ISSN 1870-4069

Generación automatizada de aplicaciones inmóticas bajo el enfoque de LPS

Jesús-Moisés Hernández-López, Ulises Juárez-Martínez

Instituto Tecnológico de Orizaba,
Veracruz, México

yeste_dracker@hotmail.com, ujuarez@ito-depi.edu.mx

Resumen. Una línea de productos de software (LPS) se refiere a los métodos de ingeniería de software, herramientas y técnicas que se utilizan para crear un conjunto de sistemas de software que comparten características similares. En este trabajo se desarrolló una LPS orientada hacia el campo de la inmótica, en la que se combinaron distintas tecnologías para automatizar la generación de aplicaciones, que incluyen: aplicación ejecutable y documentación. Algunas de las tecnologías que se integraron son: MDA (*Model-Driven Architecture*), Scala y AspectJ. El proceso de generación de las aplicaciones se realiza con base en un configurador.

Palabras clave: Variabilidad, *MDA*, inmótica, rasgos, aspectos.

Automated Generation of Inmotic Application with SPL Approach

Abstract: A software product line (SPL) refers to the methods of software engineering, tools and techniques used to create a set of software systems with similar characteristics. In this work was developed a SPL oriented to inmotic domain, with different technologies combined to automate the application generation, including: executable application and documentation. Some technologies that were integrated are: MDA (Model-Driven Architecture), Scala and AspectJ. The application generation process is based on a configurator.

Keywords: Variability, *MDA*, inmotic, traits, aspects.

1. Introducción

Hoy en día la competitividad entre las empresas que se dedican al desarrollo de software va en aumento, especialmente empresas que tienen como objetivo satisfacer las necesidades de un segmento de mercado. Estas empresas buscan cumplir con tres aspectos importantes que la ingeniería de software aun no logra del todo, como son: calidad, tiempo y costo. Por este motivo diversas empresas adquieren este paradigma como base de producción. Uno de los dominios que en la actualidad tienen una alta

demanda es el campo de la inmótica. Las empresas que se especializan en este campo requieren la implementación de un enfoque que permita garantizar la calidad de los productos y optimizar los beneficios que como empresa se requieren. Este trabajo integra distintas tecnologías para el desarrollo de una LPS, con el objetivo de automatizar la generación de aplicaciones y documentación específica para cada una de ellas. Las tecnologías que se consideran son: AspectJ, la Arquitectura Dirigida por Modelos (*Model-Driven Architecture*) y el lenguaje Scala, principalmente con el uso de *traits*.

La estructura general del artículo es la siguiente: en sección 2 se observa un resumen de los trabajos relacionados. En sección 3 se explica el enfoque propuesto. En sección 4 se detalla el caso de estudio en que se aplicó la propuesta. En sección 5 se muestran las conclusiones y trabajo a futuro. Finalmente las referencias.

2. Trabajos relacionados

En [1] se presentó la variabilidad mediante la integración del Desarrollo Dirigido por Modelos (*MDSD- Model Driven Software Development*) y Desarrollo de Software Orientado a Aspectos (*AOSD- Aspect-Oriented Software Development*). La utilización *MDSD-LPS* facilita la trazabilidad desde el dominio del problema hasta el dominio de la solución. El lenguaje orientado a aspectos es útil en la generación de código donde la arquitectura no proporciona enlaces. En [2] se propuso el método de Análisis Orientado a Aspectos (*AOA*) de los requisitos de los productos para el diseño de la Arquitectura de Línea de Productos (*PLA- Product Line Architecture*). El esquema de pasos del (*AOA*) es:(1) se separan los requisitos en cada aspecto de los requisitos originales, (2) se analizan los requisitos de cada aspecto por separado y se examina de forma aproximada la arquitectura requerida para cada aspecto, (3) se analizan los resultados y las opciones de diseño.

En [3] se presentó un enfoque que facilita la implementación, gestión y trazabilidad de la variabilidad mediante MDE (*Model-Driven Engineering*) y el Desarrollo de Software Orientado a Aspectos. Las características se separan en modelos y se componen por técnicas *AO* a nivel de modelo. Al integrar *MDSD* en SPLE (*Software Product Line Engineering*) los DSL (*Domain Specific Language*) manejan la variabilidad con respecto a su estructura o comportamiento. En [4] se presentó un nuevo enfoque para implementar LPS por mecanismos de reutilización de grano fino. Se introdujo el FRTJ (*Featherweight Record-Trait*), donde las unidades de funcionalidad de productos se modelan por los *traits* y los *records*. El grado de reutilización *traits-records* es más alto que el potencial de reutilización de las jerarquías que se basan en clases estáticas estándar.

