
TESIS DOCTORAL

# *Variabilidad Dinámica en Sistemas Sensibles al Contexto*

Autor:
**Alejandro Valdezate Sánchez**

Director/es:
**Dr. Rafael Capilla Sevilla**

Programa de Doctorado en Tecnologías de la Información y las Comunicaciones

Escuela Internacional de Doctorado

2022

Agradecimientos:

*A Rafa, por su paciencia y esfuerzo,*
*a Mabel, por su gran apoyo,*
*a Alex y David, por su luz,*
*a todos los que habéis estado cerca.*

# Abstract

The current diversity and configuration options of today's software requires adequate techniques to configure different products efficiently and maximize the reuse of software components. For this reason, numerous software systems use the so-called software variability techniques to describe the common and variable characteristics of systems. This variability allows software engineers to configure multiple products according to the market needs and the diversity of customers in the shortest time. These solutions typically follow what is called a Software Product Line approach.

However, due to the current needs in which many applications depend on context information to reconfigure their systems dynamically, traditional variability models are limited to configure the software variants after post-deployment time. For this reason, and as the main goal of this PhD work, dynamic variability techniques are necessary to reconfigure the variability of systems at runtime. In this research, we suggest a novel dynamic variability solution able to modify at runtime the structural variability of systems and facilitate the reconfiguration of context-aware system that demand reconfiguring their system options dynamically.

# Resumen

La actual diversidad y opciones de configuración del software de hoy en día requieren de técnicas adecuadas para poder configurar diferentes productos de forma eficiente y maximizar la reutilización de los componentes software. Por ello, numerosos sistemas software utilizan las denominadas técnicas de variabilidad software para describir las características comunes y variables de los sistemas. Esta variabilidad permite a los ingenieros software configurar múltiples productos según las necesidades del mercado y los clientes en un menor tiempo posible. Además, permite construir sistemas siguiendo el modelo de lo que se conocen como Líneas de Producto Software.

Sin embargo, debido a las necesidades actuales en las que numerosas aplicaciones dependen de información de contexto para reconfigurar las opciones del sistema de manera dinámica, los modelos de variabilidad tradicionales se ven limitados para configurar sus variantes una vez desplegado el producto. Por ello, y como objetivo de esta tesis, resulta necesario contar con variabilidad dinámica que permitan la reconfiguración y extensión de los modelos de variabilidad en tiempo de ejecución y la modificación de la variabilidad estructural de manera dinámica. En este sentido, en esta tesis doctoral vamos a proponer una solución de variabilidad dinámica que nos permita reconfigurar los denominados modelos de variabilidad en tiempo de ejecución y facilitar la reconfiguración de las opciones en sistemas dependientes del contexto en tiempo de ejecución.

# Índice

# Índice de Figuras

# Índice de Tablas

# Capítulo 1. Introducción

Actualmente, la mayoría de los sistemas software y productos que incluyen una componente software necesitan configurarse para diferentes entornos y clientes. Esta necesidad de configuración puede ser altamente compleja cuando el número de opciones y alternativas de configuración crecen exponencialmente. Por ello, debido a la complejidad de los sistemas intensivos de software actuales, la variabilidad que describe el número de opciones configurables para las diferentes necesidades de cada cliente necesita gestionarse de una manera eficiente para incrementar la flexibilidad de los productos y desplegar el software de manera rápida y efectiva.

Una de las formas más habituales de describir y configurar las diferentes funcionalidades en un producto software es mediante lo que se conoce como *variabilidad software*. Estas técnicas de variabilidad utilizadas desde los años 90 permiten describir las diferentes alternativas y opciones configurables en los sistemas software de manera que se pueden personalizar para cada tipo de cliente o entorno en el que se va a despegar el software.

Sin embargo, uno de los problemas actuales para gestionar esta variabilidad es el alto número de opciones configurables que existen en los sistemas industriales, compuestos generalmente por miles de variantes. La gestión de esta variabilidad software en modelos industriales es altamente compleja no sólo debido al alto número de variantes sino también al conjunto de restricciones que delimitan el número de opciones posibles que pueden implementarse en un determinado producto software. Además, la gestión y visualización de grandes modelos de variabilidad es altamente compleja y difícil de administrar por las herramientas actuales que soportan modelos de variabilidad causada por la problemática de mostrar al ingeniero software grandes modelos de variabilidad con cientos o miles de opciones configurables.

Por otra parte, la gestión de la variabilidad en el lado del cliente resulta, si cabe, más compleja debido a la necesidad de permitir la configuración y selección de opciones de un sistema en tiempo de ejecución.

En este sentido, poder disponer de soluciones que automaticen la gestión de la variabilidad de forma dinámica sería altamente beneficioso para el tratamiento de la variabilidad con mínima intervención humana y, de esta manera, poder retrasar las decisiones de diseño hasta la ejecución del sistema.

Actualmente, dado que numerosos sistemas utilizan información del contexto para reconfigurarse o actualizar software de manera inteligente y automática, se hace necesario poder gestionar dichas opciones en tiempo de ejecución y en muchos casos con mínima intervención humana.

Es por ello que en este trabajo trataremos de abordar mecanismos que faciliten la reconfiguración dinámica de los sistemas que utilizan variabilidad software y su despliegue en tiempo de ejecución, al que tiempo que investigaremos la automatización de los cambios estructurales en los modelos de variabilidad software con el fin de que sean abiertos y extensibles más allá del diseño inicial del sistema. Todo ello englobado dentro de lo que actualmente se conoce como *variabilidad dinámica* o variabilidad en *tiempo de ejecución.*

# Capítulo 2. Estado del arte

En este capítulo vamos a describir la problemática sobre la gestión de la variabilidad software tradicional y aquellas propuestas que permiten configurar la variabilidad en tiempo de ejecución.

## 2.1 Variabilidad Software

Actualmente, muchos de los sistemas de software intensivos que se desarrollan utilizan técnicas de variabilidad software para construir productos en menor tiempo. La construcción de productos software relacionados que comparten una funcionalidad común y se distinguen en una funcionalidad específica de cada producto, necesitan de arquitecturas software que sean comunes a una familia de productos software **[BOSC 2000]**. La construcción de familias de productos relacionados se realiza mediante técnicas de ingeniería de líneas de productos **[BASS 2012]**. Las líneas de producto software combinan una arquitectura software común a la familia de productos y permiten especificar los aspectos de variabilidad software que se utilizan para personalizar cada producto software.

Los aspectos comunes (*commonality*) en una familia de productos son aquellas características comunes a todos los miembros de la familia, de manera que los productos software se construyan a partir de componentes reutilizables. Por otra parte, la variabilidad (*variability*) representa aquellos aspectos que diferencian un producto de otro. Sin embargo, existen aspectos variables que son visibles al usuario final y otros aspectos permanecen ocultos hasta que son activados. En este sentido, no toda la variabilidad que se necesita en un producto es ofrecida al cliente.

Según Capilla **[CAPI 2013]**, la variabilidad software se define como la capacidad de un sistema software para ser cambiado, personalizado o configurado para su uso en un contexto particular. Debido a que las características variables de un producto software, denominadas *features*, pueden tener restricciones entre ellas, resulta necesario describir cómo modelar y comprobar dichas restricciones, tal y como describimos en la siguiente sección.

### 2.1.1 Dependencias y restricciones entre características

Con el fin de representar la variabilidad en un conjunto de productos relacionados, la técnica más habitual es utilizar el denominado árbol de variabilidad (*feature tree*) o modelo de variabilidad (*feature model*). Los modelos de variabilidad definen los tipos de relaciones entre las características comunes y variables y pueden ser de los siguientes tipos **[LEE 2004]**:

- **Obligatorias** (*mandatory*): Utilizan una relación lógica AND para describir las características que todos los productos de la misma familia deben tener.
- **Alternativas** (*alternative*): Utilizan las relaciones lógicas OR y XOR para describir características que se pueden seleccionar en diferentes productos de la misma familia, de manera que se pueden seleccionar una o varias características opcionales según la relación lógica escogida.
- **Opcionales:** Son características que pueden estar o no presentes a la hora de configurar un producto.

En la Figura 2.1 **[BENA 2005]** podemos observar un ejemplo de árbol de características donde quedan representadas las cuatro diferentes relaciones posibles (obligatoria, opcional, alternativa, y alternativa exclusiva) y la notación para representarlas utilizada se basa en la modelo *Feature-Oriented Domain Analysis* (FODA) propuesta por **[KANG 1990]**.

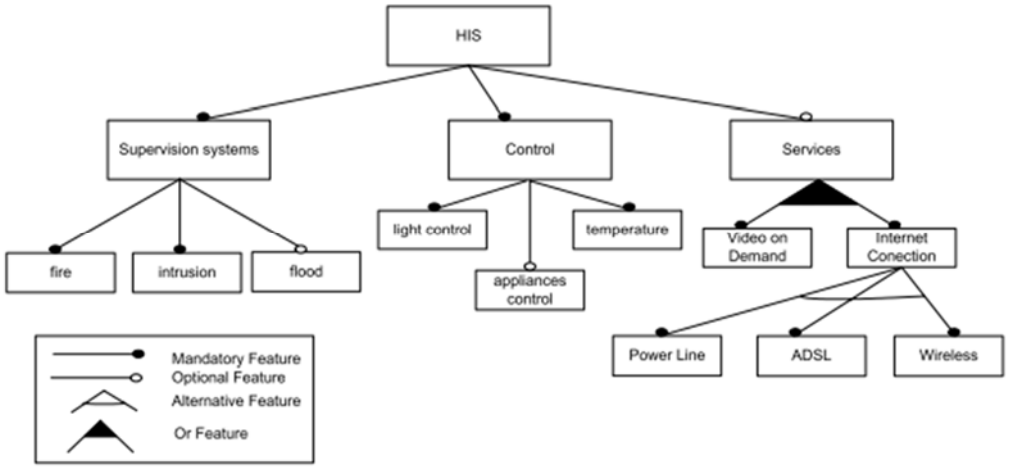



Figura 2.1. Representación de un modelo de variabilidad basado en el modelo FODA

Los modelos de variabilidad como los representados en la Figura 2.2, se describen en términos de variantes y puntos de variación. Una variante representa una característica visible del sistema que toma un conjunto de valores admisibles. Estos valores permiten configurar las características del producto software de diferentes maneras. Por otro lado, podemos agrupar diferentes variantes o características relacionadas y de manera lógica, que es lo que habitualmente se conoce como punto de variación. Un punto de variación define una fórmula lógica que conecta las variantes. Dado que la fórmula lógica que representa los puntos de variación solamente permite describir relaciones lógicas del tipo AND, OR, XOR, es necesario representar las restricciones entre las variantes, tal y como describimos a continuación.

**Dependencias *require* y *exclude***: Una dependencia de tipo *require* indica que una característica o variante requiere de otra o de varias, mientras que el tipo *exclude* indica que para activar una determinada característica existe otra que no puede haber sido seleccionada o activada. Este tipo de relaciones transversales al modelo FODA se describen a modo de ejemplo en la Figura 2.2.

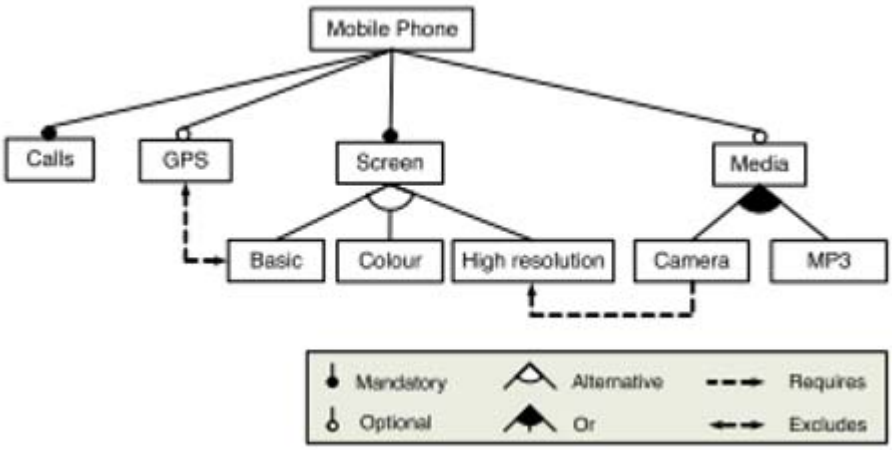



Figura 2.2. Modelo de variabilidad con restricciones de tipo *require* y *exclude*

En el árbol de la Figura 2.2 **[BENA 2010]** se representan las restricciones de tipo *require* mediante una línea discontinua dirigida, mientras que las restricciones de tipo *exclude* quedan representadas mediante una línea discontinua dirigida en ambos extremos que une las dos características que son excluyentes entre sí.

Debido a que la representación gráfica de dichas dependencias complica en exceso la representación visual de los árboles de variabilidad, se suelen utilizar dos notaciones principales para modelar dichas restricciones. Estas notaciones se conocen como con *Constraint Satisfaction Problems* (CSP) y *Boolean Satisfiability Problem* (SAT). Una notación SAT se caracteriza, fundamentalmente, por ser una expresión lógica basada en fórmulas que se construyen con los operadores lógicos AND, OR y NOT y que con dichas fórmulas permite definir relaciones entre las características. La notación SAT permite comprobar fórmulas proposicionales en formato booleano y determina si pueden cumplirse.

En el caso de los solucionadores de fórmulas SAT, cada expresión viene definida por tres operadores: AND, OR y NOT, permitiendo únicamente estos 3 elementos lógicos para definir las dependencias y restricciones, tal y como puede verse en la Figura 2.3 [BENA 2010].

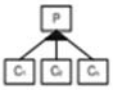
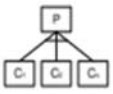
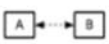
| Relationship | | PL Mapping | Mobile Phone Example |
|---|---|---|---|
| MANDATORY |  | $P \leftrightarrow C$ | MobilePhone ↔ Calls<br>MobilePhone ↔ Screen |
| OPTIONAL |  | $C \rightarrow P$ | GPS → MobilePhone<br>Media → MobilePhone |
| OR |  | $P \leftrightarrow (C_1 \vee C_2 \vee ... \vee C_n)$ | Media ↔ (Camera ∨ MP3) |
| ALTERNATIVE |  | $(C_1 \leftrightarrow (\neg C_2 \wedge ... \wedge \neg C_n \wedge P)) \wedge$<br>$(C_2 \leftrightarrow (\neg C_1 \wedge ... \wedge \neg C_n \wedge P)) \wedge$<br>$(C_n \leftrightarrow (\neg C_1 \wedge \neg C_2 \wedge ... \wedge \neg C_{n-1} \wedge P))$ | (Basic ↔ (¬Color ∧ ¬Highresolution ∧ Screen)) ∧<br>(Color ↔ (¬Basic ∧ ¬Highresolution ∧ Screen)) ∧<br>(Highresolution ↔ (¬Basic ∧ ¬Color ∧ Screen)) |
| IMPLIES |  | $A \rightarrow B$ | Camera → Highresolution |
| EXCLUDES |  | $\neg(A \wedge B)$ | ¬(GPS ∧ Basic) |

Figura 2.3. Ejemplo de restricciones booleanas en notación SAT

Por otra parte, la notación CSP no sólo permite expresiones booleanas, sino que también evalúa valores como números enteros o intervalos. Sin embargo, la notación CSP, al ser más rica y extensa, permite definir expresiones con mucha más información y exactitud, incluyendo rangos de valores y números enteros en el lenguaje de descripción de las fórmulas con los que validar la restricción, tal y como puede verse reflejado en la Figura 2.4.

.

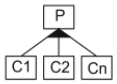
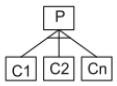
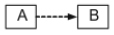
| Relationship | | CSP Mapping | Mobile Phone Example |
|---|---|---|---|
| MANDATORY |  | P = C | Mobilephone = Calls<br>Mobilephone = Screen |
| OPTIONAL |  | if (P = 0)<br>C = 0 | if (Mobilephone = 0)<br>GPS = 0<br>if (Mobilephone = 0)<br>Media = 0 |
| OR |  | if (P > 0)<br>Sum (C1, C2,...Cn) in {1..n}<br>else<br>C1 = 0, C2 = 0,...., Cn = 0 | if (Media > 0)<br>Sum (Camera, MP3) in {1..2}<br>else<br>Camera = 0, MP3 = 0 |
| ALTERNATIVE |  | if (P > 0)<br>Sum (C1 ,C2,...Cn) in {1..1}<br>else<br>C1 = 0, C2 = 0,...., Cn = 0 | if (Screen > 0)<br>Sum (Basic, Colour, High resolution) in {1..1}<br>else<br>Basic = 0, Colour = 0, High resolution = 0 |
| REQUIRES |  | if (A > 0)<br>B>0 | if (Camera > 0)<br>High resolution > 0 |
| EXCLUDES |  | if (A > 0)<br>B = 0 | if (GPS > 0)<br>Basic = 0 |

Figura 2.4. Ejemplo de restricciones booleanas en notación CSP

El software que permite resolver restricciones CSP y/o SAT se conoce como software solver (SAT[1] Solver o CSP Solver) o resolvedores de restricciones como por ejemplo Alloy o BDD solver. Entre las propuestas más relevantes para gestionar las restricciones en los modelos de variabilidad tenemos el trabajo fundamental de **[BENA 2010]** que describe una revisión sistemática de la literatura sobre qué operaciones de análisis se pueden practicar en modelos de variabilidad y las herramientas que procesan los modelos de restricciones en variabilidad software.

[1] Existen otros resolvedores de restricciones, como son los SMT (*Satisfiability Modulo Theories*) **[SPREY 2020]**, generalmente más rápidos que SAT para generar explicaciones de peticiones que no pueden satisfacerse, pero su discusión queda fuera del objetivo de esta tesis.

Los autores destacan que uno de los problemas fundamentales es resolver restricciones en modelos de variabilidad grandes. Por ello, la resolución de restricciones en modelos de gran tamaño puede mejorarse con algoritmos evolutivos como el descrito en **[SEGU 2014]**, en el cual los autores aplican dichas técnicas en modelos de hasta 5000 características.

Por otra parte, en **[YE 2005]** se describe un tipo de restricción complementaria a las conocidas como *require* y *exclude* y se denomina *impact*. Esta restricción entre variantes se define como el impacto que tiene en una característica cuando se selecciona otra para una determinada configuración. Sin embargo, los autores no detallan de qué manera se evalúa este posible impacto.

### 2.1.2 Variabilidad en espacio y tiempo

Dos aspectos clave en la gestión de la variabilidad hacen referencia al número y tipo de productos software que podemos construir y el momento en el que se pueden configurar las diferentes opciones de los productos software. En el primer caso, tratamos de delimitar el ámbito del número y tipo de productos configurables mediante variabilidad y es lo que comúnmente se conoce como *variabilidad en espacio* y que nos permite determinar el tipo y número de productos software factibles. Por otra parte, la *variabilidad en tiempo* hace referencia al momento en el cual las variantes toman un valor concreto y que se conoce como *binding time*. El *binding time* permite retrasar las decisiones de diseño hasta el último momento posible y de esta manera configurar las opciones de los productos software en etapas diferentes del proceso de desarrollo software.

**Variabilidad en espacio**: Permite seleccionar el número y tipo de productos que se van a construir. Esa variabilidad en espacio se realiza mediante las opciones configurables, es decir, las variantes del modelo de variabilidad. Generalmente, la variabilidad en espacio es utilizada por las líneas de producto software para delimitar la cantidad y el tipo de productos a construir y se describe mediante una actividad denominada *domain scoping*. En el trabajo de Käkölä **[KÄKÖ 2007]** los autores relacionan la variabilidad en espacio con el ámbito de las líneas de productos software, de manera que el concepto de *domain scoping* se utiliza para reducir el ámbito del número y tipo de productos posibles a construir dentro de la línea de productos.

A modo de ejemplo, el caso de estudio de la línea de productos de Toshiba en plantas eléctricas (EPG-SPL) **[MATS 2007]**, establece los límites del dominio de aplicación dirigido por el equipo de desarrollo del sistema de operación de plantas de Toshiba (i.e. COPOS) como encargado de conducir las tareas de análisis del dominio y de identificar 11 características de la línea de productos EPG. Asimismo, **[BART 2012]** describe de manera más detallada una actividad completa para delimitar la cartera de productos a construir (*product portfolio*) en una línea de productos para sistemas médicos. Los autores sugieren estudiar el solapamiento de las distintas aplicaciones en diferentes subdominios para descubrir los aspectos comunes en la línea de productos y delimitar el ámbito de los productos a construir.

Una de las formas de construir modelos de variabilidad de manera automática para probar las diferentes configuraciones de lo que se entiende por variabilidad en espacio, es utilizar herramientas como, por ejemplo, BeTTy **[SEGU 2012]**, la cual permite generar modelos de variabilidad y comprobar la validez del modelo en base a las restricciones entre las características. A modo de ejemplo, la Figura 2.5 muestra cómo se expresaría un árbol de características utilizando BeTTy.

```
%Relationships
'root' : ['F1'];
'F1' : ['F2'];
'F1' : [1,1] 'F3' 'F4' 'F5' 'F6';
'F3' : 'F7';
'F7' : ['F24'];
'F7' : ['F25'];
'F3' : ['F8'];
'F3' : 'F9'
'F3' : 'F10';
'F40' : ['F49'];
'F18' : [1,1] 'F41' 'F42';
'F5' : ['F19'];
'F19' : [1,2] 'F47' 'F48';
'F5' : 'F20';
'F20' : ['F33'];
'F20' : 'F34';
%Constraints
'F6' EXCLUDES 'F27';
'F49' EXCLUDES 'F12';
```

Figura 2.5. Descripción de la variabilidad generada con BeTTy

**Variabilidad en tiempo**: Permite retrasar las decisiones de diseño en la variabilidad software hasta el último momento. De esta manera, la variabilidad en tiempo es una propiedad que permite al diseñador configurar los variantes del sistema en fases posteriores al diseño o incluso permitir al usuario final configurar parte de esta variabilidad **[ELSN 2010]**. Una de las primeras propuestas para definir la variabilidad en tiempo es la descrita por **[FRIT 2002]**. En dicho trabajo se describen posibles tiempos de ligadura que van desde las fases de diseño a las de ensamblado y configuración del producto. Asimismo, los autores proponen un tiempo de ligadura durante la ejecución del sistema que puede ir desde el primer arranque a una configuración en tiempo de ejecución, pero sin detallar como realizarlo. Otra de las propuestas para describir *binding times* es **[LEEJ 2008]**, cuyos autores sugieren la activación de grupos de características denominadas FBU (*Feature Binding Units*) de manera simultánea, tal y como se muestra en la Figura 2.6 para la activación de diferentes opciones en una oficina virtual.

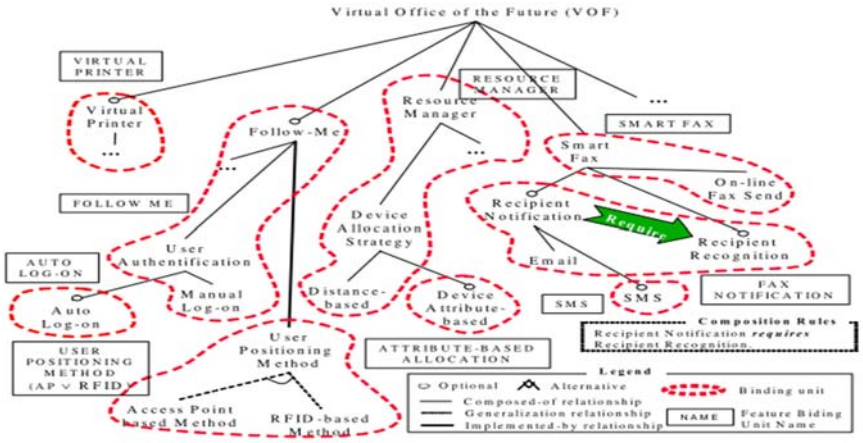



Figura 2.6. *Feature binding units* para activar grupos de características de manera simultánea

En **[CAPI 2013]**, los autores describen posibles momentos para configurar las variantes que van, desde la parte más estática concerniente al desarrollador software, hasta la parte más dinámica que permite al cliente configurar los variantes del sistema, tal y como muestra la Figura 2.7. A diferencia de propuestas anteriores, se describen cuáles de los *binding times* pertenece al lado del desarrollador y en qué momentos las variantes se pueden configurar en el lado del cliente en tiempo de post-despliegue. Por ejemplo, un usuario de un *smartphone* puede activar la opción de bluetooth en el teléfono móvil sin intervención del desarrollador.

