Computer Science, Berlin/Heidelberg: Springer-Verlag, pp. 346-382. http://www.cs.cmu.edu/

[Kas07] Kästner C 2007. CIDE: Decomposing Legacy Applications into Features.

[KPF11] Kavand M, Paarsa S y Faraahi A 2011. ciFeature: A Context-Independent Feature-Oriented Software Development Approach.

[KTG+09] Kästner C, Thüm T, Gnesi S, Feigenspan J, Leich T, Wielgorz F y Apel S 2009. FeatureIDE: A tool framework for feature-oriented software development. Proceedings of the 31st International Conference on Software Engineering (ICSE '09). ISBN: 978-1-4244-3453-4, pp. 611-614. http://www.infosun.fim.uniassau.de/cl/publications/

[Kun09] Kunert T 2009. User-Centered Interaction Design Patterns for Interactive Digital Televisión Applications. ISSN 1571-5035.

[LBL06] Liu J, Batory D, y Lengauer C 2006. Feature oriented refactoring of legacy applications. In Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), pp. 112-121.

[MDGL13] Martínez C, Díaz N, Gonnet S y Leone H 2013. Representación de la Variabilidad en Líneas de Productos de Software empleando Redes de Petri. 14th Argentine Symposium on Software Engineering (42 JAIIO - ASSE 2013). ISSN 1850-2792, pp. 127-141. http://42jaiio.sadio.org.ar/proceedings/simposios/Trabajos/ASSE/10.pdf

[MFP08] Mellado D, Fernández Medina E y Piattini M 2008. Aplicando un Proceso de Ingeniería de Requisitos de Seguridad de Dominio para Líneas de Producto Software. IEEE Latin America Transactions, Vol. 6, Nro. 3.

[MOH+12] Miranda M, Oyarzo F, Herrera F y Casas S 2012. Enfoques y Herramientas de Desarrollo para Aplicaciones de TVDi. 2° Encuentro de Investigadores de Patagonia Austral, San Julián, Argentina, ISBN 978-987-1242-66-5. http://sedici.unlp.edu.ar

[MP14] Metzger A y Pohl K 2014. Software product line engineering and variability management: achievements and challenges. In Proceedings of the on Future of Software Engineering, pp. 70-84. http://dl.acm.org/citation.cfm?id=2593888.

[MSC+14] Machado I D C, Santos A R, Cavalcanti Y C y otros 2014. Low-level variability support for web-based software product lines. Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems (VamoS ’14). http://dl.acm.org/citation.cfm?id=2556637

[NJ13] Navarro Favela J G, Juárez Martínez U 2013. Cómo desarrollar una línea de productos de software, un enfoque práctico. 4TH International Conference on Computer Science and Its Application (CIIA 2013). ISBN 978-607-9119-02-7, pp. 16-25. http://ciia.itsm.edu.mx

[NM13] Navarro N y Medel R 2013. Metodología para el Desarrollo de Aplicaciones en un Ambiente de Televisión Digital. http://unsl.edu.ar

[OHM+12] Oyarzo F, Herrera F, Miranda M y Casas S 2012. Scrum para el desarrollo de aplicaciones TVDi. XVIII Congreso Argentino de Ciencia de la Computación (IV WISS - CACIC 2012), Bahía Blanca, ISBN 978-987-1648-34-4. http://cs.uns.edu.ar/cacic2012/

[OHM+13] Oyarzo F, Herrera F, Miranda M y Casas S 2013. Experiencias y Prototipos de Aplicaciones de TV Digital Interactivas. ISSN 1852-4516. http://ict.unpa.edu.ar

[OHC14] Oyarzo F, Herrera F y Casas S 2014. API TVD, un Generador de Aplicaciones Interactivas para TV Digital. Simposio Latinoamericano de Ingeniería de Software (CLEI 2014), Montevideo, Uruguay. ISBN 978-1-4799-6129-0, pp. 298-305.

[Pen09] Peng X 2009. Analyzing evolution of variability in a software product line: From contexts and requirements to features. JComputSci& Technol 09, Vol. 24, Nro. 2, pp. 319-338.