En [5] se presentó una Arquitectura de Línea de Productos (*PLA- Product Line Architecture*). Esta investigación tuvo como objetivo obtener una *PLA* que: (1) cumpla con los atributos de calidad de los productos que se especifican, (2) que sea lo suficientemente genérico para generar los productos, (3) apoye las similitudes y variabilidad, (4) reúna los atributos de calidad para la LPS específica. En [6] se propuso un enfoque que facilita la gestión de la variabilidad en el modelado de la

arquitectura para la implementación de una LPS, donde los requerimientos del dominio, así como la arquitectura de la LPS se capturan en modelos. En la ingeniería de aplicación, el DSL se utilizó para especificar los requisitos de las aplicaciones concretas. Las técnicas *AO* se utilizaron durante la ingeniería de domino para modular los asuntos de interés en los modelos, transformadores y generadores.

En [7] se realizó un análisis que sirvió para identificar las debilidades de los Enfoques Orientados a Características (*FOA- Feature-Oriented Approach*), haciendo hincapié en la modularidad de la transversalidad. Se demostró que con la falta del apoyo de la modularidad transversal en los enfoques *FOA*, conduce a código disperso. Por otra parte, también se señaló que los *pointcut* como mecanismo para expresar la transversalidad en idiomas como AspectJ no es suficiente, ya que carece de soporte de módulos multiabstracción.

En [8] se habló sobre el montaje automatizado y la personalización de los componentes específicos del dominio de una *PLA*. Los niveles de abstracción en que se desarrolla la *PLA* son de bajo nivel. Para abordar este problema se utilizó la Ingeniería Dirigida por Modelos (*MDE*). Los beneficios obtenidos con *MDE* en LPS son: (1) agiliza el desarrollo de la *PLA* con la integración de las herramientas del modelado y la arquitectura de componentes de dominio específico, (2) las estructuras basadas en modelos ayudan a mantener la estabilidad de la evolución del dominio de los sistemas basados en *MDE*, (3) para mejorar la robustez y habilidad de la transformación de modelos se necesitan pruebas y soporte de depuración para ayudar a corregir y encontrar los errores en las especificaciones de transformación.

3. Propuesta

Para la construcción de la línea de productos se utilizó el *framework* de desarrollo para LPS [9]. Este *framework* distingue 2 procesos. El proceso de ingeniería de dominio se encarga de crear la plataforma con todos los artefactos que definen la parte común y la variabilidad de la LPS. El proceso de ingeniería de aplicación es el responsable de explotar la variabilidad que se definió y reutilizar los artefactos de la plataforma para obtener una aplicación específica.

La Arquitectura Dirigida por Modelos (*MDA-Model Driven Architecture*) se utilizó para definir una arquitectura que integra todos los elementos que trabajan en conjunto para el funcionamiento de una LPS. Esta integración no involucra o modifica los procesos que define el *framework* de desarrollo [9]. *MDA* permite la evolución de la LPS desde diferentes perspectivas, por lo que se diseñaron modelos en los que se seleccionaron 2 niveles de abstracción del enfoque, estos son: el Modelo Independiente de la Plataforma (*PIM-Platform-Independent Model*) y el Modelo

Específico de la Plataforma (*PSM-Platform-Specific Model*). En el modelo independiente se plantean conceptos a un alto nivel de abstracción, sin especificar las tecnologías que se van a utilizar. En el modelo de lado izquierdo de la figura 1, se observa el *PIM* con los siguientes elementos:

Dominio.- Corresponde al segmento de mercado para el que se desarrolla la LPS. En este caso, el *PIM* no determina hacia que segmento de mercado se orienta la LPS.

Modelo de características.- Para el desarrollo de la LPS existen diferentes lenguajes para modelar características. Esta definición permite a la LPS no depender de ningún lenguaje en específico.

Selección de características.- Este concepto se refiere a obtener una instancia del modelo de características que se utiliza, el cual representa las características de una aplicación.