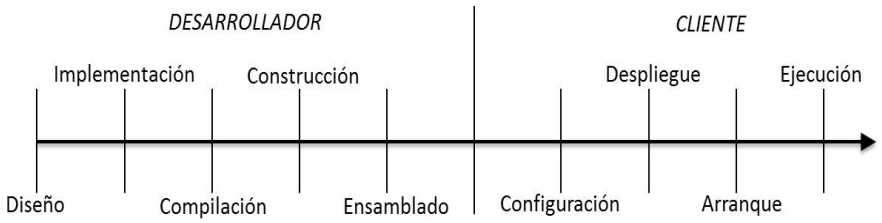



Figura 2.7. Taxonomía de *binding times* en el lado del desarrollador y el cliente

Recientemente, algunos autores **[ANAN 2020]** han propuesto la unificación de los conceptos de variabilidad en espacio y tiempo a través de un metamodelo con el fin de solucionar problemas en los sistemas de control de versiones cuando la variabilidad evoluciona con el tiempo. Sin embargo, el modelo conceptual propuesto queda limitado al control de versiones de la variabilidad en la línea de productos y no contempla los aspectos específicos de la variabilidad dinámica.

Esta propuesta se ha refinado en **[ANAN 2022],** donde los autores definen un conjunto de operaciones unificadas que incluyen precondiciones y postcondiciones orientadas a combinar las diferentes capacidades de herramientas de gestión de la variabilidad. A pesar de que la idea de unificar las operaciones de variabilidad en tiempo y espacio por parte de las herramientas actuales es muy interesante, se considera que es una idea algo inmadura de llevar a la práctica, dada la diversidad y naturaleza de las diferentes herramientas y repositorios.

Este mismo problema ha sido descrito por **[MICH 2020]**, donde la propagación de cambios en las características puede afectar a los sistemas de control de versiones y revisión de características. De igual manera, esta propuesta no contempla los aspectos de variabilidad en tiempo de post despliegue, limitándose a las configuraciones que ocurren en tiempo de compilación o construcción del producto software.

## 2.2 Representación de la variabilidad

Con el fin de poder representar las características visibles de una familia de sistemas en forma de variantes y puntos de variación y las relaciones entre ellos, existen diferentes técnicas y notaciones que van desde las gráficas hasta las textuales, tal y como describimos a continuación.

### 2.2.1 Representación de la variabilidad en forma de árbol

El primer modelo que introdujo el término *feature* como característica del sistema visible para el usuario final fue el modelo FODA (Feature-Oriented Domain Analysis) **[KANG 1990]**. Desde entonces, esta forma de modelar las características de un sistema está ampliamente aceptada por la disciplina de ingeniería de líneas de producto. A modo de ejemplo, la Figura 2.8 **[CAPI 2013]** muestra un árbol de características FODA para construir software para teléfonos móviles.

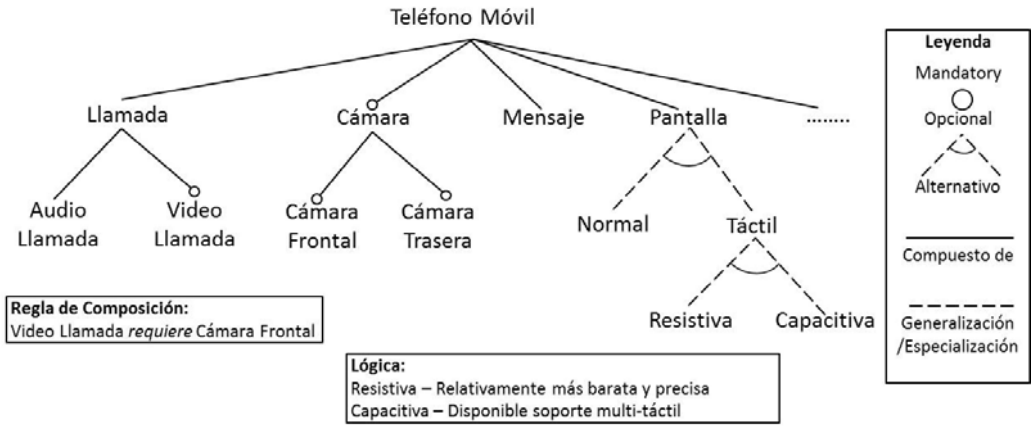



Figura 2.8. Ejemplo de modelo de características FODA

A continuación, resumimos en la Tabla 2.1 la historia más reciente de las distintas aproximaciones que han ido surgiendo para describir modelos de características con diferentes notaciones y extensiones al modelo FODA original. Aquellas propuestas que derivan o extienden algún modelo anterior planteado aparecen agrupadas.

Tabla 2.1. Propuestas de representación de la variabilidad

| Propuesta | Descripción | Autores |
|---|---|---|
| **FODA** | Modelo original para representar variabilidad software en forma de árbol. | [KANG 1990] |
| **FORM Feature Model** | Extensión de FODA para construir arquitecturas de dominio y un nuevo tipo de relación denominado *implemented-by*. | [KANG 1998] |
| **FeatuRSEB Feature Model** | FeatuRSEB integra modelos FODA con procesos de reutilización y propone características alternativas. El modelo de Van Gurp introduce características denominadas externas y generaliza y especializa las relaciones alternativas. El modelo PLUSS añade únicamente cambios en la notación. | Griss et al. (1998)<br>Van Gurp et al. (2001)<br>Eriksson et al. (2005) |
| **Hein et al. Model** | Introduce dependencias entre características. | Hein et al. (2000) |
| **Generative Programming Feature Model**<br>Representación de valores cuantitativos | GP refina las relaciones XOR y OR mientras que las aproximaciones de Capilla, Riebisch y Czarnecki sugieren incluir cardinalidad y grupos de características. La aproximación de Benavides introduce atributos en las características. | Czarnecki et al. (2000)<br>[CAPI 2001]<br>Riebisch et al. (2002)<br>Czarnecki et al. (2002)<br>[CZAR 2004]<br>[BENA 2005] |

**FODA y orígenes**: En los orígenes, se plantearon los modelos en forma de árboles de características, más conocidos como FODA. Una de las primeras extensiones, denominada FORM (*Feature-Oriented Reuse Method*), el cual amplía FODA de manera que prescribe cómo debe utilizar un modelo de características para desarrollar arquitecturas de dominio y componentes reutilizables.

**Características alternativas, externas y dependencias**: La siguiente evolución al modelo FODA, como el denominado *FeatuRSEB*, sugiere la utilización de características alternativas. Posteriormente, el modelo de Van Gurp introduce características denominadas externas, así como especialización y generalización de relaciones entre dichas características.

Finalmente, el modelo *PLUSS* únicamente introduce cambios en la notación de los modelos de características. De manera complementaria a las características alternativas, el modelo propuesto por **[HEIN 2000]** sugiere el establecimiento de dependencias de tipo *require/exclude* para definir restricciones entre las características.

**Valores cuantitativos, cardinalidad y atributos**: La siguiente evolución a los modelos FODA introduce valores cuantitativos y cardinalidad en las variantes asociados a atributos de calidad en sistemas de telecomunicaciones, tal y como refleja la propuesta **[CAPI 2001]** en el ejemplo que se muestra en la Figura 2.9. De esta manera, es posible asociar reglas externas al modelo FODA para controlar aspectos de calidad en estos sistemas e introducir más de un posible valor en las variantes a través de la cardinalidad.

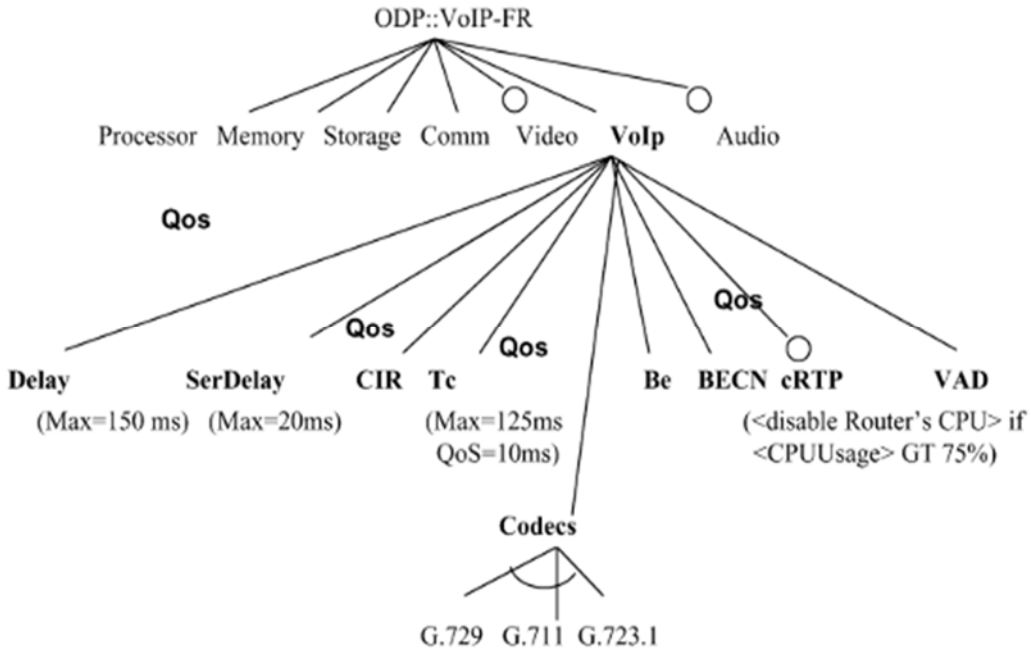



Figura 2.9. Representación de valores cuantitativos en modelos FODA

Además, la propuesta de **[BENA 2005]** sugiere la introducción de atributos en los modelos de características como extensión al modelo de Czarnecki **[CZAR 2000]**. Los autores proponen el concepto atributo como una característica medible en las *features* y similar a la propuesta descrita en **[CAPI 2001]** para representar valores cuantitativos medibles asociados a características en árboles FODA. Además, los autores sugieren dominios de atributos como el espacio de posibles valores donde los atributos toman su valor.

Finalmente, proponen también el concepto de características extra funcionales que se definen como una relación entre uno o varios atributos de una característica. La notación propuesta para describir los atributos es muy similar a la descrita en **[CAPI 2001]** para los valores cuantitativos como extensión de los árboles FODA tradicionales. Estas características son descritas en notación CSP para poder ser evaluadas de manera automática por un resolvedor de restricciones y así seleccionar los productos óptimos.

**Representación gráfica en herramientas de gestión de variabilidad**: La representación de los modelos FODA y extensiones por las diferentes herramientas para gestionar la variabilidad software es muy parecida. Aunque actualmente existen más de 40 herramientas de tipo *research* y dos comerciales, describimos a continuación dos ejemplos de cómo se representa la variabilidad en este tipo de software.

Por ejemplo, la herramienta de *research* FeatureIDE, permite representar las diferentes dependencias y restricciones, y el árbol de variabilidad se muestra de manera horizontal, tal y como muestra la Figura 2.10.

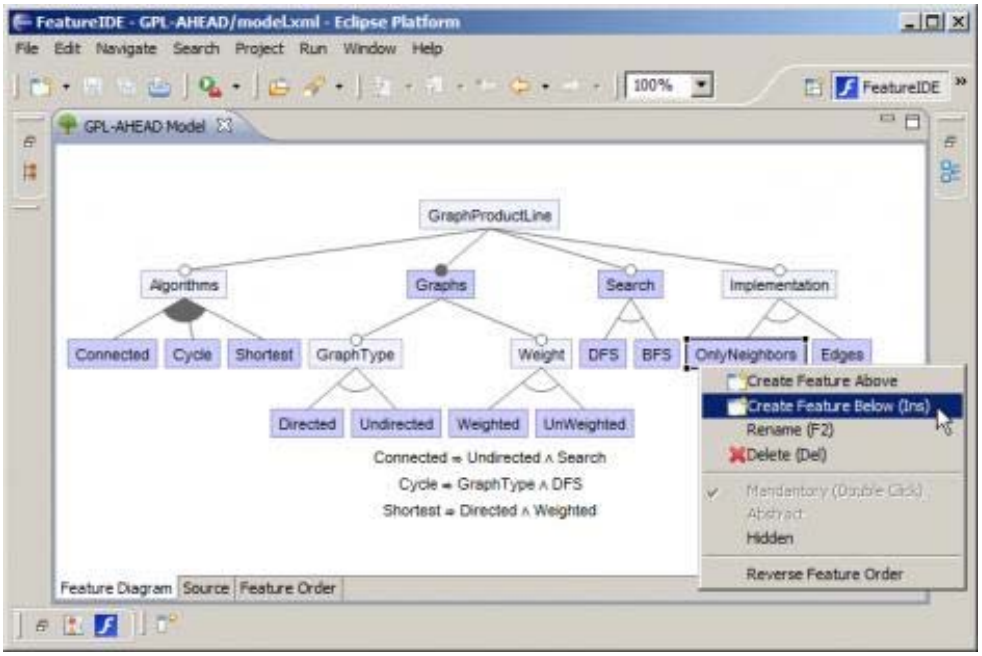



Figura 2.10. Modelado de la variabilidad con FeatureIDE

Sin embargo, debido a los problemas de visualización y escalabilidad, resulta más conveniente representar los árboles FODA de manera vertical, tal y como se muestra en la herramienta comercial *Pure::Variants*[2] mostrada en la Figura 2.11, y correspondiente a la variabilidad de una vehículo aéreo no tripulado.

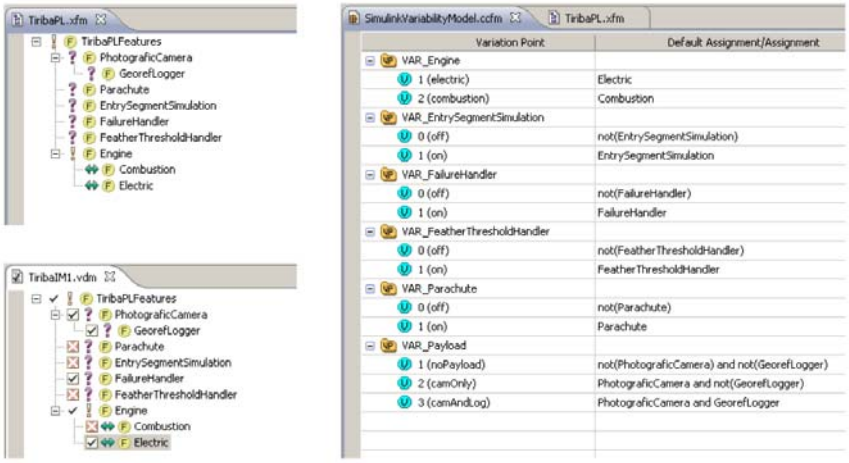



Figura 2.11. Modelado de la variabilidad con *Pure::Variants*

## 2.2.2 Lenguajes y notaciones para modelar la variabilidad

De manera complementaria a las representaciones de árboles de características siguiendo los modelos FODA, existen notaciones gráficas y lenguajes especializados para describir la variabilidad, entre los que podemos destacar: *Orthogonal Variability Language* (OVM), *Common Variability Language* (CVL) y *Variability Exchange Language* (VEL).

***Orthogonal Variability Language* (OVM)**: La representación mediante OVM **[POHL 2005]** modela los árboles de variabilidad en notación UML. En la Figura 2.12 se muestra el proceso de traducción de la variabilidad de una línea de producto de teléfonos móviles. En el modelo de representación en OVM se muestra la variabilidad del sistema mediante 2 puntos de variación y 3 variantes.

[2] https://www.pure-systems.com/

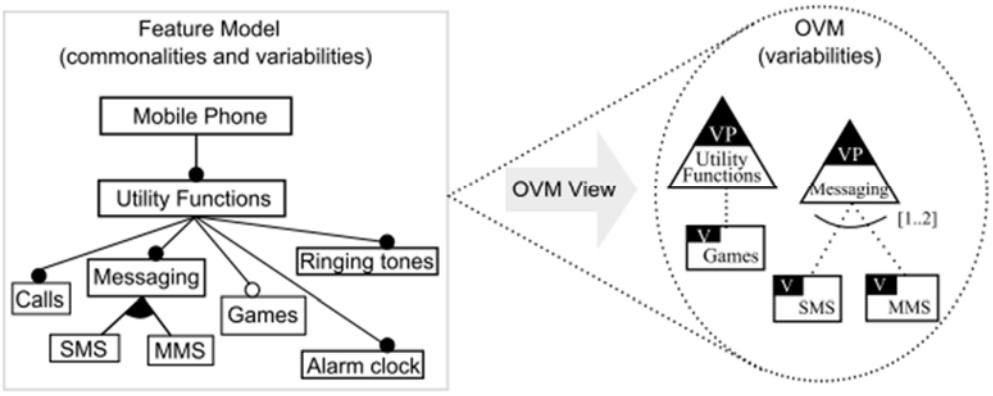



Figura 2.12. Representación de la variabilidad con OVM

**Common Variability Language (CVL)**: El segundo de los modelos, corresponde al *Common Variability Language* (CVL) **[HAUG 2013]**, siendo éste un lenguaje independiente del dominio para especificar y resolver variabilidad. CVL facilita la resolución de la variabilidad en cualquier lenguaje independiente del dominio utilizando el modelo *Meta-Object Facility* (MOF)[3] para mapear el modelo de variabilidad en CVL a un lenguaje específico de dominio (DSL) y mediante reglas de transformación resolver la variabilidad para un dominio de aplicación concreto. El funcionamiento general del modelo CVL se muestra en la Figura 2.13. Un ejemplo reciente de la utilización del lenguaje CVL se describe en **[HORC 2017]**, donde los autores plantean una extensión al motor que realiza las transformaciones (*M2M - Model to Model*) permitiendo alterar los modelos de variabilidad y posibilitando incluir etapas adicionales para antes de materializar un modelo de variabilidad, con el fin de comprobar la validez de las transformaciones especificadas en la semántica extendida de los puntos de variación, tal y como se muestra en la Figura 2.13.

[3] https://www.omg.org/mof/

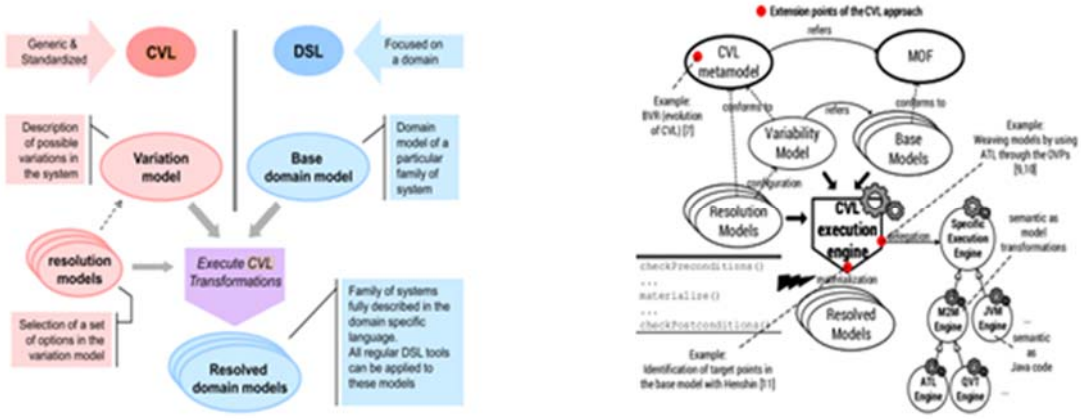



Figura 2.13. Representación de la variabilidad con OVM

**Variability Exchange Language (VEL)**: El tercer lenguaje de modelado de variabilidad y el más reciente es VEL **[SCHU 2015]**, el cual permite el intercambio de información entre herramientas de gestión de la variabilidad y herramientas de desarrollo software. Mediante VEL, podemos definir una interfaz común que posibilita la integración entre ambas herramientas, la herramienta que gestiona la variabilidad que gestiona los elementos software en el proceso de desarrollo gracias a la variabilidad introducida en ellos (ver Figura 2.14). De esta manera, los puntos de variación descritos en VEL se implementan en los elementos de software con el fin de poder utilizarlos en la configuración de las características del producto software. VEL posibilita unificar las interfaces para representar puntos de variación en diferentes elementos software.

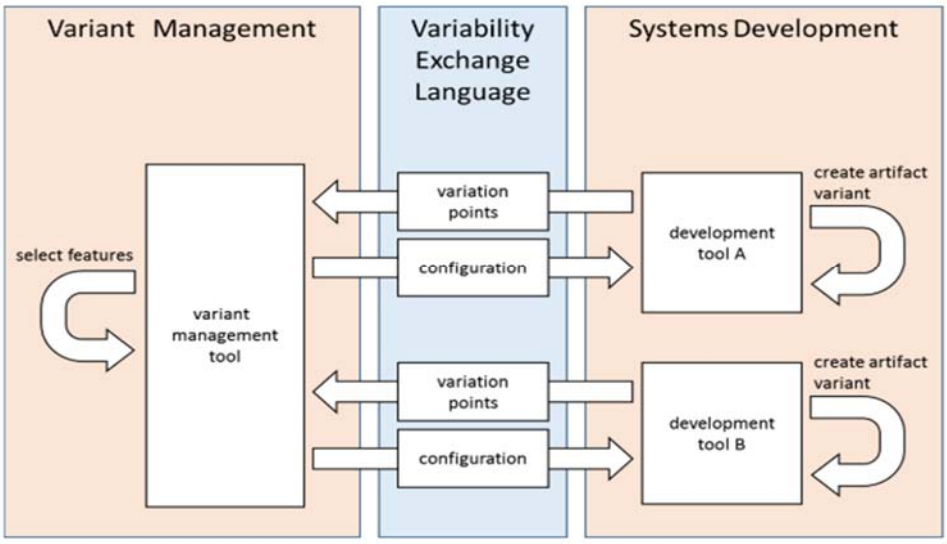



Figura 2.14. Esquema general de funcionamiento de VEL.

Además de soportar los modelos de variabilidad típicos, VEL define formas para asociar condiciones a variantes. Por ejemplo, un punto de variación estructural define una condición acerca de un artefacto software. Los puntos de variación estructurales son aquellos donde la estructura de un modelo cambia durante el proceso de *binding* de las variantes. Además, los puntos de variación por parámetros son aquellos que permiten seleccionar un valor numérico para un parámetro, pero no alteran la estructura del modelo de variabilidad. Un ejemplo de este lenguaje se puede ver en la Figura 2.15, en el cual se detalla la creación de un punto de variación estructural opcional el cual incluye un identificador opcional para el atributo vp1 **[GROß 2015]**.

```
<optional-structural-variationpoint id="vp1" name="optional variationpoint">
  <special-data name="CreatorInfo">
    <data>
      <key>Created</key>
      <value type="xs:date">1998-11-17</value>
    </data>
  </special-data>
  <variation id="vp1v1" name="optional variation">
    <condition type="single-feature-condition">Feature1</condition>
  </variation>
</optional-structural-variationpoint>
```

Figura 2.15. Código XML para representar un punto de variación estructural en VEL.

## 2.2.3 Lenguajes textuales

A pesar de que la notación VEL es textual, existen otros lenguajes para describir la variabilidad software y que son comparables a lenguajes de programación tradicionales. Eichelberger y Schmid **[EICH 2015]** exponen 11 lenguajes para describir variabilidad textual de líneas de productos software, entre los que se encuentran VSL, FAMILIAR, Clafer, TVL y VELVET, entre otros. Estos lenguajes mejoran los problemas de escalabilidad de propuestas basadas únicamente en notaciones gráficas y basadas en tablas. A continuación, describimos algunos de estos lenguajes.