[Per11] Pérez A 2011. Desarrollo de una Línea de Productos Software para Automatización de Hogares usando Clases Parciales de C#.

http://www.alumnos.unican.es/apr85/publications/pfc_AlejandroPerez.pdf

[Pis10] Pisciotta N O 2010. Sistema ISDB-Tb (Primera Parte). Publicaciones de la Universidad Blas Pascal. Serie Materiales de Investigación, Vol. 3 Nro. 9. ISBN 978-987-1954-08-7. http://ubp.edu.ar

[PJ05] Pettersson U y Jarzabek S 2005. Industrial experience with building a web portal product line using a lightweight, reactive approach. SIGSOFT Software Engineering Notes, Vol. 30, Nro. 5, pp. 326-335.

[PSF+13] Pereira J A, Souza C, Figueiredo E, Abílio R, Vale G, y Costa H A X 2013. Software Variability Management: An Exploratory Study with Two Feature Modeling Tools. In Proceedings of the 7th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS'13), pp. 36–45.

[RGC07] Rodríguez J 2007. Línea de Productos de Softwares. www.fing.edu.uy/inco/

[RGMS13] Rincón L, Giraldo G, Mazo R, y Salinesi C 2013. An Ontological Rule-Based Approach for Analyzing Dead and False Optional Features in Feature Models. http://hal.archives-ouvertes.fr/

[RS08] Ross Frantz F y Segura S 2008. Automated Analysis of Orthogonal Variability Models. A First Step (Análisis automático de líneas de producto software usando distintos modelos de variabilidad). http://www.lsi.us.es/docs/doctorado/memorias/

[SGB01] Svahnberg M, Gurp J y Bosch J 2001. On the Notion of Variability in Software Product Lines. https://www.netlearning2002.org/fou/

[Som05] Sommerville I 2005. Ingeniería del Software. ISBN 84-7829-074-5.

[TC10] Trejo N, Casas S 2010. Enfoques Emergentes de Ingeniería de Software Aplicados a Grid Computing. http://sedici.unlp.edu.ar/

[TC11] Trejo N, Casas S 2011. Separación Avanzada de Concerns para desarrollar aplicaciones Grid. ISSN 1852-4516. http://ict.unpa.edu.ar/files/ICT-UNPA-26-2011.pdf

[Thu08] Thüm T 2008. http://wwwiti.cs.uni-magdeburg.de/

[Thu13] Thüm T 2013. FeatureIDE- Get Started. http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/slides/featureide-2-getstarted.pdf

[TKB+12] Thüm T, Kästner C, Benduhn F, Meinicke J, Saake G y Leich T 2012. FeatureIDE: An extensible framework for feature-oriented software development. Science of Computer Programming, Vol. 79, Nro. 0, pp. 70-85, www.sciencedirect.com/

[TM13] Thüm T, Meinicke J 2013. FeatureIde: Background. http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/slides/featureide-0-background.pdf

[TM13] Thüm T, Meinicke J 2013. FeatureIde: Overview. http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/slides/featureide-1-overview.pdf

[TM13] Thüm T y Meinicke J 2013. FeatureIde: Development. http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/slides/featureide-3-development.pdf

[Urroz12] Urroz Urzúa G I 2012. Adaptación de software de aplicación al paradigma de la Ingeniería de Línea de Productos de Software. http://tesis.uchile.cl/tesis/

[Vaz12] Vazza F 2012. La televisión, del blanco y negro al digital. Cuaderno de cátedra del Taller de Producción Audiovisual I, Facultad de Periodismo y Comunicación Social. UNLP. Ediciones EPC. http://unlp.edu.ar

[VV10] Villanueva J M y Velazquez Díaz C 2010. Informe Preliminar: Estado del Arte de Receptores Set-Top-Box – Aplicaciones. Área de aplicaciones telemáticas INICTEL-UNI, Lima, 2010. http://aat.inictel-uni.edu.pe/

[Zam12] Zambrano A 2012. Introducción a la TV Digital Interactiva y Ginga. ar. Universidad Nacional de La Plata. La Plata–Argentina. http://tvd.lifia.info.unlp.edu.ar/ginga.ar/