Plataforma.- Contiene los artefactos que se obtuvieron al aplicar el *frame-work* de desarrollo [9]. Entre estos están los artefactos de requerimientos, diseño, implementación y pruebas, los cuales se diferencian entre artefactos del dominio y de aplicación, además se tienen los archivos que se encargan del ensamblaje de componentes de software.

Configurador.- Se encarga de utilizar de forma automática los artefactos de la plataforma, para verificar el dominio de la LPS y(o) generar aplicaciones.

Artefactos de dominio o aplicación.- Son los artefactos que se generan en función de los procesos que realiza el configurador.

A partir del *PIM* es posible obtener uno o más *PSM* que muestran una proyección del *PIM*. Para generar cada *PSM* se toman como base las reglas de transformación que se establecieron en el *PIM*. El *PSM* que se obtiene puntualiza las tecnologías que se utilizan para el desarrollo de una LPS. En el caso de estudio que se describe en la sección 4 se aplicó el *PSM* que se visualiza de lado derecho en la figura 1.

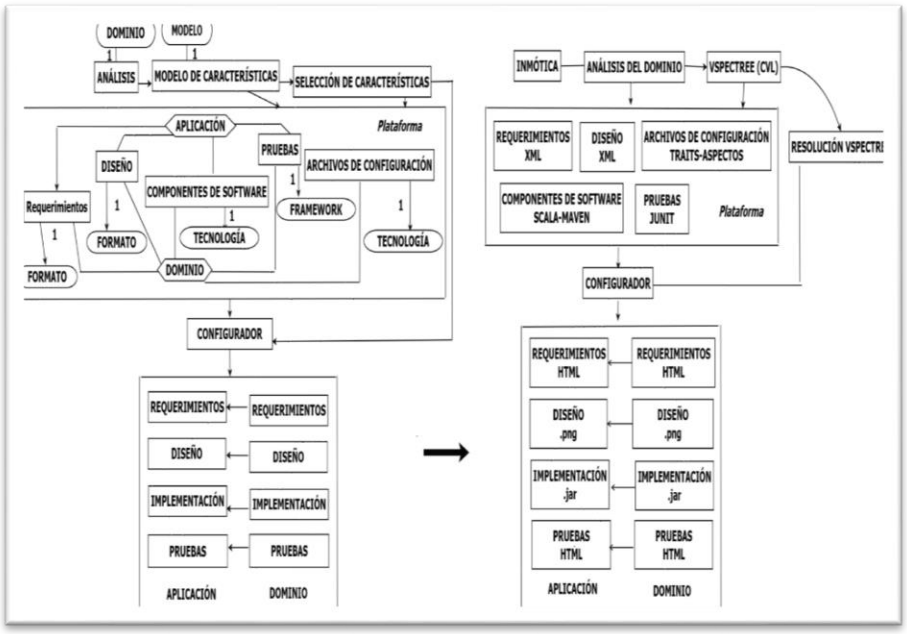

Fig. 1. *PIM* ***a*** *PSM.*

4. Caso de estudio

La inmótica es la incorporación de numerosos subsistemas en las instalaciones de uso terciario o industrial, con el fin de optimizar recursos, reducir costos y disminuir el consumo de energía innecesario, al mismo tiempo que aumenta la seguridad y el confort.

Los requerimientos que se obtuvieron para la LPS son de un caso real de una empresa privada. La administración de los productos de la LPS se realizó mediante un portafolio [9] que contiene la definición de 8 productos que la LPS es capaz de generar

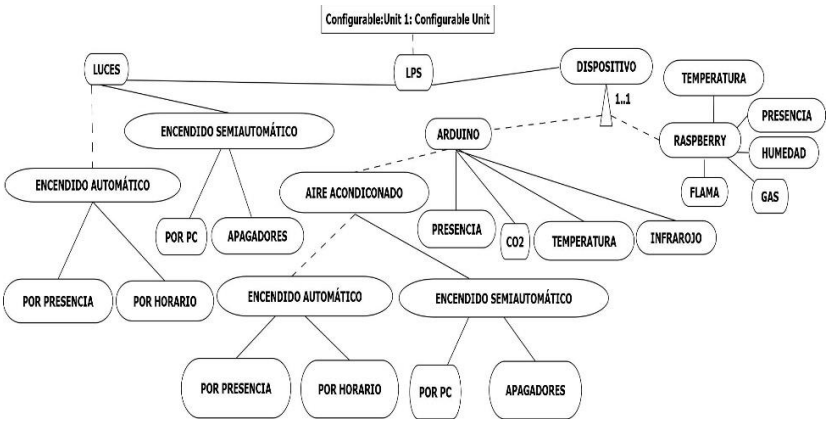

Fig. 2. Árbol de especificación de variabilidad.