El denominado *Feature Description Language* (FDL), **[DEUR 2002]** es una notación textual para diagramas de características y que aplica los principios de los lenguajes de dominio específico (DSLs). Además, cuenta con un conjunto de reglas para operar sobre los modelos FDL.

El lenguaje *Variability Specification Language* (VSL) **[ABEL 2010]** pretende la integración de modelo de características con enlaces de configuración y entidades. Los enlaces de configuración representan referencias para configurar un modelo de variabilidad destino a partir de una configuración determinada.

El denominado *Simple XML Feature Model* (SXFM) **[MEND 2009]** utiliza XML y metadatos para describir modelos de variabilidad y es el soporte del repositorio *Software Product Lines Online Tools* (S.P.L.O.T.). Por otra parte, FAMILIAR **[ACHE 2013]** es un lenguaje de *scripting* que permite definir, analizar y manipular modelos de características y consta de un editor de texto y herramientas gráfica basadas en Eclipse para manipular los modelos de variabilidad.

Otros lenguajes como el *Text-based Variability Language* (TVL) **[CLASS 2011]** utilizan una notación textual para el modelado de características para así favorecer la escalabilidad de estos modelos. Permite además cardinalidad, atributos y modularización. La propuesta denominada *CLAss, FEature, Reference approach* (Clafer) **[BAK 2010] [LIAN 2012]** combina el meta-modelado de clases con el modelado de características utilizando una semántica uniforme para ambos modelos. Clafer proporciona mecanismos de especialización y extensión a través de restricciones y herencia para soportar cardinalidad y múltiples instancias entre otros aspectos.

VELVET es una propuesta inspirada en TVL **[SCHR 2013]** y que permite separar aspectos multi-dimensionales de los modelos de variabilidad. La herramienta para VELVET incluye además la conversión de la notación SXFM a VELVET. *INDENICA Variability Modeling Language* (IVML)[4] es un lenguaje escalable para modelar la variabilidad e manera textual y se distingue de los anteriores en que se permite el modelado de requisitos (e.g. variabilidad expresada en términos no booleanos o dependencias complejas) que son relevantes para configurar y personalizar ecosistemas software complejos. En **[BEEK 2019]** los autores proporcionan una visión actual sobre las oportunidades y desafíos para diseñar lenguajes de variabilidad textuales y discuten diferentes aspectos sobre 13 lenguajes, desde aspectos de configuración de elementos hasta escalabilidad y diseño modular.

4 INDENICA Project Consortium. INDENICA Research Project http://www.indenica.eu

Finalmente, en **[VILL 2019]**, los autores introducen un nuevo lenguaje textual y extensible para describir la variabilidad denominado *High-Level Variability Language* (HLVL) y basado en una ontología. La propuesta recoge la mayoría de las construcciones expresadas en lenguajes similares como pueden ser los atributos y la multiplicidad en características, pero incluye otras construcciones para especificar de manera explícita aspectos comunes (i.e. *common*) a una línea de productos, reglas de inclusión y exclusión condicional (i.e. *mutex*) así como reglas para expresar la disponibilidad de un grupo de elementos.

A modo de resumen, podemos constatar que existen tres aproximaciones principales para modelar la variabilidad y quelas notaciones gráficas sufren de problemas de escalabilidad y visualización, siendo no aptas para grandes modelos de variabilidad.

En cuanto a las propuestas textuales, son más adecuadas para ser automatizadas e incluidas en herramientas de modelado y gestión de la variabilidad, pero el gran número de herramientas existentes complica la interoperabilidad de los modelos y no hay una propuesta claramente superior a las otras, aunque si es cierto que hay un número de herramientas de gestión de variabilidad (no descritas en esta tesis) que son más populares. Sin embargo, prácticamente ninguno de los lenguajes textuales, excepto VEL que es más reciente, incluye soporte para la realización de variantes en tiempo de ejecución (*runtime binding*), siendo este uno de los aspectos centrales de esta tesis.

## 2.3 Variabilidad de Contexto

La primera motivación para soportar variabilidad en tiempo de ejecución son los cambios en las configuraciones del software motivados por cambios en el entorno o contexto. En este sentido, en los últimos años han surgido modelos de variabilidad asociados a características que cambian con el contexto. Hoy en día, existen numerosos sistemas auto-adaptativos, inteligentes, ubicuos y autónomos que requieren información del contexto para adaptar su comportamiento de forma dinámica según varíen las condiciones del entorno.

Por ello, unas de las propuestas existentes son los denominados lenguajes de programación orientados al contexto (*Context-Oriented Programming*), más conocidos como COP. Estos lenguajes permiten la adaptación dinámica del sistema frente a cambios en el contexto, y facilitan añadir nuevas características y modificar el comportamiento mediante la activación y desactivación de contextos denominados *layers*. Actualmente, existen diferentes propuestas de lenguajes COP, como son: *ContextL*, JCOP, *Subjective-C, Context traits*, o AspectJ **[COST 2005] [APPE 2013] [GONZ 2013] [KAMI 2013] [KAMI 2014]**. La activación de contexto en tiempo de ejecución implica un comportamiento dinámico del sistema y una variabilidad implícita. Asimismo, la activación explícita de dependencias entre contextos con una clasificación de soluciones para dominios concretos se describe en **[MENS 2016]**.

En otra clasificación posterior, **[MENS 2017]** describe el problema de la interacción de contextos cuando los comportamientos son incompatibles y resume diferentes estrategias de resolución de conflictos basadas en un modelo de 5 dimensiones. Por tanto, el rol de las denominadas características de contexto y la variabilidad de contexto está cobrando cada vez más importancia como una sub-disciplina emergente para modelar y gestionar los cambios de contexto de los sistemas en tiempo de ejecución.

Una de las primeras propuestas para modelar y describir la variabilidad de contexto es la propuesta de **[HART 2008]**, donde los autores proponen el uso de dos árboles de variabilidad, uno para describir las características de contexto y otro para las del sistema, de manera que las características de contexto están conectadas a las del sistema por dependencias e influyen en la configuración del producto según qué elementos de contexto se seleccionen. Una propuesta refinada de la anterior se detalla en **[CAPI 2014A]**, donde se sugieren dos alternativas para modelar la variabilidad de contexto y del sistema. En la Figura 2.16, en la parte izquierda se muestran árboles de variabilidad de contexto separados y en la parte derecha las características de contexto aparecen mezcladas en el mismo árbol con las de no contexto. En la primera opción (Figura 2.16 izqda.) las características de contexto se modelan como una sub-rama del árbol de variabilidad, lo que lo convierte en fácil de modelar y más reutilizable, aunque puede introducir más dependencias entre ambos tipos de características.

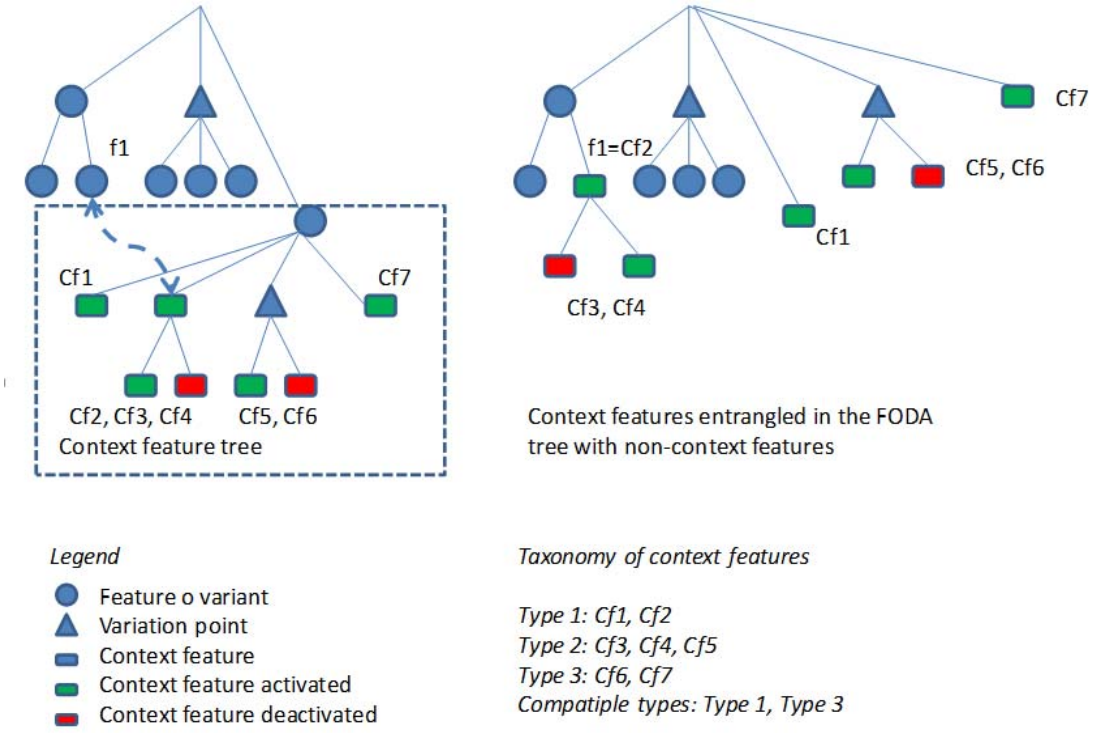



Figura 2.16. Formas diferentes de modelar la variabilidad de contexto

En la segunda opción (Figura 2.16 dcha.), ambos tipos de características aparecen mezcladas, lo cual resulta más complicado de modelar y menos reutilizable si deseamos intercambiar un contexto por otro. En cambio, reduce el número de dependencias entre características de contexto y no contexto. Como podemos observar en la Figura 2.16, algunas características pueden activarse o no dependiendo de las condiciones de contexto.

Sin embargo, dado que los lenguajes COP solamente permiten activar y desactivar o reemplazar comportamientos de contexto, es necesario ampliar esta variabilidad con más capacidades tales como la modificación estructural de los modelos de variabilidad con el fin de hacerlos más abiertos y extensibles. Es por ello que las propuestas de líneas de producto software dinámicas (DSPLs) van más allá de estos lenguajes, con el fin de proporcionar una visión más integrada entre los cambios de comportamiento y los modelos tradicionales de variabilidad software para convertirla en dinámica, tal y como describimos en la siguiente sección.

En **[PERE 2018]** se detalla una propuesta de un sistema de recomendación basado en información de contexto para predecir la selección de características y poder así configurar líneas de producto software. Los autores proponen modelar los datos de contexto de tres formas diferentes: explícita, implícita e inferida, mediante una serie de métricas funcionales a partir de un conjunto de características válidas. Estas características de contexto se basan en restricciones de dominio y preferencias del usuario para proporcionar una recomendación adecuada y que los autores evalúan mediante diferentes combinaciones de datos contextuales.

En una propuesta reciente **[CAES 2019]**, los autores proponen un método sensible al contexto para reconfigurar sistemas de fabricación procedentes de diferentes vendedores y con el fin de crear productos personalizados bajo demanda. La propuesta utiliza modelos de variabilidad para describir las características de un producto proporcionando un modelo de variabilidad basado en el contexto. Los autores utilizan una ontología para valores de contexto a nuevas configuraciones. Además, la reconfiguración del sistema basada en valores de contexto se basa en el modelo de satisfacción de restricciones descrito en **[MAUR 2016]** así como *HyVarRec*, un motor de reconfiguración dinámica contextual, el cual toma la configuración actual de un dispositivo remoto y los valores actuales de la información de contexto para comprobar si la configuración del dispositivo es válida de acuerdo a la información de contexto o bien en su defecto producir una configuración válida.

## 2.4 Líneas de Producto Software Dinámicas

La segunda motivación para implementar mecanismos de variabilidad en tiempo de ejecución es la incapacidad de las líneas de producto software convencionales para modificar la variabilidad estructural una vez desplegado el software. La capacidad auto adaptativa de muchos sistemas basados en las ideas pioneras de Kephart y Chess sobre la computación autónoma **[KEPH 2003]**, dio lugar a los sistemas basados en el modelo MAPE-K (*Monitoring-Analyzing-Planning-Executing + Knowledge*) propuesto por IBM. El ciclo MAPE-K tal y como se muestra en la Figura 2.17 **[HUEB 2008] [ABBA 2011]**.

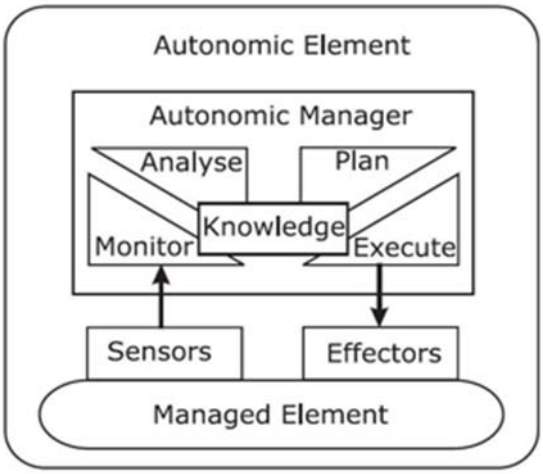



Figura 2.17. Modelo MAPE-K para sistemas auto-adaptativos dependientes del contexto.

Inicialmente, se comienza por monitorizar elementos del sistema que cambian con variaciones en el entorno, se analiza y compara con patrones existentes para conocer si existen deviaciones en los valores, y en caso afirmativo planea una estrategia de cambio o adaptación al entorno y la ejecuta. Todo este modelo suele implicar conocimiento específico del dominio de un tipo de sistemas determinado.

La utilización de los modelos auto adaptativos basados en MAPE-K dieron lugar a las denominadas arquitecturas *runtime* como el denominado modelo Rainbow **[GARL 2009]**, donde se describen elementos de la arquitectura software que permiten alcanzar el objetivo de la auto adaptación del software. Una evolución del modelo anterior es el que se denomina *models@runtime*, el cual combina las ideas del modelo MAPE-K con el concepto de líneas de producto dinámicas **[BLAI 2009] [MORI 2009] [BENC 2012]** en el cual la variabilidad software puede modificarse en tiempo de ejecución mediante un ciclo MAPE-K, tal y como muestra la Figura 2.18.

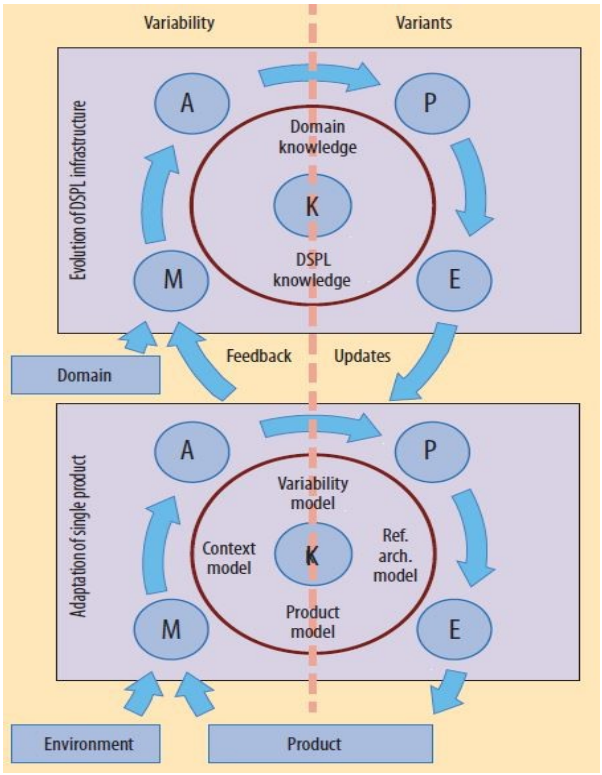



Figura 2.18. Combinación del modelo MAPE-K con aspectos de variabilidad dinámica.

En este sentido, la propuesta de **[MORI 2014]** sugiere el uso de arquitecturas de referencia para mantener modelos en tiempo de ejecución como una nueva clase de sistemas reflectivos capaces de predecir ciertos comportamientos en sistemas cíber-físicos. En **[USMA 2014]** los autores describen un proceso de adaptación para robots utilizando el concepto de *MAPE-K* y proponen modelos de adaptación formales (denominado *ActivFORMS*) que pueden ser ejecutados por una máquina virtual con el fin de satisfacer los objetivos de adaptación y soportar adaptación dinámica del modelo activo si los objetivos cambian. Sin embargo, la adaptación completamente dinámica es limitada ya que el operador del robot debe introducir los objetivos de manera manual.

Debido a que los sistemas basados en *MAPE-K* necesitan reconfigurarse en tiempo de ejecución y satisfacer las restricciones de manera dinámica, en una propuesta reciente **[GADE 2020]**, los autores proponen un modelo denominado *Sce@rist* para verificar restricciones en sistemas auto-adaptativos mediante escenarios en tiempo de ejecución. En el trabajo mencionado se propone un modelo en cinco pasos para verificar el comportamiento de un sistema a través de dichos escenarios descritos en formato MSC (*Message Sequence Charts*) y poder detectar posibles violaciones en el comportamiento utilizando un comprobador de modelos probabilístico.

Como evolución a los modelos MAPE-K y *models@runtime*, uno de los objetivos actuales relacionados con los aspectos de variabilidad dinámica, es que las variantes de un sistema software puedan asumir nuevos valores y puedan ser configuradas en tiempo de ejecución, y si es posible con mínima intervención humana. Por ello, el objetivo de las líneas de producto software dinámicas (DSPLs) es incorporar en el ciclo de vida de las líneas de producto software (SPL) tradicionales mecanismos que posibiliten variabilidad dinámica en los sistemas que demandan una reconfiguración en tiempo de post-despliegue.

El concepto de una DSPL aparece alrededor de 2008 **[HALL 2008]** y entiende que una SPL puede incorporar mecanismos que faciliten cambios en la variabilidad de forma dinámica, implicando ello un cierto grado de automatización. En una propuesta más reciente se propone el uso de SPL autónomas (ASPL) **[ABBA 2015]** con el fin de soportar la adaptación y evolución de productos mediante el control explícito de ciclos de adaptación combinados con mecanismos de variabilidad dinámica. De esta manera, se pretende la auto-adaptación y optimización de los sistemas autónomos como una evolución del ciclo MAPE-K y utilización de composiciones dependientes del contexto y reconfiguración de las variantes en tiempo de ejecución.

En la propuesta descrita en **[CAPI 2014]**, se muestra una modificación al ciclo de vida dual de una SPL en el que ciertas actividades se extienden y se añaden otras nuevas para soportar los cambios en la variabilidad en tiempo de ejecución y post-despliegue, tal y como se muestra en la Figura 2.19.

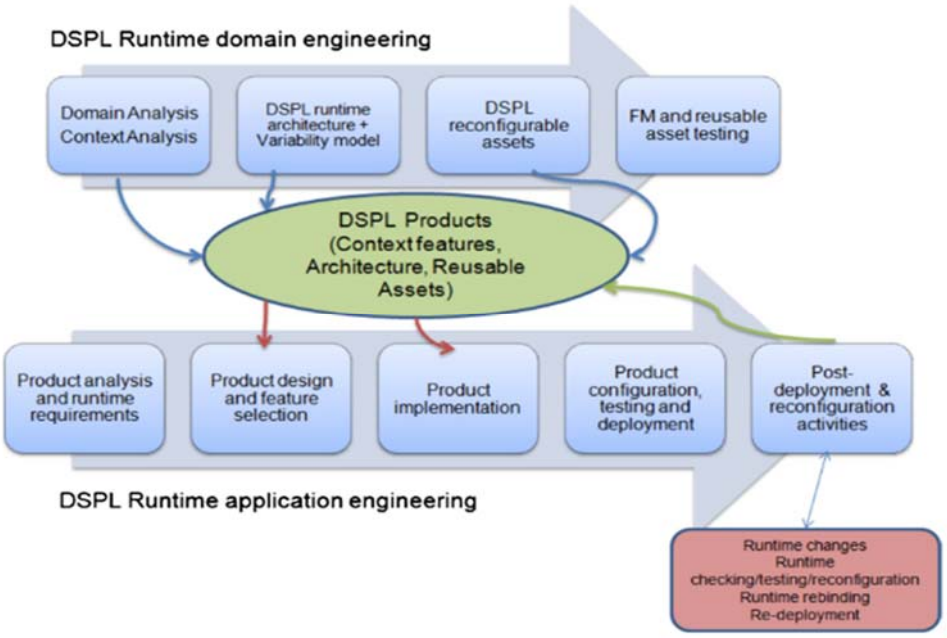



Figura 2.19. Ciclo de vida dual de una DSPL.

Como podemos observar en la Figura 2.19, en el ciclo de vida de **Ingeniería del Dominio** la fase tradicional de análisis de dominio (*domain analysis*) se ha extendido con una actividad de análisis de contexto (*context analysis*), donde se identifican aquellas características del sistema que dependen del contexto y que pueden modelarse de manera complementaria al resto de características. Posteriormente, se desarrolla una arquitectura para la DSPL en la que se tengan en cuenta que módulos o partes de la misma pueden modificarse en tiempo de ejecución y cuál va a ser el *binding time* de las variantes tanto de contexto como del sistema. En este modelo de la DSPL es posible que las variantes dispongan de más de un *binding time* y que se pueda saltar de uno a otro, al contrario que las SPL convencionales. Las principales diferencias con las SPL tradicionales residen en la fase de análisis y modelado del contexto **[DESM 2007] [ALI 2009],** que permite identificar y representar las características de contexto **[HART 2008] [CAPI 2014A]**, y un mecanismo de variabilidad dinámica **[CAPI 2012]** como aspecto central a las DSPL. Por ello, la asociación de características a contextos comienza a ser relevante para las DSPL, tal y como se describe en **[SALL 2013]**, donde los modelos de características aparecen enriquecidos con información de contexto para explicar los cambios en el entorno.

En la fase de **Ingeniería de Aplicaciones**, además de identificar requisitos del sistema para cambios en tiempo de ejecución, se añade una fase final complementaria a las SPL en las que se incluyen actividades en tiempo de ejecución y en fase de post-despliegue, de manera que los productos software pueden reconfigurarse una vez desplegados de forma manual o automática, bien dirigidos por cambios en el contexto o a petición del usuario. Motivado por cambios en el entorno, el sistema puede necesitar reconfigurarse y desplegarse varias veces con mínima intervención humana. Sin embargo, no es objeto de esta tesis discutir una aproximación para las DSPL sino centrarse en soluciones de variabilidad dinámica, como técnica principal de una DSPL.

**Dominios de aplicación de las DSPL**: Hoy en día existen numerosos sistemas donde es necesaria una reconfiguración del software o realizar modificaciones, básicas y complejas, en base a variaciones en el contexto. Aunque no significa que por existir una necesidad de cambios en el software o en las variantes en tiempo de ejecución, éste haya sido desarrollado utilizando un modelo de líneas de producto. Sin embargo, debido a la juventud de las DSPL, es difícil hoy en día encontrar ejemplos de las mismas en la industria. Por ello, describimos aquí algunos ejemplos breves de sistemas que necesitando reconfiguración dinámica podrían construirse bajo un modelo de una DSPL.

Los escenarios de casas inteligentes son uno de los dominios de aplicación típicos en los que se combina información de contexto procedente de múltiples dispositivos y reconfiguraciones no complejas provenientes del usuario y del entorno, tal y como se describe en **[CETI 2009A]**. En tiempo de ejecución, el uso de arquitecturas con soporte para *runtime* facilitan la reconfiguración dinámica de los componentes que pueden estar presentes en diferentes configuraciones, así como la activación y desactivación de características gracias al sistema denominado MoRE **[CETI 2013]** y de esta manera proporcionar capacidades autónomas a los sistemas. EL caso de estudio de un hotel inteligente (*Smart Home System – SHS*) descrito en **[CETI 2009B]** permite reconfigurar los servicios del hotel de acuerdo a cambios en el contexto y define 8 escenarios con diferentes configuraciones de características activas. Esta solución se puede integrar dentro de una DSPL con el fin de activar y desactivar componentes del SHS de manera dinámica mediante canales de comunicación que, también, se establecen de forma dinámica entre los componentes.

En la propuesta descrita en **[GAME 2011]**, se presenta el modelo Famiware para describir la variabilidad de una familia de sistemas de ambiente inteligente desarrollada con redes de sensores (*Wireless and Sensor Networks* – WSN) y teléfonos inteligentes. La propuesta utiliza las ideas de las DSPL para facilitar el proceso de adaptación en tiempo de ejecución permitiendo reconfigurar las aplicaciones Famiware cuando hay cambios en el contexto. Famiware permite reemplazar en tiempo de ejecución un modelo de características por otro. En **[GAME 2012]**, los autores proponen una extensión del modelo Famiware pero modelando las características de contexto con el fin de gestionar en tiempo de ejecución los cambios en el contexto en sistemas de inteligencia ambiental y conectadas dichas características a sensores. Para ello, los autores proponen modelar diferentes contextos para la diversidad de escenarios, así como proporcionar servicios dependientes de cada contexto.

En el trabajo descrito en **[PASC 2013]**, los autores describen una combinación de líneas de producto software dinámicas (DSPL) con computación autónoma que permite definir un servicio de monitorización de contexto y de reconfiguración dinámica de servicios que se integran en un *middleware* para sistemas adaptativos. La solución propuesta utiliza el lenguaje de variabilidad CVL con un sistema de reconfiguración en tiempo de ejecución que permite resolver modelos CVL de manera dinámica. Los autores se basan en el ciclo MAPE-K y presentan un algoritmo que optimiza el proceso de adaptación de aplicaciones móviles en función de los recursos utilizados y de la información de contexto.

Esfuerzos similares en el uso de un DSPL para configurar dispositivos IoT (*Internet of Things*) los podemos encontrar en **[PASC 2015] [AYAL 2016]**. En cuanto a las redes de sensores inalámbricos (WSAN), los trabajos de **[ORTI 2012] [MOUR 2013]** describen una aproximación de variabilidad dinámica para reconfigurar redes WSAN en tiempo de ejecución de manera que se puedan añadir o eliminar sensores de manera dinámica en el contexto de las DSPL. Sin embargo, los autores no proponen como validar las restricciones en tiempo de ejecución.

En un trabajo posterior **[MAUR 2018]**, los autores combinan el concepto de líneas de producto software (SPL) con información de contexto en el que los autores describen un motor de reconfiguración denominado HyVarRec para añadir o eliminar restricciones de manera incremental. La propuesta describe un ejemplo de sistemas de fabricación en la que se introduce un modelo de variabilidad híbrido (HyFM) que incluye características

de contexto. El motor de reconfiguración propuesto facilita la evolución de la SPL basándose en la información contextual. Por último, en un trabajo reciente **[MORA 2021]**, se describe una aproximación para capturar la variabilidad de contexto en redes neuronales IoT incluyendo su relación con el comportamiento y los atributos de calidad. Los autores modelan la variabilidad de contexto de dispositivos IoT y lo aplican a un caso de estudio del control inteligente de frutas para predecir comportamientos. Sin embargo, los autores no van más allá para realizar posibles cambios estructurales en los modelos de variabilidad.

## 2.5 Gestión de la Variabilidad Dinámica

La evolución de los modelos de variabilidad es una de las razones fundamentales para añadir, eliminar o modificar las características que se representan en forma de variantes y puntos de variación **[JI 2015]**, como por ejemplo, mover una característica de un lugar a otro.

### 2.5.1 Modificación de la variabilidad en tiempo de ejecución

Uno de los puntos fuertes de los modelos de variabilidad es modificar la estructura de dichos modelos en tiempo de ejecución, permitiendo que los sistemas tengan la capacidad de configurar y modificar sus variantes y puntos de variación con otra forma diferente a la inicial, o bien permitir que las variantes tomen valores diferentes una vez desplegado el sistema **[CAPI 2011] [BOSC 2012] [CAMA 2013]**. De esta manera, los modelos de variabilidad se convierten en extensibles y más abiertos. Según este esquema, podríamos caracterizar los cambios en la variabilidad estructural de diferentes formas yendo más allá de la simple activación y desactivación de las características de un producto durante el tiempo de ejecución. Por ejemplo, podría ser interesante añadir o eliminar ciertas opciones una vez desplegado el producto. En este sentido, destacamos las principales características que un modelo de variabilidad dinámica podría ofrecer.

**Añadir/Eliminar/Cambiar variantes**: Esta facilidad implica modificar la forma del árbol o modelo de variabilidad, al permitir las tres operaciones mencionadas sin intervención del diseñador o con mínima intervención, es decir, de una manera automática o semi-automática. EL problema de añadir variantes de esta forma es situar la nueva variante en el lugar correcto o más apropiado y comprobar previamente que no existen restricciones que impidan su inclusión. Si eliminamos una variante debemos eliminar las restricciones asociadas y dar lugar a eliminar los hijos si los tuviere, siempre y cuando no cause conflicto con otras variantes y generar un problema de recursividad. Por último, cambiar una variante puede implicar mover de sitio una característica, lo que implica eliminar y añadir la misma, pero en otra localización. Esta última operación puede ser compleja y requerir intervención del diseñador. Hasta ahora hay pocas experiencias que sugieran la modificación estructural de la variabilidad, como por ejemplo la de **[BOSC 2012]** donde los autores proponen una clasificación de las características basadas en lo que denominan supertipos para clasificar y modificar las variantes de acuerdo a un conjunto de supertipos definidos previamente para cada familia de productos. En una referencia posterior **[CAPI 2016]**, se describen distintos escenarios donde añadir una nueva *feature* en el modelo de variabilidad. Una propuesta similar para modificar la variabilidad estructural de sistemas Cloud **[BARE 2015]**, donde los autores describen una DSPL para la plataforma PaaS (*Platform-as-a-Service*) Heroku que permita añadir y eliminar características del modelo de variabilidad.

**Clonar características**: Otra forma de modificar la variabilidad estructural es clonar variantes, tal y como se menciona en **[GAME 2013]**. Una característica clonable utiliza modelos basados en cardinalidad para definir el número de veces que una *feature* puede clonarse y así soportar diferentes configuraciones del producto. Este proceso se podría realizar de forma recursiva y facilitar la reutilización de características. Básicamente, podríamos decir que consiste en añadir características ya existentes, pero de forma dinámica.

**Modificar puntos de variación**: Una extensión a la modificación de las variantes en tiempo de ejecución es modificar la fórmula booleana que los conecta. Sin embargo, automatizar este cambio es mucho más complejo ya que normalmente requiere intervención humana para definir la fórmula lógica que conecta las variantes mediante relaciones AND/OR/XOR.

Una primera aproximación es la descrita en **[COLA 2016]**, donde los autores sugieren el uso de modelos funcionales para simular arquitecturas ejecutables que pueden replicar el comportamiento de las relaciones AND/OR/XOR de los modelos FODA. Los autores describen una aproximación que utiliza diagramas de estado y diagramas de transición para modelar formalmente dichas relaciones y construyen un prototipo basado en técnicas de ingeniería dirigida por modelos para generar las relaciones típicas de los modelos de variabilidad mediante operadores que simulan las mismas. El prototipo mencionado permite generar de manera automática los diagramas de estado y transiciones de un sistema o una familia de productos. Sin embargo, la solución propuesta no permite modelar modelos de variabilidad ni restricciones a los de modelos de variabilidad, lo cual es un inconveniente a la hora de añadir características de manera dinámica.

Por otra parte, en un trabajo más reciente se describe una propuesta incipiente que permite almacenar dicha meta-información sobre puntos de variación en una base de datos **[CAPI 2016]** pero la solución aún no está implementada. En el caso de eliminar una variante simplemente habría que eliminarlo de la fórmula donde se encuentra el punto de variación.

***Binding time* dinámico y *Multi-binding*:** En las SPL convencionales, la mayoría de los *binding time* se definen antes del despliegue del producto y solamente se considera un único tiempo para resolver las variantes. En cambio, las DSPL permiten definir tiempos de ligadura de las variantes posterior al despliegue del producto y pudiendo reconfigurarse de nuevo dichas variantes en tiempo de ejecución. Es lo que se conoce como *binding time* dinámico. Además, debido a la necesidad de ciertos sistemas de cambiar de un estado a otro (e.g. de modo de ejecución a modo testing), sobre todo aquellos más críticos o con capacidades de tiempo real, es posible permitir la transición de un modo de *binding time* a otro. De esta manera, las variantes se podrían configurar de una manera para un determinado *binding time* y de otra manera en otro diferente. Por ejemplo, una variante puede configurarse en tiempo de post despliegue (*binding time* de configuración) manualmente por el usuario (e.g. activar la opción de *bluetooth*), por el sistema automáticamente (e.g. actualizar el reloj de un dispositivo móvil basado en un cambio de localización geográfica), mientras que en un *binding time* en modo testing las variantes se configuran mediante una carga dinámica de ficheros según el modo de testeo en el que entre el software.

Por ello, en la propuesta de **[BOSC 2012]**, se detallan diferentes modos de *binding time* y *multi-binding* así como posibles transiciones entre los mismos, tal y como se muestra en la Figura 2.20.

| Binding time | Static/dynamic binding | Configurability | Single/multiple binding times | Binding on the developer/customer side | Transition for multiple binding times |
|---|---|---|---|---|---|
| Design | S | N/A | SI | D | N/A |
| Compilation/link | S | Low | SI | D | N/A |
| Build/assembly | S | Low | SI | D | N/A |
| Programming | S | Medium | SI | D | N/A |
| Configuration (Cf) | S/D | Medium/high | SI/MU | D/C | Cf→Dpl<br>Cf→RT |
| Deploy (Dpl) and redeploy (Rdpl) | S/D | Medium/high | SI/MU | D/C | Dpl→Rdpl<br>Rdpl→Cf |
| Runtime (RT) (start-up) | D | High | SI/MU | C | Dpl→RT<br>Rdpl→RT<br>RT→Rdpl<br>RT→Cf |
| Pure runtime (PRT) (operational mode) | D | Very high | SI/MU | C | RT→PRT<br>PRT→Cf<br>Cf→PRT<br>PRT→PRT |

Figura 2.20. Diferentes *binding times* para soluciones de variabilidad dinámica.

La propuesta de transiciones entre *binding times* descrita en la Figura 2.21, se describe con más precisión en **[CAPI 2018]** donde se sugieren transiciones entre diferentes estados de un sistema, de manera que las variantes podrían tomar valores diferentes según el estado de un sistema (e.g. configuración o testing).

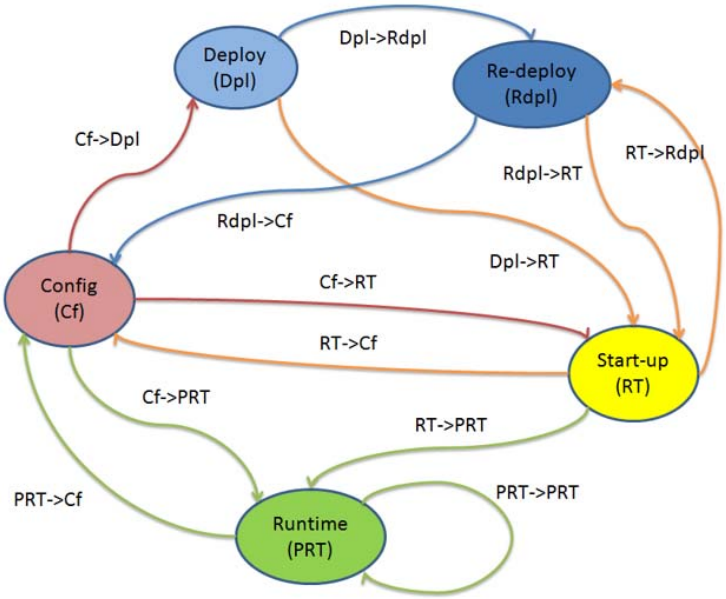



Figura 2.21. Transiciones entre posibles *binding times*.

De esta manera, las transiciones descritas en la Figura 2.21 se reflejan ofreciendo distintas transiciones entre diferentes modos operacionales de un sistema donde la configuración de las variantes puede asumir valores para diferentes *binding times*.

Esta idea es aplicable en sistemas críticos tal y como se menciona en **[CAPI 2016A]** donde dos posibles *binding times* se proponen para generar configuraciones en una SPL de Toshiba para la creación de plantas de energía (EPG).

Además, en la propuesta de **[TAIN 2015]** se describe un mecanismo para implementar *binding* dinámico con el fin de soportar la adaptación y evolución de sistemas que encapsulan comportamiento dinámico y permitir variaciones en tiempo de ejecución anticipándose así al proceso de adaptación. Los autores proponen un mecanismo denominado LyRT, que permite la activación y desactivación del comportamiento dinámico de los sistemas y la instanciación de dichos comportamientos en tiempo de ejecución. Sin embargo, la propuesta queda limitada a una activación y desactivación de comportamientos en tiempo de ejecución, similar a los lenguajes COP.

### 2.5.2 Restricciones en tiempo de ejecución

La gestión de las restricciones entre características de manera dinámica o en tiempo de ejecución es aún un campo virgen en el que existen pocas propuestas. Tradicionalmente, las restricciones de tipo *require* y *exclude* se validan mediante un resolvedor de restricciones que comprueba la validez de un modelo de variabilidad y permite ayudar a configurar un determinado producto. Una de las primeras propuestas que describe un sistema de restricciones en tiempo de ejecución para una línea de productos software dinámica (DSPL) es **[ROSE 2011]**. Los autores proporcionan un mecanismo de variabilidad dinámica y soportan la auto-configuración de la DSPL basándose en un mecanismo de adaptación que permite reducir el esfuerzo para calcular la configuración más óptima en diferentes escenarios. Además, facilitan la integración entre *binding times* estáticos y dinámicos. La solución propuesta, denominada FeatureAce, aporta: (i) reutilización de fragmentos reusables para una DSPL de manera estática, (ii) un mecanismo de adaptación basado en reglas, y (iii) autoconfiguración para componer modelos de características.

En este sentido, las unidades se componen en la DSPL en tiempo de ejecución permitiendo añadir o eliminar unidades (i.e. grupos de características). Un proceso de validación basado en un SAT solver comprueba la validez de la unión antes de componer las características. Los autores no describen de manera explícita qué SAT solver utilizan y, por otra parte, el algoritmo propuesto sólo añade o elimina restricciones de un modelo de variabilidad dado del que se conocen todas las características de antemano, pero no detalla si dispone de capacidades para modificar características de manera dinámica.

Entre las propuestas que discuten cómo verificar restricciones tenemos el trabajo de **[SANT 2016]**, donde se propone una forma para modelar y verificar la evolución y el comportamiento adaptativo de una DSPL mediante reglas de adaptación para activar y desactivar características de forma dinámica y verificar así las características de contexto. En una propuesta similar, **[SOUS 2017]** describe el uso de restricciones temporales para modelar la reconfiguración de una DSPL en sistemas Cloud. Estas restricciones extienden los modelos de variabilidad tradicionales para permitir múltiples alternativas de reconfiguración, añadiendo restricciones que posibilitan la inclusión de aspectos temporales y controlan mejor el comportamiento adaptativo del software.

Más recientemente, en **[FELF 2018]** se describe un modelo denominado FLEXDIAG para diagnosticar la reconfiguración de un sistema en tiempo de ejecución o en cualquier tiempo (*anytime*). Los autores evalúan la prestación del algoritmo propuesto y su calidad para diagnosticar en términos de precisión y variando el valor de un parámetro. Los modelos de configuración son modelos de variabilidad que incluyen restricciones estructurales en notación CSP. La solución propuesta se integra en la herramienta FaMa para proporcionar una gestión integrada de los procesos de diagnosis. Los autores utilizan modelos grandes de variabilidad (hasta 20.000 características y entre un 10% y 30% de restricciones) con el fin de comprobar las prestaciones del algoritmo que resultan ser mejores que otras propuestas basadas en algoritmos evolutivos.

Otros autores como **[MAUR 2021]** sugieren la detección de posibles anomalías en los modelos de características de contexto y proponen utilizar fórmulas booleanas cuantificadas (QBF) para que los resolvedores de características detecten dichas anomalías. Los autores extienden un motor denominado HyVarRec para mejorar la detección de dichas anomalías. En la propuesta, extienden un motor de reconfiguración HyVarRec para presentar evidencias de la superioridad de los resolvedores QBF sobre SAT.

En el trabajo de **[SANO 2021]** los autores discuten una aproximación denominada RETAKE (*Runtime Testing of Dynamically Adaptive Systems*) y orientada al testing en tiempo de ejecución en sistemas adaptativos con el fin de verificar la corrección de propiedades de comportamiento. Los autores identifican variabilidad de contexto (eCFM – *extended Context Feature Model*) y modelan las relaciones entre características de contexto y las del sistema, que luego utilizan para detectar cambios en el contexto en sistemas adaptativos.

La propuesta de testing en tiempo de ejecución (*runtime*) consiste en inyectar datos al sistema para guiar el proceso de adaptación y de esta manera evaluar los resultados producidos. Los autores han desarrollado una herramienta para realizar la actividad de testing en *runtime* simulando tests cambiantes y estados de contexto aleatorios. Sin embargo, la propuesta no considera cambios en los modelos de variabilidad de contexto, sino únicamente en los valores de las variables de contexto.

### 2.5.3 Soluciones y experiencias en variabilidad dinámica

Las soluciones propuestas hasta ahora para implementar la variabilidad dinámica son pocas, así como las experiencias. A pesar de ello, existen algunas propuestas incipientes para implementar dicha aproximación tal y como describimos a continuación. Algunas de las primeras experiencias las podemos encontrar en **[HELL 2009] [FROS 2009]**, donde un mecanismo de variabilidad dinámica se utiliza para activar y desactivar características en vehículos guiados autónomos (AGVs) en entornos industriales y la posibilidad de añadir y eliminar variantes con el fin de modelar líneas de producto software dinámicas. En **[INGL 2013]** se describe un lenguaje declarativo específico de dominio (VML – *Variability Modeling Langua*ge) para representar variabilidad en tiempo de ejecución centrada en aspectos de seguridad y eficiencia en robots y que incluye características de contexto representadas por variables de contexto. La propuesta de los autores permite el *binding* dinámico de los puntos de variación, pero no la modificación estructural de la variabilidad. Sin embargo, en **[BARE 2015]**, los autores describen una DSPL para la plataforma Cloud PaaS (*Platform-as-a-Service*) Heroku con una serie de reglas que posibilitan añadir y eliminar variantes en tiempo de ejecución.

Por otra parte, en **[MURG 2015]**, los autores describen un *framework* para modelar y gestionar la variabilidad de los procesos mediante la composición de un modelo base, fragmentos reutilizables de procesos y modelos de variabilidad, con el fin de retrasar el *binding time* de las variantes a tiempo de ejecución. De esta manera, es posible reducir el esfuerzo de modelado de las variantes en flujos de trabajo (*workflow*s) inteligentes, lo cual resulta muy útil en mercados dinámicos. Los autores proponen el modelo LateVa como una combinación de flujos inteligentes y DSPLs para proporcionar capacidades de variabilidad dinámica en notación BPMN (*Business, Process, Model and Notation*). También proponen el uso de *workflows* dependientes del contexto para modificar de forma dinámica el modelo de proceso y poder resolver la configuración de las variantes en tiempo de ejecución. Finalmente, utilizan al menos dos *binding times* (*start-up* y *pure runtime*) descritos en **[BOSC 2012]** para automatizar los inventarios en almacenes de manera inteligente.

En la propuesta descrita en **[PINT 2015]**, se propone una extensión al modelo Famiware descrito anteriormente para proporcionar capacidades de reconfiguración dinámica de políticas de seguridad en redes WSAN. Para ello, los autores sugieren una extensión del modelo de variabilidad de *Famiware* que combina un *framework* de seguridad denominado INTER-TRUST para negociar de forma dinámica el despliegue de políticas de seguridad y así, el middleware *Famiware* (basado en el concepto de las DSPL) utiliza las características de contexto y permite reconfigurar de manera automática las instancias para las redes WSAN. Los autores validan su propuesta en el dominio de los sistemas inteligentes de transporte, proporcionando así diferentes configuraciones en base a diferentes condiciones de contexto. Sin embargo, no describen la necesidad de modificar la variabilidad estructural de manera dinámica.

Otros autores como **[CARD 2016]** sugieren diferentes estrategias para transformar la variabilidad en tiempo de ejecución implementada sobre el lenguaje CVL, de manera que pueden reemplazarse fragmentos de los elementos del modelo, o bien añadir características de manera incremental y después reconfigurar los modelos, o sintetizar dichos modelos en tiempo de ejecución.

El campo de la robótica es sumamente adecuado para implementar soluciones de variabilidad dinámica con características de contexto. En una propuesta reciente **[GARC 2019]**, los autores sugieren que la variabilidad dinámica es un desafío abierto y proponen su aplicación para gestionar la configuración del robot mediante la carga de diferentes

ficheros de configuración, así como reglas para mejorar la adaptación del robot e incrementar su inteligencia frente a diversas situaciones. Sin embargo, los autores no proponen soluciones concretas para implementar dicha variabilidad.

Por otra parte, en **[KRIE 2019]** los autores describen una aproximación de gestión dinámica de la variabilidad para una DSPL que permite implementar sistemas adaptativos para el software de SGX (*Intel's Software Guard eXtensions*). La solución propuesta permite configurar el código binario que puede ser actualizado en tiempo de ejecución. Los autores describen un prototipo que permite cargar y descargar (añadir y eliminar) características de manera dinámica y satisfacer las dependencias entre ellas. De esta manera, las características opcionales se pueden añadir en tiempo de ejecución mediante un gestor de variabilidad dinámica y así poder evaluar el rendimiento cuando los recursos de hardware son limitados.

En la propuesta descrita en **[QUIN 2020]** y basado en **[QUIN 2015]**, se describe cómo evolucionar una DSPL y se presenta una arquitectura de referencia para sistemas ciber-físicos que utiliza una infraestructura combinada con variabilidad dinámica orientada a monitorizar los cambios durante la ejecución de los sistemas. Los autores proponen un modelo intermedio para asociar los cambios en los modelos de variabilidad desde el espacio del problema al espacio de la solución, con el fin de poner añadir, eliminar y actualizar características en un modelo de variabilidad. Así mismo, proponen reglas de adaptación para comprobar que no se viola ninguna restricción en los procesos de reconfiguración.

Sin embargo, los autores no discuten si las posibles localizaciones donde incluir las características nuevas se resuelven de manera automática, ni si es posible añadir restricciones en tiempo de ejecución asociadas a dichas características. La propuesta permite la intervención humana en aquellos casos donde la inclusión automática de características no sea posible pero no aclara en qué casos concretos no es posible una adición automática, ya que depende de las reglas de adaptación existentes. Los resultados que se muestran parecen bastante satisfactorios en cuanto a prestaciones y eficacia.

Por último, en un trabajo reciente **[ROME 2022]**, los autores sugieren un modelo de variabilidad dinámica en el campo de la robótica que permite añadir variantes y valores en tiempo de ejecución. Estas variantes se pueden incluir en el modelo de variabilidad de un robot y estimar las propiedades no funcionales de calidad de servicio (Quality of Service – QoS) durante los procesos de auto adaptación con el fin de encontrar la configuración más adecuada. La solución propuesta permite añadir variabilidad en árboles de comportamiento del robot.

Sin embargo, la propuesta no indica si es posible modificar las fórmulas lógicas que conectan las variantes de los puntos de variación, ni cuál sería la mejor localización de las nuevas variantes en el árbol de variabilidad. Esta propuesta se aproxima bastante a la solución que vamos a proponer en esta tesis pero en este trabajo se incluyen otros aspectos no considerados en el trabajo mencionado.

## 2.6 Conclusiones

En este capítulo del estado del arte hemos descrito los orígenes y utilidad de los sistemas que utilizan variabilidad software convencional para poder gestionar las diferentes opciones y características visibles de los sistemas software. Además, hemos descrito las diferentes formas de representar y gestionar esta variabilidad con sus restricciones y discutido los problemas de escalabilidad y visualizar grandes modelos de variabilidad. De manera complementaria, la propiedad del *binding time* donde las variantes se traducen a valores concretos para configurar un sistema software determinado, resulta clave para poder ofrecer una gama de *binding times* una vez desplegado el sistema, y las posibles transiciones entre ellos como solución para cambiar entre diferentes modos operacionales de un sistema.