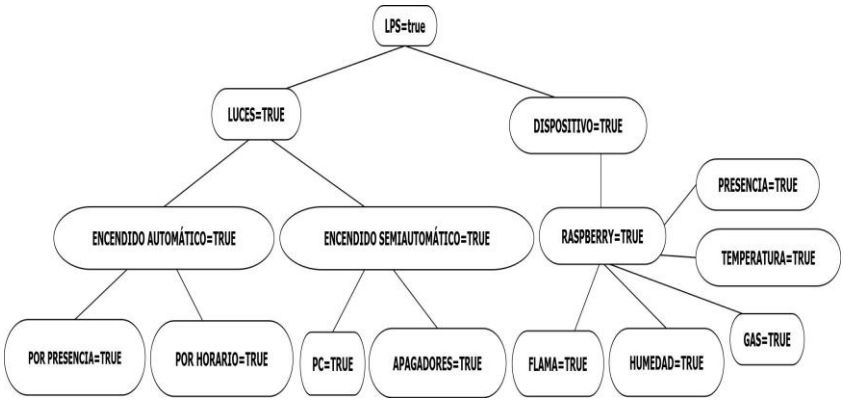

Fig. 3. Resolución de árbol de especificación de variabilidad para aplicación 7.

La variabilidad de la LPS se representó mediante un Árbol de Especificación de Variabilidad (*VSpectree*) con el Lenguaje Común de Variabilidad (*CVL-Common*

Variability Language) [10]. En la figura 2 se observa el *VSpectree* con las características de la LPS. Para representar las características de una aplicación con el *CVL*, es posible obtener instancias validas del árbol de figura 2. Al nuevo árbol que se obtiene se denomina Resolución de Árbol de Especificación de Variabilidad (Resolución *VSpectree*). A el árbol de la figura 2 que representa las características de una aplicación de la LPS, y se define con base en las restricciones del *CVL* [10]. En el árbol de figura 3 se observa la Resolución *VSpectree* para la aplicación o producto 7.

4.1. Configurador

El objetivo del configurador es automatizar los procesos de generación y verificación de artefactos de dominio y aplicación, con el fin de obtener aplicaciones completas que incluyan: documentación y aplicación ejecutable (.jar). El funcionamiento del configurador se basa principalmente en un archivo de configuración con formato XML y la selección de características válidas para una aplicación, como la resolución que se visualiza en la figura 3. La selección de características se realiza a través de la interfaz que se observa en la figura 4.

El archivo de configuración contiene la ruta donde se empaquetan las aplicaciones que se generan, además de 3 secciones que son: características, productos y recursos. En la sección de características se definen todas las características que administra la LPS, estas características son las que se encuentran en el árbol de especificación de variabilidad que se observa en la figura 2. La sección de productos describe todos los productos de la LPS con las características que conforman cada uno de ellos.

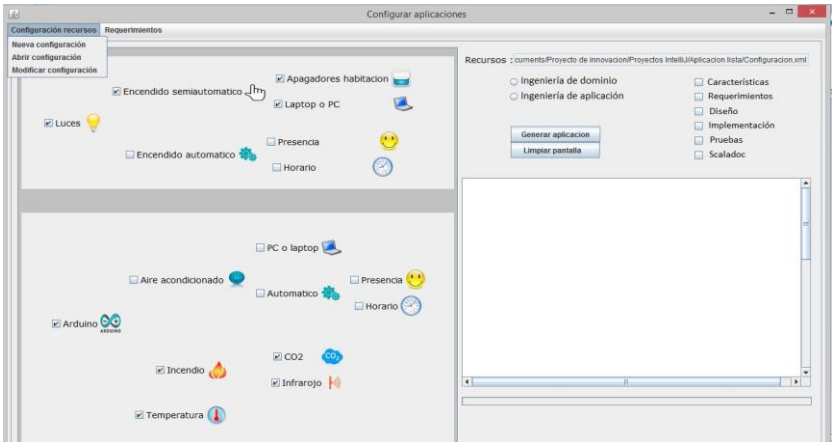

Fig. 4. Interfaz principal del configurador.