Así mismo, hemos destacado la importancia de la identificación y modelado de la variabilidad de contexto y reflejado la importancia que tiene en los sistemas auto-adaptativos e inteligentes para posibilitar la reconfiguración de los mismos. Esta reconfiguración dinámica está parcialmente soportada por los modelos basados en MAPE-K y modernizados por las aproximaciones de líneas de producto software dinámicas, donde se describen soluciones parciales para la gestión dinámica de la variabilidad como mecanismo central para reconfigurar las variantes una vez desplegado el sistema.

Actualmente, la gestión dinámica de las variantes no tiene soporte por parte de las herramientas actuales de gestión de variabilidad software ni por las de líneas de productos software, principalmente debido a la carencia por parte de la industria de este tipo de soluciones. Sin embargo, debido a la fuerte demanda y despliegue de sistemas basados en el contexto y de sistemas inteligentes que demandan cambios o actualizaciones y soporte de nuevas características, una vez desplegado el sistema y durante la ejecución del mismo, resulta cada vez más necesario proporcionar soluciones de variabilidad dinámica que permitan la inclusión de nuevas características de manera automática y poder reconfigurar un sistema software complejo con mínima intervención humana.

Como conclusiones principales, podemos destacar que no existe una solución completa de variabilidad dinámica que permita añadir, eliminar y modificar variantes en tiempo de ejecución ni tampoco de los puntos de variación.

Actualmente, existen propuestas parciales, pero no completas. Además, tampoco existen soluciones para soportar múltiples *binding times* y las transiciones entre ellos. Solamente existen propuestas limitadas y con validación limitada. Por estos motivos, podemos afirmar que las soluciones de variabilidad dinámica o en tiempo de ejecución son parciales y limitadas y se hace necesario aportar soluciones más integradas.

Finalmente, no existen soluciones que comprueben las restricciones en los modelos de variabilidad en tiempo de ejecución, lo que resulta clave para añadir o eliminar características en tiempo de ejecución y sin intervención humana. Solamente en una de las propuestas se sugiere utilizar un SAT solver pero no se describe su integración con algoritmos de variabilidad dinámica.

# Capítulo 3. Planteamiento del Problema e Hipótesis

Tal y como se ha comentado a lo largo del capítulo del estado del arte, actualmente no hay un mecanismo definitivo que permita añadir o eliminar características en tiempo de ejecución y a ser posible sin intervención humana o con mínima intervención. En este sentido, tampoco existen mecanismos que permitan la gestión automática o semi automática de los puntos de variación debido a que la definición de la fórmula lógica que conecta las variantes es hasta ahora una tarea manual y humana. Además, en los sistemas dependientes de contexto, es necesario proporcionar modelos de representación más adecuadas para las características de contexto de un sistema y para las de no contexto, indicando como ambos tipos definen sus relaciones.

Por otra parte, los sistemas que comprueban restricciones en los modelos de variabilidad resuelven éstas de manera *off-line* cuando una característica se modifica en el árbol de características de manera manual. Por ello, resulta necesario identificar en las soluciones de variabilidad dinámica una forma que permita comprobar las restricciones en tiempo de ejecución sin que éstas se tengan que resolver de manera separada en el tiempo cuando un modelo de variabilidad se modifica dinámicamente.

Así mismo y de manera complementaria, las propuestas actuales de líneas de productos dinámicas y de soluciones de variabilidad en tiempo de ejecución, necesitan de mecanismos más adecuados para soportar múltiples *binding times* en las variantes y un dispositivo adecuado que permita la transición entre diferentes modos operacionales de un sistema, de manera que las variantes puedan tomar valores en diferentes momentos.

Por lo anterior, el problema que vamos a tratar de resolver en esta tesis es el siguiente:

***Se pretende proporcionar una solución de variabilidad dinámica que resuelva la modificación estructural de la variabilidad en tiempo de ejecución.***

En base a este problema general, vamos a descomponer en subproblemas el objetivo general, tal y como describimos a continuación.

**Subproblema 1**: Proporcionar un modelo para modificar la variabilidad estructural en tiempo de ejecución.

**Subproblema 2**: Proporcionar un mecanismo que sea capaz de clasificar una característica en el lugar más adecuado en un árbol de variabilidad cuando existan diferentes posibilidades.

**Subproblema 3**: Proporcionar un mecanismo que permita eliminar una característica en un modelo de variabilidad y el tratamiento de sus restricciones asociadas.

**Subproblema 4**: Proporcionar una solución que permita comprobar las restricciones entre características de forma automática y en tiempo de ejecución.

**Subproblema 5**: Proponer una solución para semi automatizar el tratamiento de los puntos de variación.

Con el fin de abordar los diferentes subproblemas vamos a definir las siguientes hipótesis de trabajo que van a delimitar el ámbito de la solución propuesta.

**Hipótesis 1**: No vamos a considerar el movimiento de características entre ramas diferentes en un árbol de variabilidad.

**Hipótesis 2**: No vamos a considerar múltiples *binding times* ni transiciones entre los mismos, pero si se va a tomar el *runtime binding time* como tiempo de referencia en el que las variantes pueden añadirse o asumir sus valores.

**Hipótesis 3**: No se requiere una completa automatización en el proceso de tratar la inclusión de puntos de variación, ya que en algún momento es posible que se necesite cierta intervención humana.

En base a los problemas, subproblemas e hipótesis de trabajo descritos anteriormente vamos a proponer en el siguiente capítulo la solución a los problemas planteados.

# Capítulo 4. Solución Propuesta

En este capítulo vamos a guiar al a lector a través de los diferentes pasos acerca de la solución propuesta y como planteamos la resolución de los desafíos de investigación descritos en el capítulo anterior.

## 4.1 Modelo para Gestionar la Variabilidad dinámica

Esta sección abordará el tratamiento al **subproblema 1** consistente en proporcionar un modelo para modificar la variabilidad estructural en tiempo de ejecución. Para ello nos vamos a inspirar en las ideas descritas en **[BOSC 2012]** y su modelo de supertipos ya discutido en el capítulo del estado del arte. En este sentido, utilizamos el modelo de supertipos el cual vamos a integrar con un algoritmo de variabilidad dinámica que se describirá en la sección 4.2.

Teniendo en cuenta las hipótesis de trabajo 3 y 4, nuestra solución se va a centrar en el *binding time* en ejecución, pero sin realizar transiciones a otros *binding times* o modos operacionales de un sistema. Además, dado que nuestra solución no sugiere una completa automatización del proceso de modificación de las variantes, es posible que en algún momento del proceso sea necesaria la intervención humana, como por ejemplo en aquellos casos en que el modelo de supertipos no sea capaz de decidir en qué lugar del árbol de variabilidad resulte mejor colocar una característica.

### 4.1.1 Solución genérica para gestión de variabilidad dinámica

En este apartado describimos la solución general que proponemos y que consta de las siguientes partes:

1. Un algoritmo que utilice el modelo de supertipos y que sea capaz de añadir una característica o nueva variante en el lugar más adecuado. Más adelante trataremos el caso de eliminación de características. Los diferentes supertipos deberán estar definidos previamente por el ingeniero software en función del problema y del dominio de aplicación.

2. Una base de datos de repositorio de supertipos y características correspondientes a un modelo de variabilidad. Las características deberán tener su supertipo asociado o posibilidad de ser precargado. Esta asociación suele realizarse de forma manual al poblar la base de datos.
3. Una herramienta o resolvedor de restricciones al cual el algoritmo de variabilidad dinámica pueda invocar previamente al proceso de añadido o eliminación de características y permita determinar si la operación es posible. Esta parte de la solución va a requerir de una integración de los formatos o modelo de datos entre el algoritmo de variabilidad dinámica y el resolvedor de restricciones correspondiente. De esta manera, tratamos de evitar la multiplicidad de soluciones existentes creando un resolvedor de restricciones propio.

Los tres puntos anteriores que corresponden al modelo de solución general propuesto se muestran en la Figura 4.1.

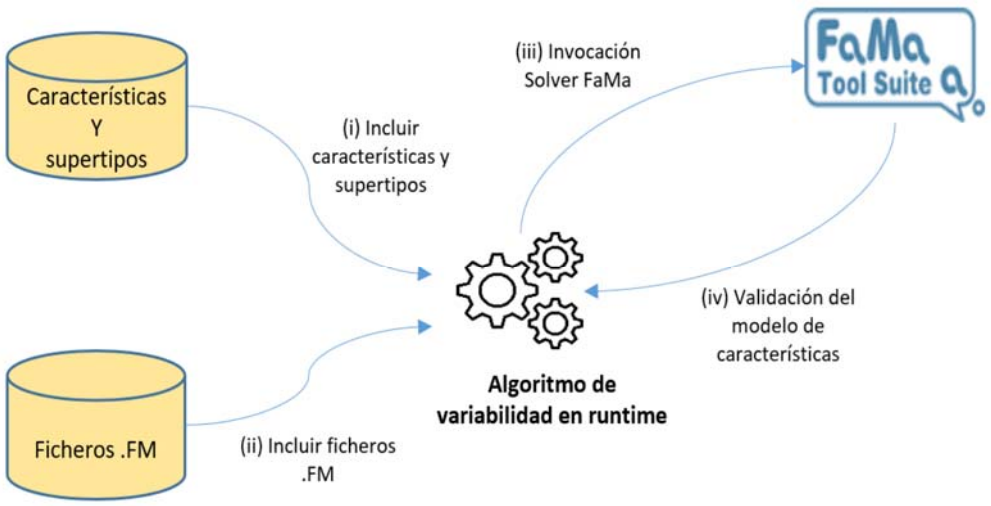



Figura 4.1. Esquema general de solución propuesta para gestionar la variabilidad dinámica basada en un modelo de supertipos.

El proceso por el cual se realiza la solución anterior consta de los siguientes pasos:

(i) El ingeniero software almacena o precarga si fuera posible un árbol de variabilidad y una lista asociada de supertipos, descrita en un formato de datos apropiado para el algoritmo de variabilidad dinámica y el resolvedor de restricciones.

(ii) Ante la necesidad de incluir una nueva característica (o eliminar una existente), bien de forma manual o bien automáticamente generada por una condición de contexto, la solución automática que se propone toma como entrada la petición y el modelo de variabilidad existente con el fin de determinar la localización óptima de la nueva característica. En la sección 4.2 se describirán las distintas posibilidades para añadir una nueva característica y si el algoritmo es capaz de optimizar la inclusión de diferentes características.

(iii) El algoritmo de variabilidad dinámica invocará a un resolvedor de restricciones para comprobar si las restricciones de la nueva característica son compatibles con las restricciones existentes y poder decidir si es posible o no añadir dicha característica.

Con el fin de comprender mejor la arquitectura software de la solución, describimos en la Figura 4.2 los diferentes elementos funcionales de la solución.

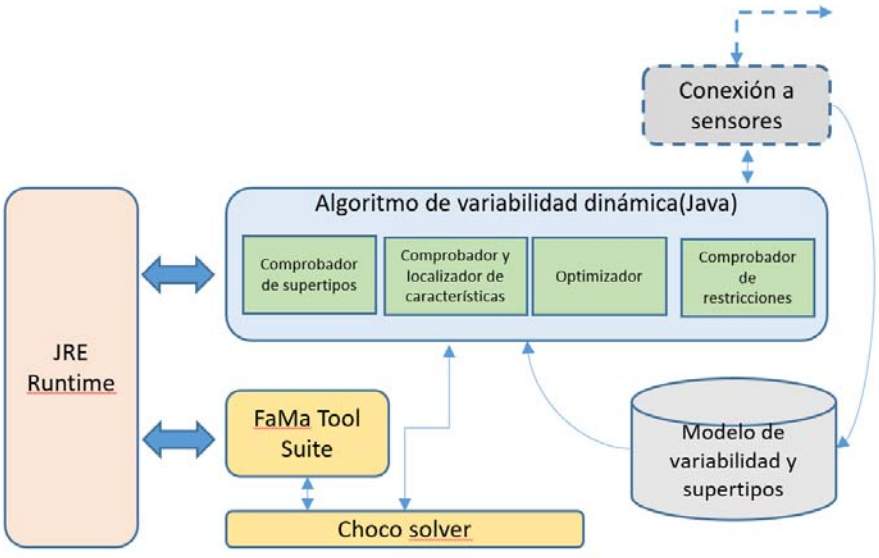



Figura 4.2. Arquitectura software de la solución para gestionar la variabilidad dinámica.

Los pasos principales de nuestra solución son los siguientes:

1. En primer lugar, dispondremos de un ***repositorio*** para almacenar los modelos de variabilidad y un conjunto de supertipos predefinidos para cada modelo de variabilidad.
2. El ***comprobador de supertipos*** comprueba que los supertipos de una nueva característica (caso de añadir una) coinciden con alguno de los supertipos previamente definidos.
3. El ***comprobador y localizador de características*** determina una o varias localizaciones posibles donde añadir la característica nueva a incluir. Para ello, es necesario comparar los supertipos de la característica a incluir con los supertipos de las características situadas en diferentes ramas.
4. El ***optimizador*** determina la mejor localización posible con el fin de no situar la nueva característica en la misma rama siempre, en caso de que pueda darse esta situación y con el fin de no sobrecargar siempre una misma rama del árbol de variabilidad.
5. El ***comprobador de restricciones*** dinámico invoca al resolvedor Choco para comprobar si las restricciones asociadas a la nueva característica son compatibles con la lista de restricciones existentes en el modelo.

La conexión a un posible sistema externo en el que se puedan detectar cambios en el contexto asociadas a características de contexto para motivar un cambio en la variabilidad estructural se puede implementar mediante el módulo ***conexión a sensores*** mostrado en la Figura 4.2 y que actualizaría la base de datos en el caso de que hubiera una nueva característica que se hubiera añadido o incluso una característica que se activara o desactivara.

La solución propuesta utiliza el resolvedor de restricciones Choco[5] que es el que utiliza la herramienta de gestión de variabilidad FaMa[6] (denominada FaMa Tool Suite). Hemos adoptado esta integración con el fin de no duplicar esfuerzos creando un resolvedor de restricciones nuevo y facilitar la interoperabilidad con otras herramientas existentes y soportadas en la actualidad como es FaMa y teniendo en cuenta que al ser

[5] https://www.cosling.com/choco-solver
[6] http://www.isa.us.es/fama/?FaMa_Current_Projects

una herramienta *research,* facilita mejor la integración de nuestra solución que si se tratara de una herramienta comercial.

### 4.1.2 Descripción del modelo de supertipos

Como comentamos al inicio de este capítulo, nuestra solución va a incluir el modelo de supertipos definido en **[BOSC 2012]** para clasificar y agrupar características similares en cuanto a funcionalidad. Los supertipos describen un sistema de clasificación superior de características y en nuestra tesis cumplirán con los siguientes requisitos:

1. Una característica podrá pertenecer a uno o varios supertipos.
2. Podrán existir características sin supertipos asociados.
3. Si una característica dispone de más de un supertipo, estos aparecerán ordenados mostrando en primer lugar los supertipos más relevantes. Este requisito define de manera implícita una prioridad en los supertipos no contemplada en **[BOSC 2012].**
4. Podrán existir supertipos compatibles definidos por el ingeniero software.

El formato de almacenamiento que definiremos para los supertipos será en forma de una lista ordenada en la que los supertipos más importantes aparecerán en primer lugar. A modo de ejemplo, si utilizamos el acrónimo ST para un supertipos, un ejemplo de lista de supertipos para una característica cualquiera $F_x$ sería de la siguiente manera:

$$ST_{(Fx)} = \{ST_1, ST_2, ST_3)$$

Siendo de manera que $ST_1$, $ST_2$ y $ST_3$ son tres supertipos asociados a $F_x$, siendo $ST_1$ el más relevante y $ST_3$ el menos relevante. De esta forma, los supertipos asociados a cualquier característica del árbol aparecen priorizados. Cada supertipo individual se define como un *string* en la que se indica el nombre del supertipo (e.g. *Multimedia*, *Seguridad*, etc.).

Asimismo, tal como se indica en **[BOSC 2012]**, contemplamos que existan supertipos compatibles y que se definen de la siguiente manera:

**Comp ($ST_A$, $ST_B$) = 1**, si $ST_A$ es igual a $ST_B$; o si $ST_A$ y $ST_B$ pertenecen a una lista de supertipos compatibles

**Comp ($ST_A$, $ST_B$) = 0**, en otro caso

Por último, y esta parte es contribución de esta tesis, asumimos que los hijos de una característica $F_X$ tienen los mismos supertipos que su padre. De esta manera, a la hora de incluir una nueva característica en una rama del árbol de variabilidad no es necesario comparar los supertipos de los variantes hijos. Esta decisión de diseño simplifica los casos en los que un hijo pudiera tener un supertipo diferente al padre y facilita la agrupación de características en unidades funcionales.

Una vez descrita la arquitectura general del algoritmo de variabilidad dinámica y el trasfondo teórico basado en una comparación de supertipos, en el apartado 4.2, pasamos a describir la parte de la solución propuesta, consistente en un algoritmo para gestionar la variabilidad de forma dinámica.

## 4.2 Algoritmo de Variabilidad Dinámica

Esta sección abordará el tratamiento al **subproblema 2** consistente en proporcionar un mecanismo que sea capaz de clasificar una característica en el lugar más adecuado en un árbol de variabilidad cuando existan diferentes posibilidades. Para ello, se necesita elaborar un modelo que se encargue de gestionar las variantes que intervienen en un sistema y al mismo tiempo sepa conjugar el modelo de supertipos, ya explicado en el punto anterior, y que en todo momento se guarde la compatibilidad con la base de restricciones bajo la que impera el modelo de características que rige el sistema.

Para ello vamos a tener muy en cuenta las hipótesis 1 y 2, y por ello esta solución va a centrar exclusivamente en la operación de añadir una característica a un árbol de variabilidad, sin entrar a considerar la eliminación de ésta o su localización a otra rama del árbol.

El desarrollo de un algoritmo como éste precisa disponer de los siguientes elementos y funcionalidades:

1) Un repositorio de datos donde estén almacenados las diferentes bases de información con todo lo relativo a los árboles de variabilidad, con sus variantes y puntos de variación. Así mismo, este repositorio deberá contener información sobre la lista de supertipos definidos previamente para cada variante y deberá ser en algún formato común o exportable para la herramienta FaMa.
2) El módulo principal del algoritmo tiene como misión obtener la información de un árbol de características y crear una representación del mismo dentro del sistema de manera que sea posible acceder y evaluar cada nodo por separado y evaluar sus relaciones tanto verticales (padres a hijos) como horizontales (entre nodos hermanos) y de esta manera poder elaborar diferentes estrategias a la hora de agrandar el árbol. En cada cambio que experimente el árbol, siempre será necesario la actuación de cuatro módulos que trabajarán de forma sincronizada para garantizar que la variabilidad de los árboles resultantes se mantiene y es coherente con los límites impuestos al modelo de características.

Estos módulos son los siguientes:

a) Un **localizador de características** para buscar las diferentes posibilidades donde colocar una nueva característica en el árbol de variabilidad. Este módulo analiza los nodos que componen el árbol y evalúa sus relaciones verticales y horizontales, así como el índice de compatibilidad otorgado por la comparación de los supertipos. De esta manera podemos generar una lista de los posibles lugares donde es factible añadir un nuevo elemento en base a la afinidad de sus supertipos.
b) Un **comprobador de supertipos**, que realizará una valoración de compatibilidad entre los supertipos afines a las variantes presentes y los de la característica a añadir.
c) Un **resolvedor de restricciones** que examinará la compatibilidad de las restricciones de la nueva característica con las ya existentes, con el fin de determinar si existe alguna incompatibilidad en el nuevo conjunto de restricciones que hiciera imposible añadir una característica en el modelo de variabilidad. Para esta funcionalidad utilizaremos el resolvedor empleado por FaMa e integraremos nuestra solución con dicho resolvedor.

En el caso de borrado de restricciones es necesario comprobar que no eliminamos restricciones que deban permanecer en el modelo.

d) **Optimizador** para determinar la mejor localización a la hora de incluir características.
e) **Eliminador de características** ya existentes
f) Un **comprobador de puntos de variación,** que permita modelar de manera automática, o semi-automática cuando sea posible, el nuevo punto de variación de una característica a incluir.

Una vez descritas las diferentes partes que integran el algoritmo de variabilidad, queda por definir la casuística que puede aparecer cuando disponemos de diferentes localizaciones para añadir una nueva característica, tal y como describimos en los siguientes apartados.

### 4.2.1 Caso 1. Ningún supertipo coincidente

El escenario más común a la hora de añadir una característica a un modelo es que la nueva característica no disponga de ningún supertipo compatible con los existentes. Es decir, entendemos que la nueva característica debe formar parte del modelo de variabilidad, pero puede ser que no se haya definido ningún supertipo para ella y que por tanto no pueda ser clasificada en alguna de las ramas del árbol de variabilidad. Este caso podía corresponder a una característica tanto obligatoria como opcional pero que se encuentra aislada de otras funcionalidades existentes. En este caso, el algoritmo sólo dispone de una posibilidad que consiste en añadirla al primer nivel del árbol de variabilidad con una relación de tipo AND. Si esta característica incluye algún supertipo y no corresponde con los ya definidos, entonces el supertipo nuevo se añade también al repositorio.

**Escenario 1**: La Figura 4.3 muestra el resultado de un ejemplo sencillo que no corresponde a ningún modelo de variabilidad en concreto y en el que podemos observar en la parte izquierda de la figura un árbol de variabilidad formado por 4 características. En la parte de derecha de la figura, la nueva característica $F_x$ se añade a la raíz del árbol de variabilidad como una característica del primer nivel. En este ejemplo no hemos anotado $F_x$ como característica obligatoria u opcional ni tampoco hemos definido ningún nuevo supertipo.

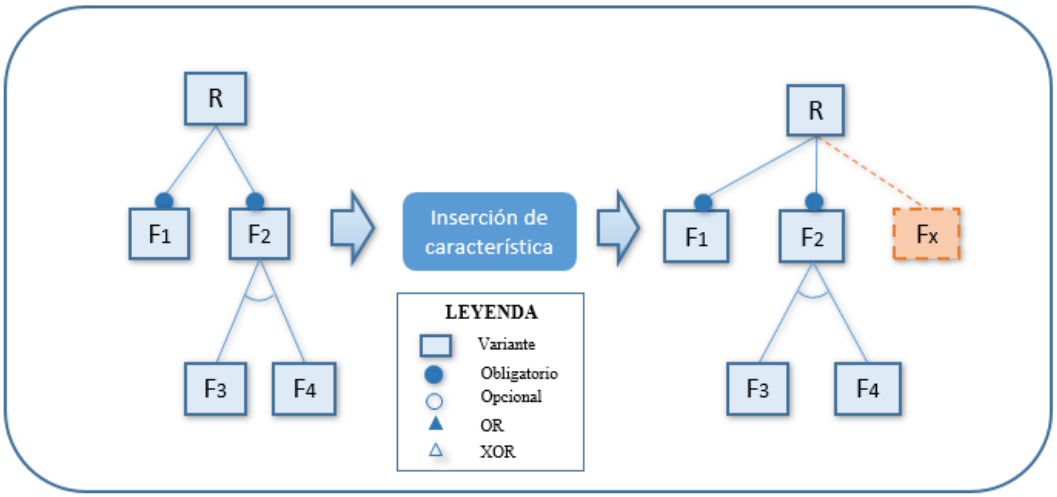



Figura 4.3. Inclusión de una característica en un escenario sin supertipos compatibles.

Hay que resaltar que Figura 4.3 describe en la leyenda únicamente un árbol de variabilidad con variantes y aunque hemos indicado si existe una relación OR o XOR, por el momento no estamos haciendo referencia a puntos de variación de manera explícita, ya que éstos los trataremos en un apartado posterior.

## 4.2.2 Caso 2. Un nodo con supertipos compatibles

Avanzando en complejidad creciente, el segundo caso que estudiaremos es el de añadir una característica en la que existe, al menos, un supertipo coincidente o compatible con uno de los puntos de variación ya presentes en el árbol de características. Es decir, que de entre todos los nodos del árbol, hay un candidato único posible. Al estudiar las posibilidades que plantea insertar una característica en un nodo con supertipos compatibles, podemos observar dos alternativas bien distintas que van a condicionar el

lugar donde ubicar esta nueva característica. El punto decisivo que diferenciará estos dos comportamientos se refiere al nodo candidato para hospedar la nueva característica y concretamente, a si tiene otros nodos hijos por debajo suyo o no. Por ello, distinguimos los siguientes escenarios:

**Escenario 2**: Si la característica existente en el modelo de variabilidad no tiene hijos, entonces la nueva característica a añadir con un supertipo compatible se añade como un hijo. La característica nueva puede ser opcional u obligatoria. En el ejemplo de la Figura 4.4 podemos observar que la característica $F_x$ tiene un supertipo $ST_1$ compatible o coincidente con los supertipos de la característica $F_2$. Sin embargo, no dispone de supertipos coincidentes con los de la característica $F_1$.

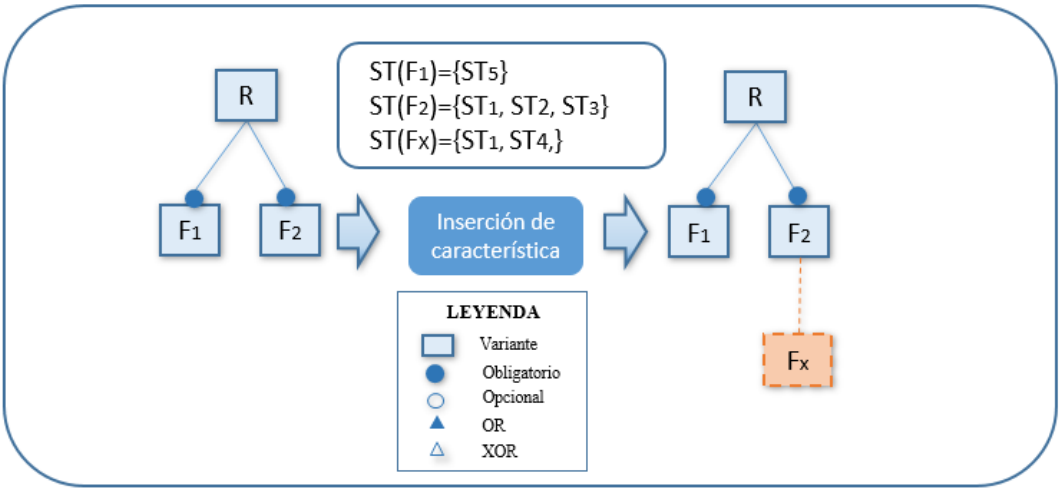



Figura 4.4. Inclusión de una característica en un escenario con un supertipo compatible

Al no hallar descendientes en $F_2$, el algoritmo opta por añadir la nueva característica como un hijo de este nodo y la nueva característica $F_X$ asume por defecto los supertipos del padre.

**Escenario 3**: El segundo de los dos escenarios posibles está representado en la Figura 4.5, y muestra el subcaso en el que el nodo candidato donde se quiere insertar la nueva característica ya posee hijos, por lo que la nueva característica sería un hermano más de las ya existentes.

En la parte izquierda de la figura se muestra el punto de partida, consistente en un árbol similar al del subcaso anterior, pero incluyendo en el nodo $F_2$ dos hijos $F_3$ y $F_4$. La relación entre ambos hermanos es de tipo XOR. Igualmente, la compatibilidad entre el nodo $F_x$ a insertar y el nodo $F_2$ se produce a través de un supertipo común $ST_1$.

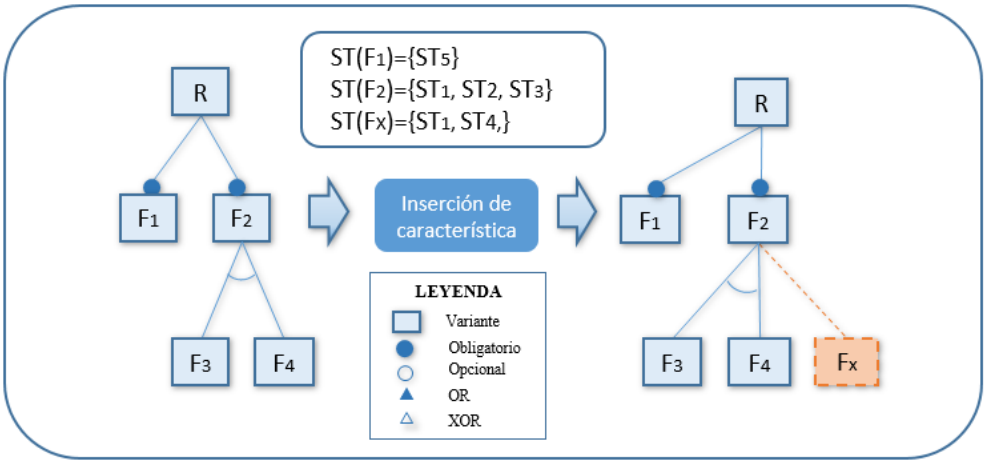



Figura 4.5. Inclusión de una característica en un escenario con un supertipo compatible

En este escenario, es necesario disponer de un mecanismo para que la característica $F_x$ se incluya como un XOR o como un OR. Para ello es necesario modelar los puntos de variación de alguna manera semiautomática o automática y disponer de las relaciones lógicas del árbol de variabilidad existente. Este problema lo discutiremos en una sección posterior ya que por el momento asumimos que la nueva característica se incluye sin problemas con la relación lógica que estuviera especificada. En nuestro caso, no asumimos ninguna relación OR/XOR predefinida para dicha característica por lo que se añade como hija de $F_2$ en base a una relación AND. Además, no hemos indicado, ya que no es relevante en este momento, si el nuevo hijo se considera una característica opcional u obligatoria. A continuación, describimos nuestro tercer caso en el que se pueden dar más de un supertipo compatible.

### 4.2.3 Caso 3. Varios nodos con supertipos compatibles

Por último, existe un caso más que completa lo ya expuesto en los dos puntos anteriores, y es la situación en la que la característica a añadir al sistema comparte supertipos comunes con algunas de las características ya presentes en el árbol de variabilidad existente. En este caso, podría haber más de un candidato para incluir la nueva característica y por ello es necesario que exista un mecanismo que discrimine o priorice la inclusión de la nueva característica en alguna de las ramas del modelo de variabilidad. Para este caso contemplamos los siguientes escenarios.

**Escenario 4**: Si existe más de una rama candidata en el árbol de variabilidad para colocar la nueva característica, entonces colocamos ésta en aquella que tenga un mayor número de supertipos coincidentes, ya que entendemos que pueden tener una mayor afinidad o bien que la nueva característica comparte su funcionalidad con diferentes partes de un sistema. Al igual que hemos descrito en los escenarios anteriores, en el ejemplo de la Figura 4.6 podemos observar el caso de inserción de una característica $F_x$ con dos supertipos $ST_1$ y $ST_2$. En este caso, existe una posibilidad donde colocar la nueva característica en base a sus supertipos coincidentes.

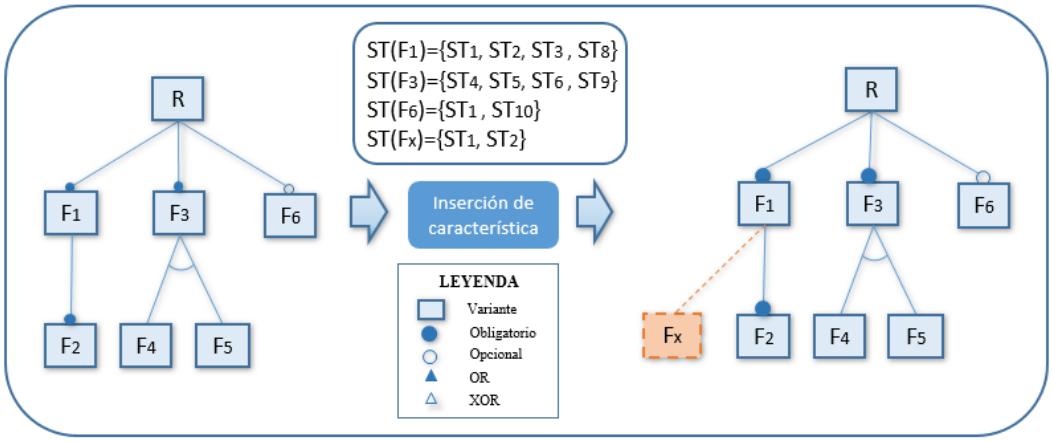



Figura 4.6. Inclusión de una característica en un escenario con varios tipos compatibles

**Escenario 5**: Finalmente, en el caso de que hubiera dos o más ramas con igual número de supertipos en común con los de la característica a añadir es necesario resolver esta situación de empate. En este caso, y con el fin de evitar cargar en exceso una determinada rama del árbol de variabilidad, colocaremos la característica nueva en aquella rama con menor número de hijos, que pueden ser variantes o puntos de variación

(escenario 5a). Si también sucediera que el número de hijos es el mismo entonces la nueva característica se incluirá en la primera opción que el algoritmo encuentre (escenario 5b).

En el ejemplo de la Figura 4.7 podemos observar que la característica $F_x$ tiene dos supertipos $ST_1$ y $ST_2$, los cuales están presentes entre los supertipos de la característica $F_1$, y al tiempo en los de la característica $F_3$. En el caso de $F_1$ hay un nodo hijo y en $F_3$ ya hay dos nodos hijos. Por ello el algoritmo decide colocar $F_x$ en la rama correspondiente a la característica $F_1$.

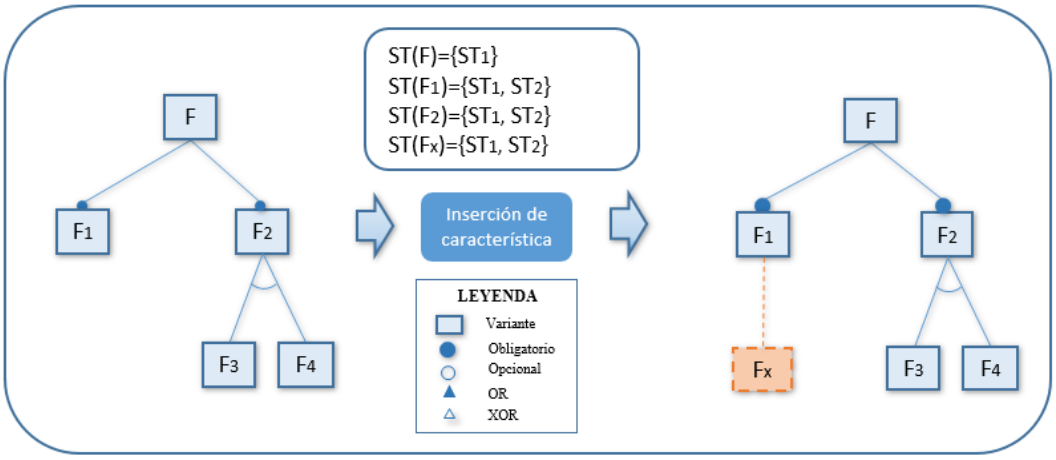



Figura 4.7. Inclusión de una característica compatible con varios puntos de variación

**Otras alternativas**: Podríamos utilizar diferentes criterios o combinaciones de ellos para decidir la mejor localización para añadir características en un árbol de variabilidad. Además de los escenarios discutidos anteriormente, otra opción sería priorizar el orden de los supertipos, de manera que la inserción de una nueva característica que cuente con dos o más supertipos se inserte, no sólo en la rama con mayor número de supertipos coincidentes, sino en aquella en los que el orden de los supertipos sea más prioritario.

Sin embargo, dado que pueden existir diferentes posibilidades de modelar un árbol de variabilidad y que no se espera que exista un gran número de supertipos para una misma característica, la elección de no sobrecargar la misma rama del árbol en función del número de hijos puede constituir una alternativa razonable que haga innecesario complicar el algoritmo de variabilidad dinámica incluyendo dicha priorización. En este sentido, cabrían otras formas de optimizar la inclusión de nuevas características en modelos de variabilidad muy grandes, pero que dejamos para trabajos futuros.

### 4.2.4 Eliminación de características

Con el fin de tratar el **subproblema 3,** disponemos de un mecanismo para eliminar característica de forma dinámica. En este caso, las operaciones son más sencillas que cuando se trata de eliminar características, ya que nos ahorramos las tareas de localización y comprobación de supertipos. En este caso, si se pretende eliminar una característica del sistema habría que analizar antes el escenario en el cual se realizaría la eliminación, atendiendo al número de hijos que tuviera dicha característica.

**Caso 1**: El caso más sencillo y que reviste menos complejidad es el de una característica sin hijos. Al no tener descendencia, no habría que hacer más comprobaciones que las que aplican a dicha característica únicamente, sin tener en cuenta las de sus hijos. En este caso, sería necesario revisar que no existiera una restricción del tipo:

$$F_A \textit{ require } F_B$$

Siendo $F_A$ la característica que se pretende eliminar. Si se diera este caso, habría que revisar o bien la implicación de quitar $F_B$, o bien de eliminar la restricción en caso de que ya no aplicara sobre el modelo. Pero en cualquier caso, sería una decisión que se aplicaría de manera manual y no automática.

En caso contrario, si no existieran restricciones como la descrita, la eliminación de la característica podría realizarse de forma automática.

**Caso 2**: El otro caso que restaría estudiar sería el de la característica que tiene 1 o varios niveles de hijos por debajo de ella (típicamente, un punto de variación). Para cada uno de los hijos, sería necesario analizar las restricciones que aplican sobre ellos. Únicamente en el caso de que ninguno de dichos hijos se viera afectado por alguna restricción, sería posible realizar el borrado de dicha característica, empezando primero por los hijos de nivel inferior, y una vez eliminados todos los descendientes, proceder al borrado de la característica. En caso de que existieran dependencias que afectaran a alguno de los descendientes, previamente habría que solucionar dichas restricciones antes de proceder a la eliminación, por lo que no sería posible proceder de forma automática.

## 4.3 Restricciones en tiempo de ejecución

Esta sección trata de dar respuesta al **subproblema 4** descrito en el planteamiento del problema. Una vez que hemos propuesto una solución con diferentes escenarios para incluir una característica nueva en un modelo de variabilidad existente y en tiempo de ejecución, es necesario comprobar que las restricciones que tiene la nueva característica a incluir son compatibles con la lista de restricciones existente, ya que en caso contrario la característica no se podría añadir. Por ello, y con el fin de no duplicar esfuerzos, ya que existen diversos resolvedores de restricciones en el mercado, vamos a decantarnos por integrar nuestra solución de variabilidad dinámica con el resolvedor de la herramienta FaMa[7], tal y como explicamos en las siguientes secciones.

### 4.3.1 Descripción de las restricciones en el modelo FaMa

FaMa es capaz de trabajar con los dos tipos principales de reglas o restricciones, las de tipo ***require*** y ***exclude***. Con estos dos tipos, es posible definir y establecer relaciones de dependencia que permitan establecer qué combinaciones son posibles de entre todas las características que están presentes en el árbol y con ello obtener un conjunto de soluciones permitidas calculando así el número de variaciones totales.

La relación ***require*** implica que, dadas dos características A y B, si está presente la característica A también debe estarlo la B para cumplir la regla. La segunda, ***exclude***, hace justamente lo contrario. Dadas dos características C y D, resultan mutuamente excluyentes, es decir, si está presente la característica C, entonces no puede estarlo la característica D. A pesar de que FaMa soporta varios formatos de modelos de características y de que uno de los más utilizados sea la representación en XML, usaremos principalmente el formato propio de FaMa por su versatilidad a la hora de adaptarse a multitud de herramientas y aplicaciones.

FaMa dispone de su propio formato de datos para expresar restricciones entre características, de manera que dadas dos características A y B, una restricción de tipo ***require*** se representa como `A REQUIRE B` y un de tipo ***exclude*** se representa como `A EXCLUDE B`.

[7] https://www.isa.us.es/fama/

## 4.4 Gestión de Puntos de Variación

Por último, esta sección va a dar respuesta al desafío planteado en el **subproblema 5**. En este sentido, cuando añadimos una característica nueva a un modelo de variabilidad de manera automática, una vez que nuestra solución ha encontrado la localización óptima en base a los supertipos compatibles, es habitual que las características hijas de la rama donde se va a incluir la nueva característica tengan una relación lógica conformando lo que se conoce como punto de variación. Llegados a este punto, sería necesario la intervención manual del diseñador para organizar de manera lógica la nueva relación al incluir dicha característica. Sin embargo, en esta tesis vamos a tratar de proporcionar una solución semi-automatizada que en algunos casos pueda reconstruir el nuevo punto de variación sin intervención humana o con mínima intervención.

La solución que proponemos para ello es almacenar en un repositorio las relaciones de los puntos de variación de un modelo de variabilidad dado y si la característica nueva es obligatoria (relación AND) o bien alternativa (relación OR o XOR). Dado que existen diferentes posibilidades para relacionar de manera lógica las características existentes en un punto de variación, vamos a describir cada uno de los posibles subcasos y la solución que proponemos para cada uno de ellos.

### 4.4.1 Característica con un solo hijo

En este primer caso presentado se añade una característica a una rama que cuenta con un único nodo hijo. En este caso, al haber un solo hijo, en el sitio donde se espera incluir la nueva característica, no existe ninguna relación lógica en la característica hijo de tipo AND, OR o XOR. Por ello, la relación que viene como la nueva característica $F_x$ es la que determinará la relación lógica entre la característica a incluir y la ya existente. En la Figura 4.8 se muestran los tres tipos de relación lógica que podrían darse al incluir $F_x$ debajo de la característica $F_2$.

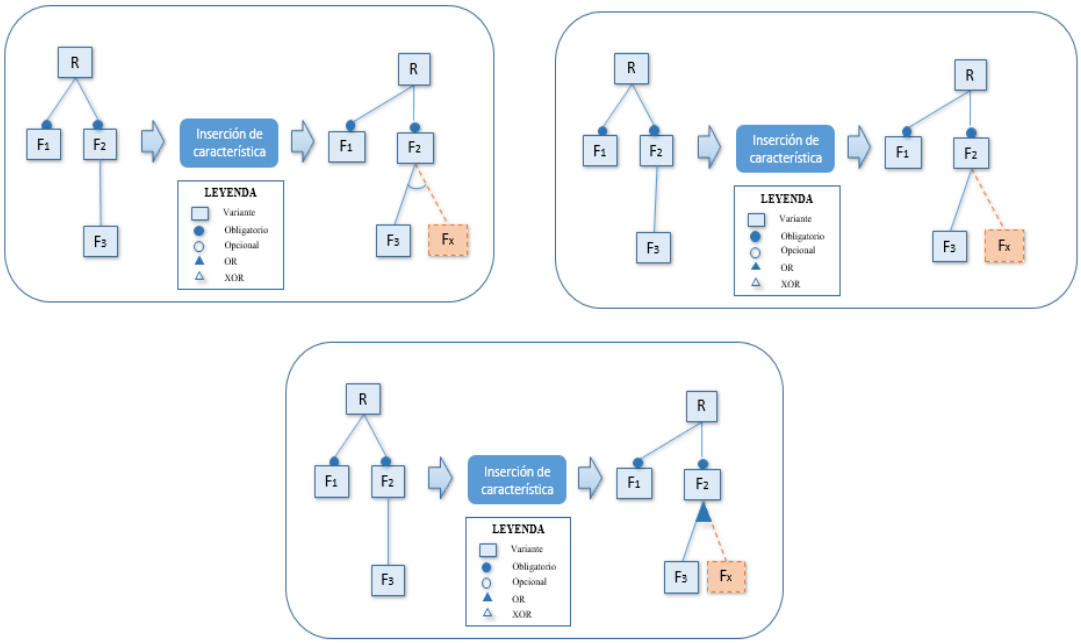



Figura 4.8. Inserción de característica en nodo con un hijo

## 4.4.2 Relación AND con dos o más características

El siguiente caso que contemplamos es un punto de variación que contenga una relación de tipo AND entre las características hijas, entonces la nueva característica se añade como AND a las existentes, tanto si no lo especifica como si viene con una relación AND predefinida. La figura 4.9 muestra cómo quedaría el árbol de variabilidad después de añadir $F_x$ como una relación AND.

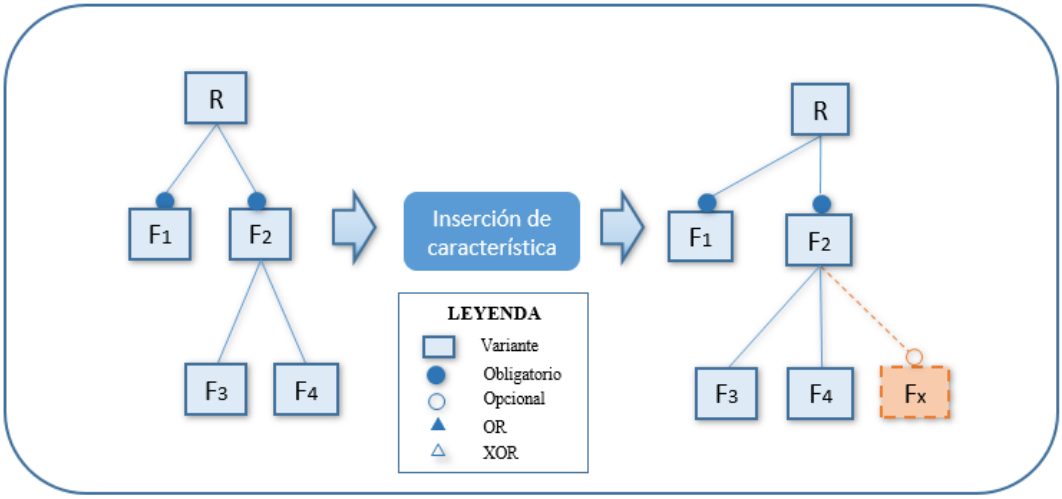



Figura 4.9. Inserción de característica en nodo AND.

Sin embargo, si la nueva característica viene definida con una relación OR o XOR y el punto de variación está modelado con relaciones AND, entonces el diseñador deberá modelar manualmente este caso ya que no podemos incluir una característica con una relación OR o XOR ya que nos harían falta al menos dos características.

### 4.4.3 Relación OR/XOR con dos o más características

Este caso es similar al anterior, pero con relaciones de tipo OR/XOR. En esta situación la nueva característica podría venir definida como OR o XOR y el punto de variación tiene una relación de OR o XOR entre sus características. Si las relaciones entre las características del punto de variación y la nueva característica son del mismo tipo, entonces la nueva característica se añade como una más a la relación existente, tal y como muestra la figura 4.10.

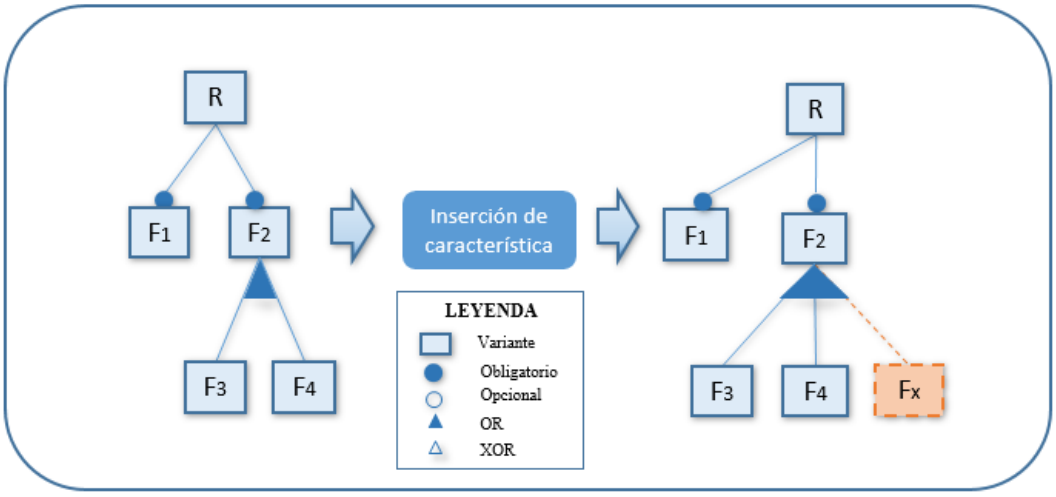



Figura 4.10. Inserción de característica en nodo OR.