A continuación se muestra un ejemplo de la estructura del archivo de configuración en formato XML.

<lps nombre="LPS" ubicacion="C: /Aplicaciones/"> <caracteristicas>

```
    <caracteristica nombre="Luces por presencia"/> </caracteristicas>
  <productos>
    <producto nombre="Producto 1">
      <caracteristica nombre="Luces por presencia"/> </producto>
  </productos>
  <recursos>
     <recurso clave="1" tipo="Componentes_xml" url="C:/"/>
  </recursos>
</lps>
```

En el apartado que corresponde a recursos, se coloca la ubicación de los diferentes artefactos que se definieron con base en los procesos de ingeniería de dominio e ingeniería de aplicación del *framework* de desarrollo [9]. Una vez que se define el archivo de configuración es posible generar de forma automática las aplicaciones que se deseen.

4.2. Artefactos de la LPS

Para la definición de requerimientos textuales se diseñó un formato en XML donde se distinguen requerimientos de dominio y de aplicación, lo que permite al configurador (sección 4.1) obtener los requerimientos de forma automática.

A continuación se muestra un fragmento de la estructura del archivo XML para la definición de requerimientos.

```
<requerimientos>
<requerimiento clave="1" condicion="Base">R1</requerimiento>
<requerimiento clave="2" condicion="Flama">R3</requerimiento>
```

Un requerimiento que tiene el valor “Base” en el atributo condición, indica que el requerimiento pertenece al dominio, en caso de tener el nombre de alguna de las características que se describen en el diagrama de la figura 2, pertenecerá a la aplicación.

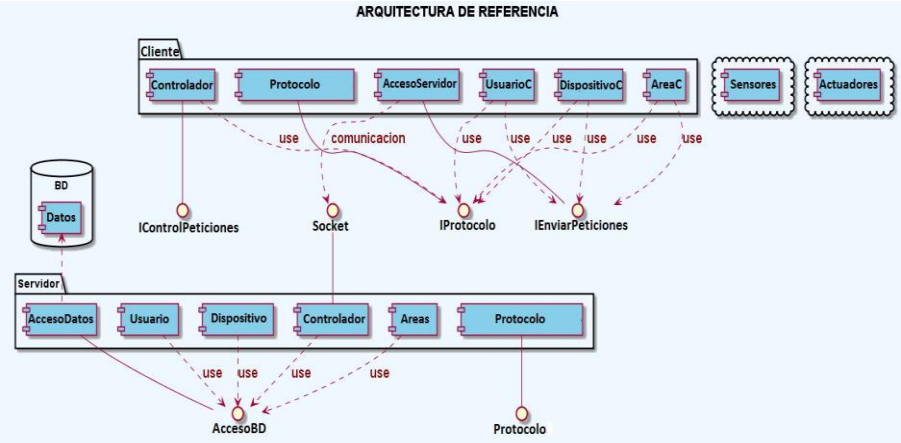

Fig. 5. Arquitectura de referencia.

En la fase de diseño de dominio y diseño de aplicación [9], se obtuvieron diferentes artefactos, entre ellos se encuentra la arquitectura de referencia que se reutiliza para cada una de las aplicaciones de la LPS, y la arquitectura de aplicación que representa la funcionalidad completa de una aplicación específica. En las figuras 5 y 6 se observa la arquitectura de referencia y únicamente la arquitectura del servidor de una de las aplicaciones de la LPS, donde se aprecia la reutilización de la arquitectura de referencia y el ensamblaje de los componentes del servidor necesarios para configurar la aplicación 7

Para automatizar la generación de los artefactos de diseño se definió un archivo XML que representa los componentes del dominio y de la aplicación. Este archivo facilita al configurador la selección de los componentes para generar el diagrama de la arquitectura de referencia y/o de la aplicación.