Sin embargo, si la relación existente es de tipo OR y la nueva característica es de tipo XOR o viceversa, en este caso, el diseñador deberá modelar manualmente la solución ya que necesitaríamos dos características nuevas con el mismo tipo de relación. Si la nueva característica es de tipo AND, entonces se añade directamente.

### 4.4.4 Relaciones AND, OR, XOR con varias características

Entramos ahora en los casos que combinan relaciones de tipo AND y OR/XOR. Si en la nueva característica a añadir no se especifica la relación, entonces es tarea del diseñador modelar manualmente el punto de variación con la nueva característica. En el caso de que la nueva característica indique explícitamente que se debe modelar como un AND, OR o XOR, se añade de manera automática al grupo que corresponda. En el caso de que la característica que se vaya a añadir sea de tipo AND, esta se insertará, con independencia de que existan o no otras características AND en el nodo.

En el ejemplo de la Figura 4.11, se muestra un punto de variación con una relación OR que comprende las características $F_3$ y $F_4$, y otra de tipo AND que engloba $F_5$ y $F_6$. Se pretende añadir una característica $F_x$ que consideramos AND y que va a depender del punto de variación $F_2$. En este caso el resultado de la inserción queda como se muestra en la parte derecha de la Figura 4.11, ya que existe una relación previa de tipo AND.

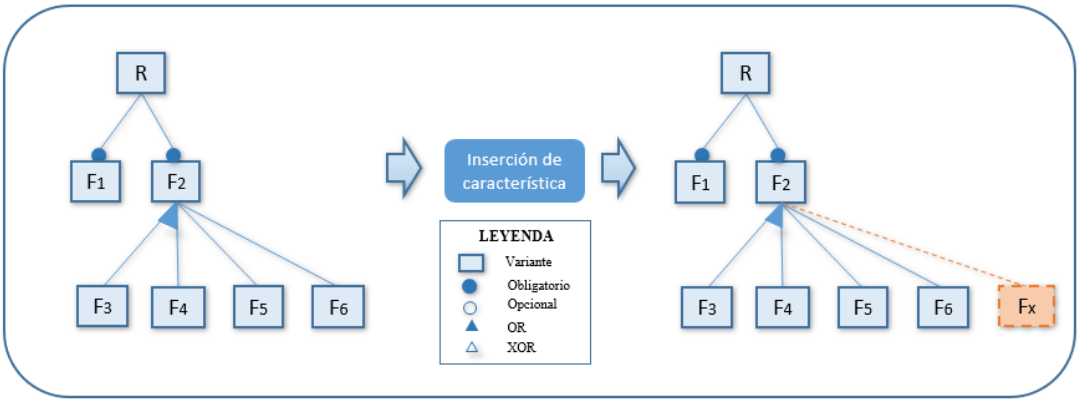



Figura 4.11. Inserción de característica obligatoria en punto de variación AND y OR.

En el caso de que la característica que se va a añadir sea de tipo OR, entonces cambiará el lugar y la relación en la que se englobaría $F_x$, tal y como muestra la Figura 4.12.

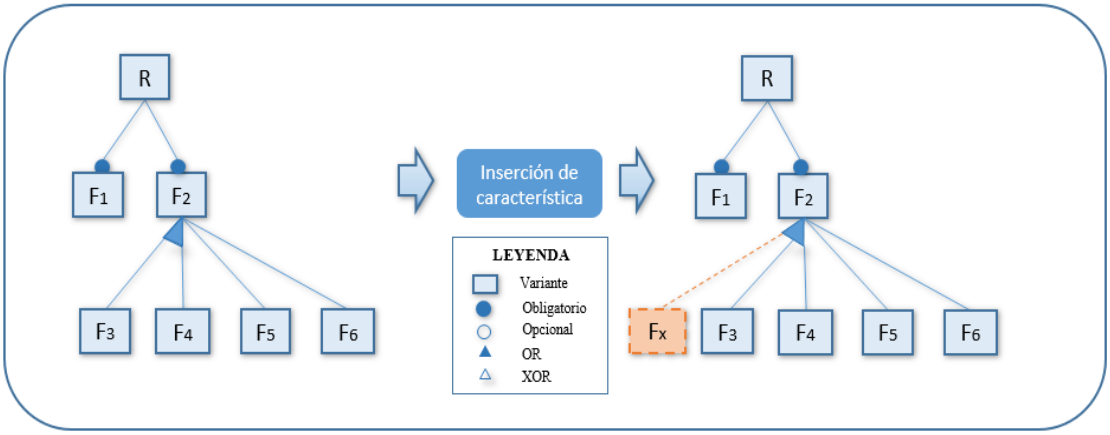



Figura 4.12. Inserción de característica opcional en un punto de variación AND y OR.

Por último, en el caso de que la característica que se va a añadir no venga especificada si es un OR o un XOR, por defecto la incluimos en la rama del OR, ya que consideramos que mediante una relación OR la nueva característica tiene más posibilidades de ser seleccionada. La Figura 4.13 muestra el ejemplo descrito.

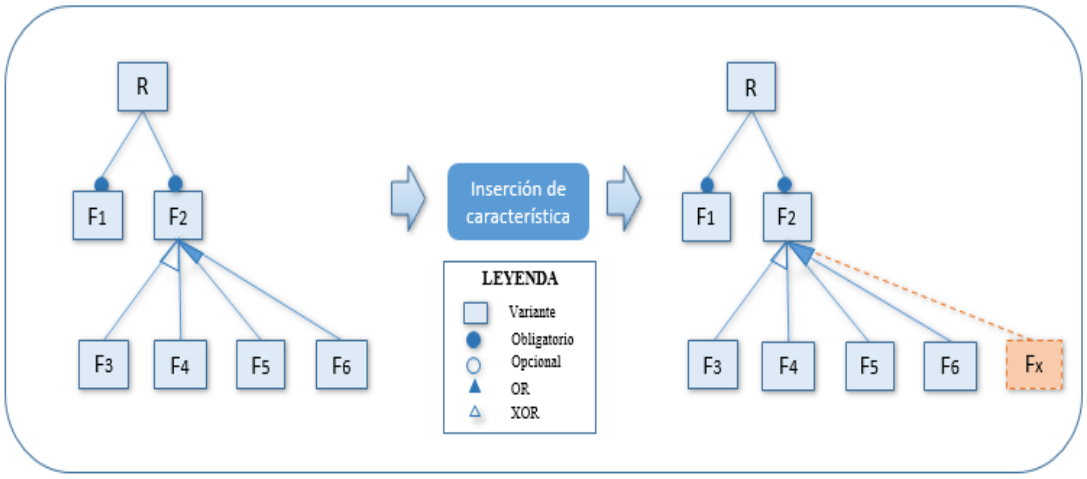



Figura 4.13. Inserción de característica opcional en punto de variación con los tres tipos de relaciones.

## 4.5 Implementación e Integración de la Solución

En este apartado vamos a describir los aspectos más importantes de la implementación y su integración con herramientas externas.

### 4.5.1 Algoritmo de variabilidad dinámica (RuVa)

En esta sección pasamos a describir el pseudocódigo correspondiente a los diferentes escenarios de la sección 4.2. Para ello utilizaremos como punto de partida la figura 4.14, donde se muestra el pseudocódigo completo que contempla los tres casos descritos de la sección 4.2.

```
RuVa_Algorithm
    fmFile <- readFMFile (fileName)
    fmTree<-populateFMFromFMFile (fmFile)
    newFeature <- featureToAdd
    for each node in nodes of fmTree
        compatibleSupertypesNumber <- compare (getSupertypes(newFeature), getSupertypes(node))
        if compatibleSupertypesNumber > 0 then
            isValidSolution <- invokeQuestionTrader(solution)
            if isValidSolution then
                childrenNumber <- getChildrenNumber(node)
                selectedNodes <- addDataNode(node,compatibleSupertypesNumber, childrenNumber)
            end if
        end if
    end for
    sort selectedNodes by compatibleSupertypesNumber desc,childrenNumber asc
    if size(selectedNodes)>0 then
        firstSelectedNode <- selectedNodes(1)
        if size(selectedNodes)>1 then
            secondSelectedNode <- selectedNodes(2)
            if compatibleStNumber(firstSelectedNode)>compatibleStNumber(secondSelectedNode) then
                newTree <- addNode(firstSelectedNode)
            else if compatibleStNumber(firstSelectedNode)==compatibleStNumber(secondSelectedNode) then
                if childrenNumber(firstSelectedNode)>=childrenNumber(secondSelectedNode) then
                    newTree <- addNode(firstSelectedNode)
                else
                    newTree <- addNode(secondSelectedNode)
                end if
            end if
        else
            newTree <- addNode(firstSelectedNode)
        end if
        fmTree=newTree
        writeFMFile(fmTree)
    else
        fmTree <- addNode(newFeature,rootNode)
        writeFMFile(fmTree)
    end if
end RuVa_Algorithm
```

Figura 4.14. Pseudocódigo de inserción de una característica

Con el fin de poder gestionar adecuadamente cualquiera de los casos representados en el punto anterior, el algoritmo ha de realizar algunos cálculos previos para obtener la información que va a permitir conocer ante qué caso nos encontramos y en atención a ello aplicar cada uno de los casos enumerados.

La primera operación y que va a servir de base para el resto de las comprobaciones, es construir una lista de características con supertipos compatibles, y dejando al margen aquellas características que no tienen supertipos en común, tal cual se puede ver en la figura 4.15, corresponde a las líneas de la 5 a la 14 del pseudocódigo. Para hacer la construcción de esta lista, se recorren todos los nodos que componen el árbol y se van anotando el número de supertipos compatibles de cada nodo y el número de hijos.

```
5  for each node in nodes of fmTree
6      compatibleSupertypesNumber <- compare (getSupertypes(newFeature), getSupertypes(node))
7      if compatibleSupertypesNumber > 0 then
8          isValidSolution <- invokeQuestionTrader(solution)
9          if isValidSolution then
10             childrenNumber <- getChildrenNumber(node)
11             selectedNodes <- addDataNode(node,compatibleSupertypesNumber, childrenNumber)
12         end if
13     end if
14 end for
```

Figura 4.15. Generación de lista de características con supertipos compatibles

La segunda operación complementa a la anterior y ordena lista obtenida en el cálculo anterior, de manera que sea posible acceder a las características que más supertipos compatibles tengan con la característica a añadir. Esto se corresponde con la línea 15 del algoritmo, reseñada en la figura 4.16. La lista se ordenará en base a dos criterios, que son el número de supertipos compatibles de mayor a menor, y el número de características hijas que poseen, de menor a mayor a través de una operación *sort*.

```
14 end for
15 sort selectedNodes by compatibleSupertypesNumber desc,childrenNumber asc
16 if size(selectedNodes)>0 then
```

Figura 4.16. Lista ordenada de opciones donde insertar una característica

Posteriormente, describimos las partes del algoritmo que resuelven los tres casos o escenarios descritos en la sección 4.2 y correspondientes a la inserción de una característica nueva.

**Caso 1. Ningún supertipo compatible**

El pseudocódigo correspondiente a esta parte se muestra en la Figura 4.17. Tras consultar en la línea 16 por el número de características con supertipos compatibles, si no se encontrara ninguna característica, quiere decir que nos encontraríamos ante el primero de los escenarios correspondiente al caso 1, lo que equivale a que no hay nodos con supertipos compatibles. Tal y como podemos observar entre las líneas 34 y 37, cuando no se encuentran características con supertipos compatibles con los de la característica a añadir, la única opción es insertar la característica en la raíz del árbol.

```
15      sort selectedNodes by compatibleSupertypesNumb
16      if size(selectedNodes)>0 then
17          firstSelectedNode <- selectedNodes(1)
33          writeFMFile(fmTree)
34      else
35          fmTree <- addNode(newFeature,rootNode)
36          writeFMFile(fmTree)
37      end if
38  end RuVa_Algorithm
```

Figura 4.17. Caso 1 representado en el pseudocódigo

**Casos 2 y 3. Uno o más supertipos compatibles**

El pseudocódigo correspondiente a este caso incluye los escenarios 2, 3, 4 y 5 de la sección 4.2 en la que se contemplan ambas opciones de esta sección en las que existe una o más características con al menos un supertipo en común con la característica a añadir. Este pseudocódigo se muestra en la Figura 4.18. Con el fin de gestionar esta casuística, en la línea 18 del pseudocódigo es donde se realiza la comprobación para determinar si nos encontramos en un escenario con una única solución (en cuyo caso esta sería la escogida, siempre que el motor de restricciones lo aprobara) o por el contrario, de varias.

En caso de que no se cumpla la condición expresada (que el número de nodos candidatos sea superior a 1), quiere decirse que sólo hay una opción posible donde alojar la nueva característica, y por tanto, se insertará bajo esta, tal y como se refleja entre las líneas 29 y 31, correspondiendo al escenario 2 y 3 (un nodo con supertipos compatibles, sin hijos y con hijos respectivamente).

```
17      firstSelectedNode <- selectedNodes(1)
18      if size(selectedNodes)>1 then
19          secondSelectedNode <- selectedNodes(2)

27          end if
28      end if
29  else
30      newTree <- addNode(firstSelectedNode)
31  end if
32  fmTree=newTree
33  writeFMFile(fmTree)
```

Figura 4.18. Escenarios 2a y 2b representado en el pseudocódigo

Cuando la característica a insertar en el árbol posee varias posibilidades, esta situación tiene su representación en la Figura 4.19.

```
18  if size(selectedNodes)>1 then
19      secondSelectedNode <- selectedNodes(2)
20      if compatibleStNumber(firstSelectedNode)>compatibleStNumber(secondSelectedNode) then
21          newTree <- addNode(firstSelectedNode)
22      else if compatibleStNumber(firstSelectedNode)==compatibleStNumber(secondSelectedNode) then
23          if childrenNumber(firstSelectedNode)<=childrenNumber(secondSelectedNode) then
24              newTree <- addNode(firstSelectedNode)
25          else
26              newTree <- addNode(secondSelectedNode)
27          end if
28      end if
29  else
```

Figura 4.19. Escenarios 3 y 4 representados en el pseudocódigo

Como en los casos previos, la primera parte es realizar una búsqueda de entre todos los nodos para obtener cuáles tienes mayor número de supertipos compatibles y elegimos aquel que tenga más supertipos compatibles con la característica a insertar. En este caso, el tamaño de la lista de nodos candidatos obtenida siempre va a ser mayor que uno (línea 18), lo cual quiere decir que habrá más de una solución viable para alojar la nueva característica. Este escenario se refleja en las líneas 19 a 28, que corresponden a los escenarios 4 (varios nodos con supertipos compatibles) y 5 (varios nodos con igual número de supertipos compatibles).

El escenario 4 se corresponde con que hay varias características con uno o más supertipos compatibles, y por tanto el algoritmo busca aquella que tenga más supertipos en común que las demás. En la línea 20 se hace la comprobación de si el primer elemento de la lista ordenada es el que más supertipos en común tiene con la nueva característica, y si es así se escoge este nodo como lugar para añadir la nueva característica (línea 21), con lo que ya estaría contemplado el escenario 4 (varios nodos con supertipos compatibles).

En caso de que el primer y segundo elementos de la lista tuvieran igual número de supertipos en común con la característica a añadir, se utilizaría el criterio de buscar el que menos nodos hijos tuviera de las dos opciones a valorar. Esta comprobación se realiza en las líneas 23 a 27 y contemplaría el escenario 5 (varios nodos con igual número de supertipos compatibles). Aquí entra en juego el optimizador de características que facilita la elección de la mejor opción de entre las posibles. Una muestra del resultado generado por RuVa se encuentra en el Anexo I.

## 4.5.2 Integración con FaMa

A continuación, pasamos a explicar en esta sección cómo funciona el resolvedor de restricciones de la herramienta elegida y como lo integramos con nuestra solución de variabilidad dinámica. FaMa dispone de un módulo denominado «QuestionTrader» que permite interactuar con los resolvedores[8] y lanzarles preguntas que puedan determinar si un modelo de características presenta errores o por el contrario es válido, o para extender las decisiones en un proceso de configuración.

La integración entre nuestro algoritmo y el resolvedor consiste en lo siguiente. En primer lugar, adaptamos nuestra solución para que pueda funcionar con los modelos de características de FaMa, tanto en formato FaMa-XML como FaMa-.fm. Una característica auxiliar es una característica cuya función es la de asistir para realizar la representación del árbol en los casos donde viene indicada una cardinalidad. Estas características abstractas nos ayudan a indicar la cardinalidad de las relaciones AND, OR y XOR.

[8] La suite FaMa no dispone de un solver propio, sino que incorpora un módulo que comunica con los diferentes resolvedores como Choco Solver

La integración entre la solución de variabilidad dinámica y FaMa para realizar la operación de inserción y borrado de características consiste en los siguientes pasos referidos a la figura 4.20:

**Paso 1**: Se carga el modelo de características tal y como se describe en la fase 1 de la Figura 4.20.

**Paso 2**: Con el modelo de características seleccionado, se decide la acción a llevar a cabo (añadir o eliminar característica), lo que desencadena la petición para localizar la posición donde se encuentra la característica sobre la que se va a realizar la acción, como se muestra en la Figura 4.20. Estas comprobaciones constan de una serie de etapas que pasarán por el «Localizador de características», el «comprobador de supertipos» y el «optimizador».

**Paso 3**: Se devuelve el «resultado de la búsqueda de características», tal y como se describe en la fase 3 de la Figura 4.20.

**Paso 4**: Se realiza la «solicitud de comprobación de restricciones de características» desde RuVa hacia FaMa, invocando el módulo de «QuestionTrader» con la propuesta del nuevo modelo de características sugerido por el módulo de Control en el paso anterior, tal y como se describe en la fase 4 de la Figura 4.20.

**Paso 5**: El módulo de «QuestionTrader» lanza la petición hacia el Solver de FaMa, con el modelo de características a validar y las restricciones, tal y como se describe en la fase 5 de la Figura 4.20.

**Paso 6**: El módulo de Solver devuelve el resultado de la comprobación, que puede ser positivo o negativo, tal y como se describe en la fase 5 de la Figura 4.20.

**Paso 7**: El módulo de «QuestionTrader» de FaMa ya puede determinar la respuesta a la pregunta de si el nuevo modelo es o no válido con la información que tiene tras invocar al Solver, y traslada la respuesta de vuelta hacia el módulo de operaciones de RuVa, tal y como se describe en la fase 7 de la Figura 4.20.

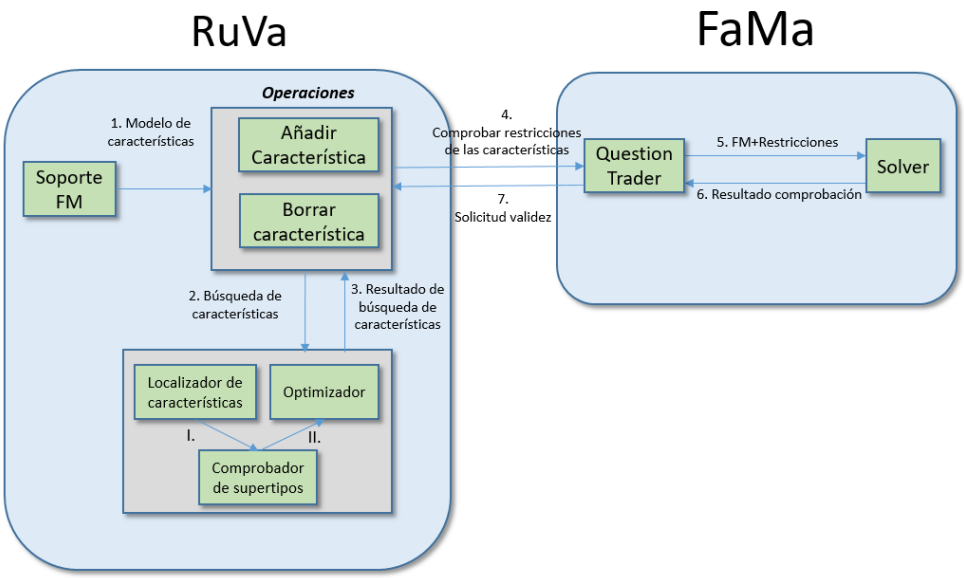



Figura 4.20. Modelo de interacción entre RuVa y FaMa

Una vez que hemos descrito las partes principales de nuestra solución, pasamos a detallar en el siguiente capítulo como hemos realizado la experimentación. Asimismo, los detalles del código fuente se encuentran publicados en el siguiente repositorio[9] y la estructura de los ficheros de comandos utilizados por RuVa se encuentra descrita en el Anexo II.

[9] http://github.com/valdezate/RuVa

# Capítulo 5. Experimentación

En este capítulo vamos a proponer la solución a los cuatro subproblemas planteados en el capítulo anterior. La experimentación va a consistir en tres tipos de simulaciones. En el primer tipo simularemos la ejecución del algoritmo con un modelo de variabilidad predefinido conteniendo ejemplos de las diferentes casuísticas expuestas en el capítulo 4 y en el que probaremos la ejecución de diferentes escenarios. En el segundo tipo se utilizan modelos autogenerados con la herramienta BeTTy[10] con un abanico de modelos con diferente número de características (100, 1000, 2000, 3000, 4000 y 5000) y supertipos, en donde se mostrarán mediciones de tiempos y se aplicarán pruebas de estrés sobre el algoritmo que permitan mostrar su rendimiento. En el tercer tipo utilizaremos un modelo de variabilidad correspondiente a un robot y simularemos la actualización y reconfiguración del robot de manera dinámica.

## 5.1 Simulación de Modelos de Variabilidad

En primer lugar, haremos una prueba de eficiencia del algoritmo y, a continuación, pasaremos a las pruebas de rendimiento.

Para árboles de 50 características o menos, consideramos un máximo de 5 supertipos diferentes que podrán estar presentes en los nodos del árbol. Para árboles mayores, este número llegará hasta 10 supertipos diferentes como máximo.

### 5.1.1 Pruebas de Eficiencia del Algoritmo

En esta primera simulación se van a realizar pruebas de comportamiento sobre un modelo creado manualmente para probar todos los casos expuestos en el Capítulo 4.4. Este modelo, tal y como se muestra en la Figura 5.1, cuenta con 50 características y 5 supertipos.

[10] Los modelos generados con BeTTy han sido realizados en colaboración con la Universidad de Sevilla.

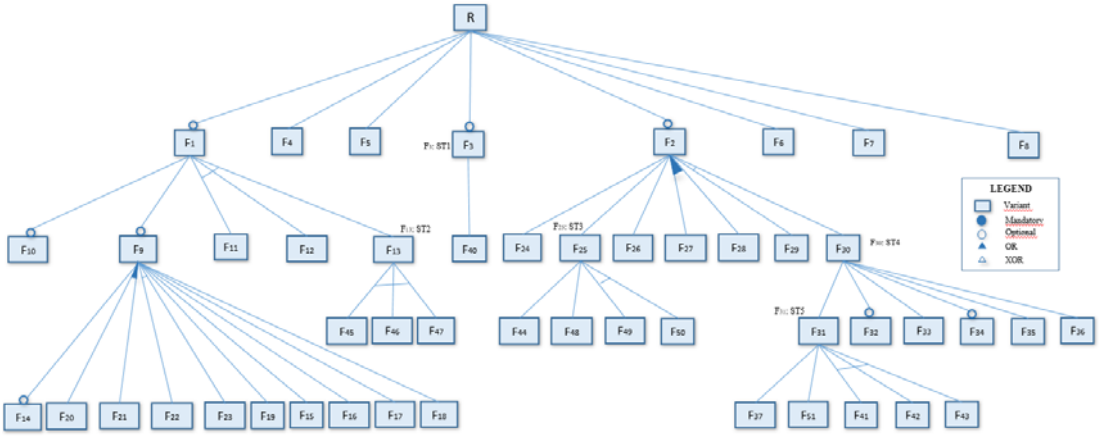



Figura 5.1. Modelo de variabilidad para testing de eficiencia

Del total de nodos, 41 son nodos terminales (82%). En el árbol existen 3 tipos de relaciones, que quedan agrupadas de la forma siguiente: 8 nodos de tipo AND, 3 nodos de tipo OR y 7 nodos de XOR. De las características contenidas en el árbol, existen 42 de tipo obligatorio (84%) y 8 de tipo opcional (16%).