Los componentes que tienen el valor "Base" en el atributo condición, indican que el componente pertenece al dominio, en caso de tener el nombre de una característica, pertenece a la aplicación. La sintaxis que se utilizó se basa en la herramienta PlantUML [11]. A continuación se muestra un fragmento de la estructura del formato XML para la definición de componentes.

```
<componentes>
  <componente clave="1" condicion="Base"> [C1] </componente> <componente clave="2"
  condicion="Luces"> [C2] </componente>
```

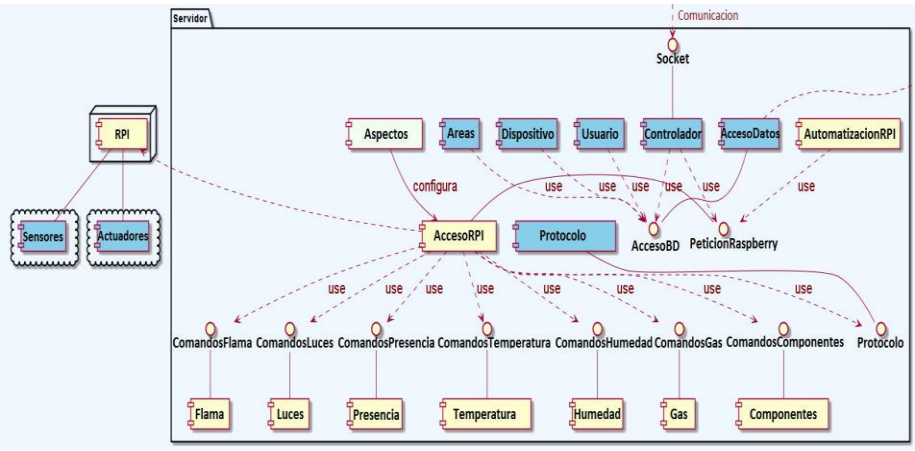

Fig. 6. Arquitectura del servidor para aplicación 7.

Durante la fase de realización, el configurador genera de forma automática la aplicación (ejecutable.jar) con la herramienta *Apache Maven* [12]. La aplicación se configura con base en la arquitectura de aplicación que se obtiene en la fase de diseño de la aplicación. El ensamblaje de las diferentes variantes para cada aplicación se realiza con composición *mixin* mediante el uso de *traits* del lenguaje Scala.

Cada *trait* que se utiliza contiene la funcionalidad para las principales características que se definieron en el *VSpectree* (Figura 2), de tal forma que la

configuración de características para cada aplicación basta con agregar o quitar *traits* en la composición *mixin*. El elemento que hereda el comportamiento de los *traits* es otro *trait* (CaracteristicasAplicacion). Además de la configuración con composición *mixin*, también se utilizaron aspectos para conectar componentes a nivel binario.

En la figura 7 se observa una parte de un diagrama de clases de una aplicación con la combinación de aspectos y *traits*. La fase de pruebas requiere de la entrega de los artefactos que proporciona la fase de realización de la aplicación, que corresponde a los componentes de software con la configuración de una aplicación específica.

Para la reutilización y manejo de la variabilidad de las pruebas se definieron *suites* de pruebas con el *framework JUnit*. Estas *suites* se ejecutan de forma automática para verificar únicamente el dominio de la LPS o la aplicación completa que se va a generar.

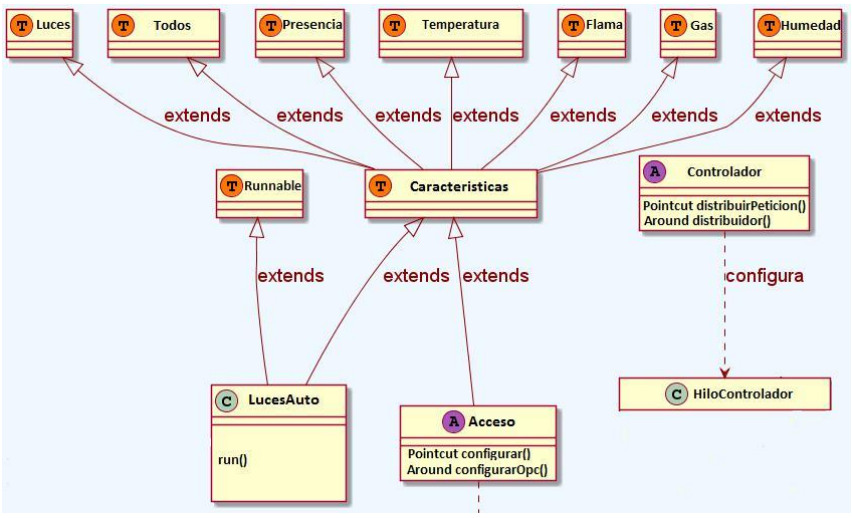

Fig. 7. Combinación de aspectos y *traits.*

Cada una de las clases que se especifican en la suite contiene las diferentes pruebas para el dominio y las aplicaciones. A continuación se muestra un ejemplo de Suite de pruebas para la aplicación 7.

```
@RunWith (classOf [Categories])
@IncludeCategory (classOf[Producto7])
@SuiteClasses(Array(classOf[Luces],classOf[Automatizacion)) class
SuiteProducto7Test {}
```

En la implementación para cada prueba de la plataforma se determina si pertenece al dominio, para una aplicación específica o para ambos. De esta forma se maneja la variabilidad de forma automática para evitar aplicar pruebas que no corresponden a

la aplicación que se desea generar. Si la prueba no devuelve el valor que se establece, se marca como error y se registra automáticamente en un reporte de pruebas. Se tienen 75 pruebas en la LPS.

A continuación se muestra un ejemplo de la estructura de una prueba para las aplicaciones 7 y 8.

```
@Category(Array(classOf[Producto7],classOf[Producto8]))
@Test
def pruebaDeIntegracionEstadoDeSensorPresenciaRaspberry={}
```

5. Conclusiones y trabajo a futuro

La automatización de los procesos de la LPS disminuyó el tiempo de generación de aplicaciones y mejoró la calidad por la constante revisión de los componentes de software que se realizan en una LPS, además de aplicar el esquema de pruebas que maneja el *framework* de desarrollo. El uso de *MDA* permitió separar las responsabilidades de los diferentes elementos que integran la LPS en un *PIM* y *PSM*, lo que facilita la evolución de la LPS en ambos modelos. Con la combinación de Scala y AspectJ se obtuvieron beneficios para el manejo de la variabilidad en la implementación de la LPS, con el uso de *traits* mediante composición *mixin* y con aspectos para el ensamblaje de componentes de software a nivel binario. El desarrollo del configurador permitió automatizar la generación de aplicaciones en las fases de requerimientos, diseño, implementación y pruebas. Como trabajo a futuro se pretende incrementar el número de aplicaciones del portafolio de productos de la LPS, agregar más tipos de diagramas UML, y aumentar la capacidad del configurador para crear más de una aplicación de forma concurrente.

Referencias

1. Voelter, M., Groher, I.: Product line implementation using aspect-oriented and model-driven software development. In: Software Product Line Conference, SPLC 2007, 11th International, IEEE, pp. 233–242 (2007)
2. Kishi, T., Noda, N.: Aspect-oriented analysis for product line architecture. In: Software Product Lines, Springer, pp. 135–145 (2000)
3. Groher, I., Voelter, M.: Aspect-oriented model-driven software product line engineering. Transactions on aspect-oriented software development VI, Springer, pp. 111–152 (2009)
4. Bettini, L., Damiani, F., Schaefer, I.: Implementing software product lines using traits. In: Proceedings of the 2010 ACM Symposium on Applied Computing, ACM, pp. 2096–2102 (2010)
5. Verdín, M. K., Olalde, C. L., Van Der Hoek, A., Peña, J. F.: Diseño de Arquitecturas de Líneas de Productos de Software empleando AOPLA. In: VIII CIINDET, Cuernavaca, Morelos, México (2009)
6. Groher, I., Voelter, M., Schwanninger, C.: Integrating Models and Aspects into Product Line Engineering. In: SPLC, pp. 355 (2008)

7. Mezini, M., Ostermann, K.: Variability management with feature-oriented programming and aspects. In: ACM SIGSOFT Software Engineering Notes, ACM, Vol.29, pp. 127–136 (2004)
8. Deng, G., Schmidt, D. C., Gokhale, A., Gray, J., Lin, Y., Lenz, G.: Supporting Evolution in Model-Driven Software Product-line Architectures. In: ACM SIGSOFT Software Engineering Notes, ACM (2007)
9. Van Der Linden, F., Pohl, K.: Software Product Line Engineering: Foundations, Principles, and Techniques. In: Springer Science Business Media (2005)
10. CVL Submission Team: Common variability language (CVL), OMG revised submission (2012)
11. PlantUML, http://plantuml.com/
12. Maven, I.: Apache maven project (2011)