**Baterías de pruebas:** Para poner a prueba el algoritmo se ha llevado a cabo la inserción de 4 bancos de pruebas, que consisten en insertar 1, 3, 5 y 10 características respectivamente. Cada juego de pruebas puede contener hasta 5 supertipos. Cada juego de pruebas se repetirá dos veces, la primera usando los supertipos que se encuentran en el árbol y la segunda añadiendo otros supertipos a los ya presentes.

Para evaluar la eficiencia se han aplicado los siguientes juegos de prueba (*sets*) mostrados en las tablas 5.1, 5.2, 5.3 y 5.4:

Tabla 5.1. Sets de pruebas de 1 elemento

| SET 1-1 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | AND | ST2 |

| SET 1-2 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | XOR | ST3 |

Tabla 5.2. Sets de pruebas de 3 elementos

| SET 3-1 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | XOR | ST2 |
| FX2 | OR | ST5 |
| FX3 | XOR | ST3 |

| SET 3-2 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | AND | ST2 |
| FX2 | OR | ST4 |
| FX3 | OR | ST3 |

Tabla 5.3. Sets de pruebas de 5 elementos

| SET 5-1 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | OR | ST1 |
| FX2 | AND | ST3 |
| FX3 | OR | ST2 |
| FX4 | OR | ST4 |
| FX5 | OR | ST5 |

| SET 5-2 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | AND | ST2 |
| FX2 | XOR | ST4 |
| FX3 | AND | ST3 |
| FX4 | XOR | ST1 |
| FX5 | OR | ST5 |

Tabla 5.4. Sets de pruebas de 10 elementos

| SET 10-1 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | AND | ST2,ST4 |
| FX2 | OR | ST1,ST3 |
| FX3 | OR | ST2 |
| FX4 | XOR | ST1 |
| FX5 | AND | ST7 |
| FX6 | AND | ST1,ST5 |
| FX7 | AND | ST4,ST5 |
| FX8 | OR | ST2,ST3 |
| FX9 | OR | ST4 |
| FX10 | XOR | ST5,ST4 |

| SET 10-2 | | |
|---|---|---|
| FX | TIPO | ST |
| FX1 | OR | ST4 |
| FX2 | OR | ST5,ST4 |
| FX3 | OR | ST7 |
| FX4 | XOR | ST3 |
| FX5 | OR | ST4,ST3 |
| FX6 | AND | ST1,ST2 |
| FX7 | XOR | ST3,ST2 |
| FX8 | AND | ST1,ST6 |
| FX9 | AND | ST10,ST7 |
| FX10 | XOR | ST9,ST3 |

**Resultados:** Como resultado hemos obtenido un promedio de tiempo entre 0.60 y 1.00 segundos por cada intento de inserción. Los tiempos de inserción para cada *set* de pruebas se muestran en la tabla 5.5, así como el número de inserciones exitosas y fallidas (columnas 3 y 4 respectivamente).

Tabla 5.5. Resultados al aplicar los SETs de inserción.

| Juego de prueba | Tiempo (segundos) | Inserciones válidas | Inserciones no válidas | ratio |
|---|---|---|---|---|
| 1-1 | 1,03 | 1 | 0 | 100,0% |
| 1-2 | 0,99 | 1 | 0 | 100,0% |
| 3-1 | 1,98 | 2 | 1 | 66,7% |
| 3-2 | 0,96 | 1 | 2 | 33,3% |
| 5-1 | 1,80 | 2 | 3 | 40,0% |
| 5-2 | 2,65 | 3 | 2 | 60,0% |
| 10-1 | 7,63 | 8 | 2 | 80,0% |
| 10-2 | 6,89 | 8 | 2 | 80,0% |

En la Figura 5.2 se muestra el modelo resultante tras aplicar el juego de pruebas 10-1 de esta batería.

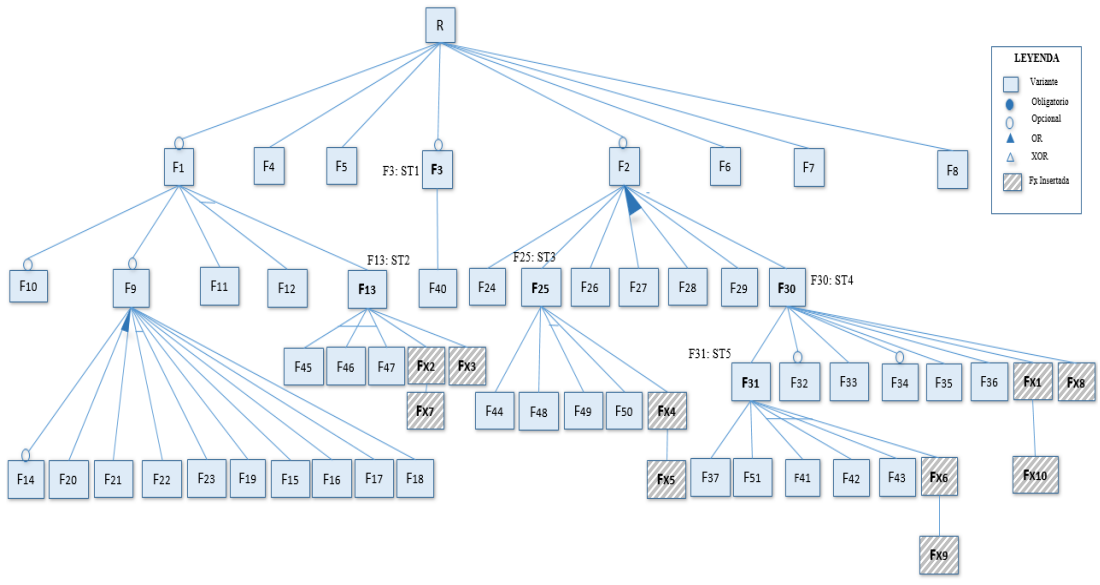



Figura 5.2. Modelo de variabilidad resultante para el set 10-1

## 5.1.2 Repetición aleatoria de Pruebas de Eficiencia

En este apartado vamos a comprobar que los resultados de eficiencia de las pruebas anteriores son correctos. Por ello, vamos a simular 100 veces la inserción de 1, 3, 5 y 10 características de manera aleatoria. La figura 5.6 muestra los resultados en tiempo y porcentaje de éxito de 100 repeticiones para insertar una característica. El resto de tablas con los resultados obtenidos de insertar 3, 5 y 10 características se muestran en el Anexo III.

Tabla 5.6. Repeticiones para insertar 100 veces una característica

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-1-01.init | 0 | 1 | 0% | 19 |
| set-1-02.init | 1 | 0 | 100% | 1183 |
| set-1-03.init | 1 | 0 | 100% | 1132 |
| set-1-04.init | 1 | 0 | 100% | 853 |
| set-1-05.init | 1 | 0 | 100% | 957 |
| set-1-06.init | 1 | 0 | 100% | 828 |
| set-1-07.init | 1 | 0 | 100% | 962 |
| set-1-08.init | 1 | 0 | 100% | 787 |
| set-1-09.init | 0 | 1 | 0% | 0 |
| set-1-10.init | 0 | 1 | 0% | 1 |
| set-1-11.init | 1 | 0 | 100% | 953 |
| set-1-12.init | 1 | 0 | 100% | 843 |
| set-1-13.init | 1 | 0 | 100% | 772 |
| set-1-14.init | 1 | 0 | 100% | 893 |
| set-1-15.init | 1 | 0 | 100% | 843 |
| set-1-16.init | 1 | 0 | 100% | 814 |
| set-1-17.init | 1 | 0 | 100% | 831 |
| set-1-18.init | 1 | 0 | 100% | 757 |
| set-1-19.init | 1 | 0 | 100% | 904 |
| set-1-20.init | 1 | 0 | 100% | 863 |
| set-1-21.init | 1 | 0 | 100% | 793 |
| set-1-22.init | 1 | 0 | 100% | 772 |
| set-1-23.init | 1 | 0 | 100% | 903 |
| set-1-24.init | 0 | 1 | 0% | 0 |
| set-1-25.init | 1 | 0 | 100% | 857 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-1-26.init | 1 | 0 | 100% | 794 |
| set-1-27.init | 1 | 0 | 100% | 728 |
| set-1-28.init | 1 | 0 | 100% | 813 |
| set-1-29.init | 1 | 0 | 100% | 705 |
| set-1-30.init | 1 | 0 | 100% | 809 |
| set-1-31.init | 1 | 0 | 100% | 899 |
| set-1-32.init | 1 | 0 | 100% | 767 |
| set-1-33.init | 1 | 0 | 100% | 835 |
| set-1-34.init | 1 | 0 | 100% | 776 |
| set-1-35.init | 0 | 1 | 0% | 0 |
| set-1-36.init | 0 | 1 | 0% | 0 |
| set-1-37.init | 0 | 1 | 0% | 0 |
| set-1-38.init | 1 | 0 | 100% | 800 |
| set-1-39.init | 0 | 1 | 0% | 0 |
| set-1-40.init | 1 | 0 | 100% | 793 |
| set-1-41.init | 1 | 0 | 100% | 723 |
| set-1-42.init | 1 | 0 | 100% | 773 |
| set-1-43.init | 1 | 0 | 100% | 716 |
| set-1-44.init | 1 | 0 | 100% | 736 |
| set-1-45.init | 1 | 0 | 100% | 791 |
| set-1-46.init | 1 | 0 | 100% | 762 |
| set-1-47.init | 1 | 0 | 100% | 774 |
| set-1-48.init | 1 | 0 | 100% | 688 |
| set-1-49.init | 1 | 0 | 100% | 735 |
| set-1-50.init | 1 | 0 | 100% | 774 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-1-51.init | 1 | 0 | 100% | 724 |
| set-1-52.init | 1 | 0 | 100% | 779 |
| set-1-53.init | 1 | 0 | 100% | 729 |
| set-1-54.init | 0 | 1 | 0% | 0 |
| set-1-55.init | 1 | 0 | 100% | 736 |
| set-1-56.init | 1 | 0 | 100% | 754 |
| set-1-57.init | 1 | 0 | 100% | 750 |
| set-1-58.init | 1 | 0 | 100% | 750 |
| set-1-59.init | 0 | 1 | 0% | 0 |
| set-1-60.init | 1 | 0 | 100% | 785 |
| set-1-61.init | 1 | 0 | 100% | 1130 |
| set-1-62.init | 1 | 0 | 100% | 826 |
| set-1-63.init | 1 | 0 | 100% | 886 |
| set-1-64.init | 1 | 0 | 100% | 785 |
| set-1-65.init | 1 | 0 | 100% | 772 |
| set-1-66.init | 1 | 0 | 100% | 808 |
| set-1-67.init | 1 | 0 | 100% | 874 |
| set-1-68.init | 1 | 0 | 100% | 769 |
| set-1-69.init | 1 | 0 | 100% | 799 |
| set-1-70.init | 0 | 1 | 0% | 0 |
| set-1-71.init | 1 | 0 | 100% | 788 |
| set-1-72.init | 0 | 1 | 0% | 0 |
| set-1-73.init | 1 | 0 | 100% | 801 |
| set-1-74.init | 1 | 0 | 100% | 784 |
| set-1-75.init | 1 | 0 | 100% | 829 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-1-76.init | 1 | 0 | 100% | 755 |
| set-1-77.init | 1 | 0 | 100% | 800 |
| set-1-78.init | 1 | 0 | 100% | 825 |
| set-1-79.init | 1 | 0 | 100% | 789 |
| set-1-80.init | 1 | 0 | 100% | 774 |
| set-1-81.init | 0 | 1 | 0% | 0 |
| set-1-82.init | 1 | 0 | 100% | 899 |
| set-1-83.init | 1 | 0 | 100% | 757 |
| set-1-84.init | 0 | 1 | 0% | 0 |
| set-1-85.init | 1 | 0 | 100% | 817 |
| set-1-86.init | 0 | 1 | 0% | 0 |
| set-1-87.init | 1 | 0 | 100% | 833 |
| set-1-88.init | 0 | 1 | 0% | 0 |
| set-1-89.init | 0 | 1 | 0% | 0 |
| set-1-90.init | 0 | 1 | 0% | 0 |
| set-1-91.init | 1 | 0 | 100% | 944 |
| set-1-92.init | 0 | 1 | 0% | 0 |
| set-1-93.init | 1 | 0 | 100% | 752 |
| set-1-94.init | 0 | 1 | 0% | 0 |
| set-1-95.init | 1 | 0 | 100% | 798 |
| set-1-96.init | 0 | 1 | 0% | 0 |
| set-1-97.init | 0 | 1 | 0% | 0 |
| set-1-98.init | 1 | 0 | 100% | 803 |
| set-1-99.init | 0 | 1 | 0% | 0 |
| set-1-100.in | 0 | 1 | 0% | 0 |

Una vez realizadas las repeticiones y con el fin de contrastar la eficiencia del algoritmo, hemos obtenido los resultados que se muestran en la tabla 5.7. Para comparar los diferentes resultados se han tenido en cuenta los rangos de tiempo obtenidos para cada uno de los conjuntos de prueba, expresados en segundos, el tiempo medio de las inserciones, los rangos de éxito (entendiéndose por éxito cuando una inserción se lleva a cabo), y la tasa media de éxito.

Tabla 5.7. Resultados de pruebas de inserción de conjuntos de 100 elementos.

| Conjuntos de prueba | Rangos de tiempo (s) | Tiempo medio (s) | Tasa media de éxito |
|---|---|---|---|
| Repeticiones de inserción de 1 característica | [0,688-1,183] | 0,621 | 76% |
| Repeticiones de inserción de 3 características | [0,770-2,787] | 0,678 | 90% |
| Repeticiones de inserción de 5 características | [1,363-3,631] | 0,628 | 90% |
| Repeticiones de inserción de 10 características | [4,314-9,359] | 0,741 | 92% |

Para las inserciones de una característica, los rangos de tiempo obtenidos en milisegundos son los que figuran en la Tabla 5.7. En las siguientes repeticiones observamos los tiempos van creciendo a medida que se inserta un mayor número de características, lo cual es razonable. Como podemos observar, el tiempo máximo para insertar 10 características ronda los 10 segundos, que supone prácticamente multiplicar por 10 el valor máximo para insertar 1 característica.

Respecto a las tasas de éxito podemos decir que la tasa media arranca con un 76% en inserciones satisfactorias en juegos de pruebas una sola inserción. Este ratio medio se incrementa hasta un 90% en repeticiones de inserción de 3 y 5 características, lo cual supone un aumento de 14 puntos con respecto al primer caso. Finalmente, las repeticiones que insertan 10 características arrojan resultados cercanos al 92%, lo cual es un resultado bastante satisfactorio y que para superarlo sería necesario disponer de un modelo de características con otra topología y una distribución de supertipos más favorable.

A continuación, pasamos a describir las pruebas de rendimiento realizadas.

### 5.1.3 Pruebas de Rendimiento

En este apartado vamos a realizar pruebas de rendimiento para comprobar cómo se comportaría en caso de modelos grandes de variabilidad. Entendemos por rendimiento el tiempo que se tarda en insertar un número de características en modelos grandes de variabilidad y que demuestren la escalabilidad de la solución aportada y en qué momento puede decaer dicho rendimiento.

Por ello, y dado que BeTTy puede generar modelos de miles de características, hemos considerado realizar pruebas con modelos de variabilidad de diferente tamaño que van desde las 100 hasta las 5.000 características. Además, con el fin de simular las mismas condiciones en todas las pruebas, utilizaremos el mismo equipamiento hardware consistente en un ordenador con procesador Intel i7-6700T CPU a 2.81 GHz y 16 GB de RAM y con Windows 10 Professional como sistema operativo. Si bien esta máquina tiene potencia para desarrollar las pruebas, un equipo con menos prestaciones de memoria RAM también podría arrojar resultados similares. En todas las pruebas utilizaremos modelos aleatorios de 100 repeticiones de 10 características con combinaciones aleatorias de relaciones AND, OR, XOR.

Para todos los modelos, consideramos un 2% como el número máximo de supertipos con un límite de 50 para modelos de variabilidad muy grandes. En aquellos nodos que tengan supertipos asociados, se utilizarán un máximo de 2 supertipos de 10 posibles.

**Modelo de 100 características:** En este primer caso consideraremos un modelo de variabilidad con cien características, diez de ellas con supertipos, siendo dos el máximo de supertipos que puede albergar un mismo nodo. El modelo tiene cinco restricciones, siendo dos de tipo *require* y tres de tipo *exclude*. En la Figura 5.3 se expone dicho modelo en formato de texto y en la Figura 5.4 se muestra un subconjunto del modelo de variabilidad. Situaremos los supertipos manualmente entre los niveles *2* y *n-1*.

```
%Relationships
R: [F1] [F2] F3 [F4] [F5] F6 F7 F8 [F9];
F2: F28 [F29] [F30] F31 [1,3]{F32 F33 F34} [1,1]{F35 F36 F37};
F3: F27;
F4: [F91];
F6: [F10] [F11] F12 F13 F14 F15 [1,2]{F16 F17};
F7: [F38] F39 F40 [1,1]{F41 F42 F43};
F8: [F55] [F56] F57 F58 F59 [1,1]{F60 F61 F62};
F9: F78 [F79] [F80] [1,1]{F81 F82};
F28: [1,1]{F94 F95 F96 F97 F98 F99};
F32: F63 [F64] [F65] [F66] F67 F68 [1,4]{F69 F70 F71 F72};
F35: F83 F84 [1,2]{F92 F93};
F36: F88;
F27: [1,1]{F52 F53 F54};
F11: F46 F47 [F48];
F12: [F44] F45;
F13: [F49] [F50] F51;
F16: [F18] F19 F20 [F21] [F22] [1,2]{F23 F24} [1,1]{F25 F26};
F72: F73 [F74] [F75] [F76] F77;
F19: [F85] F86 F87 [F89] [F90];

%Constraints
F18 EXCLUDE F22;
F78 REQUIRE F79;
F18 REQUIRE F19;
F14 REQUIRE F28;
F38 EXCLUDE F40;

%Supertypes
F28: ST1;
F32: ST3,ST5;
F3: ST7;
F4: ST2,ST4;
F11: ST1,ST6;
F12: ST9;
F16: ST5,ST9;
F7: ST5;
F8: ST10;
F9: ST1,ST5;
```

Figura 5.3. Modelo de 100 características en formato texto

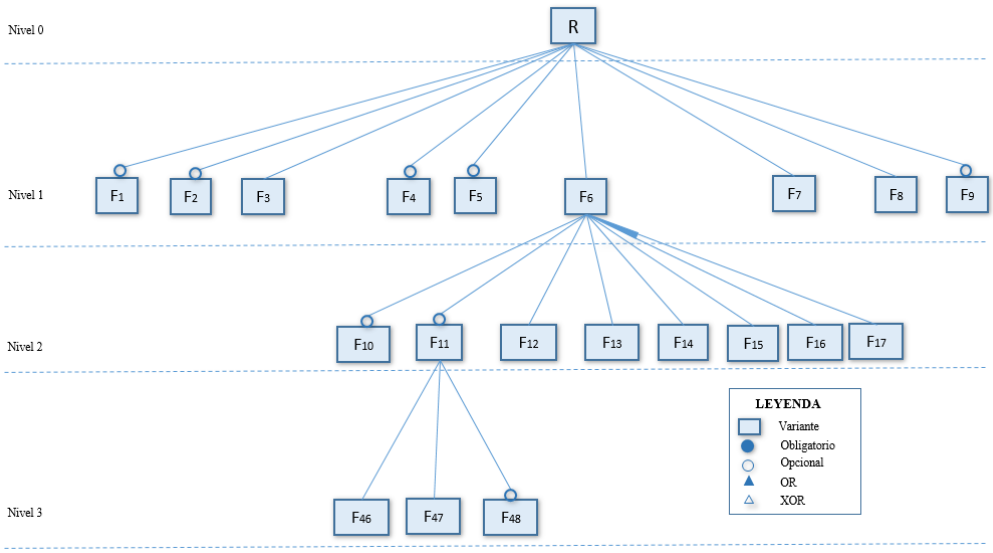



Figura 5.4. Subconjunto del modelo de 100 características

**Modelos de 1.000 a 5.000 características:** A continuación, pasamos a repetir las pruebas realizadas anteriormente para modelos de 1000, 2000, 3000, 4000 y 5000 características. Para ello, se ejecutarán 2 juegos de prueba con 10 características cada uno (ver Anexo III). Los resultados de rendimiento que hemos obtenido son desde 0,80 segundos en el modelo de 1.000 características, hasta 1,61 segundos en el modelo de 5.000 características. En la Tabla 5.8 se indican los ratios y tiempos de inserción medios en los modelos testeados. En la gráfica de la Figura 5.5, podemos consultar la comparativa de los resultados de rendimiento.

Tabla 5.8. Ratios y tiempos de inserción medios en modelos de 1000 a 5000 características.

| Tamaño FM | Tiempo (segundos) |
|---|---|
| 1000 | 7,98 |
| 2000 | 10,38 |
| 3000 | 12,61 |
| 4000 | 13,82 |
| 5000 | 16,07 |

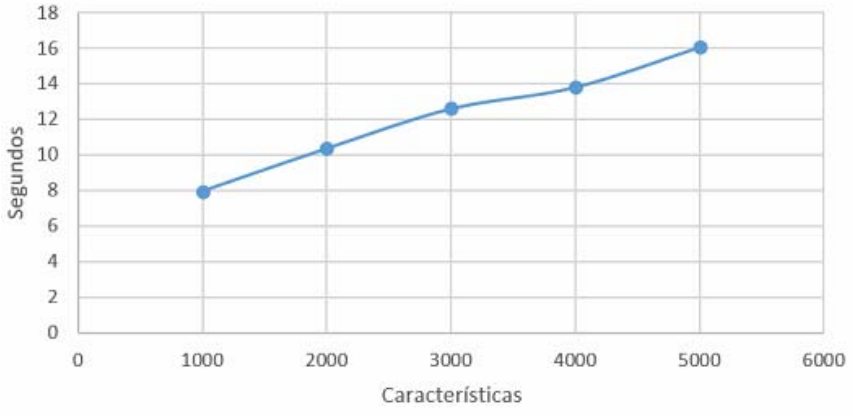



Figura 5.5. Progreso del tiempo medio con inserciones de 10 características.

Como se puede ver en la Figura 5.5, existe una progresión lineal que muestra un aumento del tiempo necesario para añadir una característica al modelo conforme este es más grande. En modelos de características cuya diferencia es de 1000 características más en cada uno, se puede ver que el aumento de tiempo para la inserción de 10 características es de aproximadamente 2 segundos adicionales respecto al anterior.

## 5.2 Simulación con variabilidad de un robot

Una vez que hemos realizado la simulación de nuestro algoritmo respecto a la eficacia y al rendimiento, vamos a describir en este apartado cómo se comporta nuestra solución integrada con un robot que necesita reconfigurarse dinámica. En primer lugar, pasamos a describir el robot utilizado como caso de estudio y posteriormente la descripción de cómo se ha realizado el experimento y sus resultados.

### 5.2.1 Descripción del robot

Para ello, vamos a utilizar un robot *TurtleBot* modelo *TurtleBot 2*, el cual se compone de un ordenador *Intel NUC* equipado con un procesador *Intel i5* con 8 GB de RAM, un sensor de distancia 2D/3D *Microsoft Kinect*, una batería y el kit para integrar los diferentes componentes de *TurtleBot*, tal como se muestra en la Figura 5.6. El robot utiliza como Sistema operativo Ubuntu 20.04 e incluye ROS Noetic.


Figura 5.6. Robot TurtleBot 2

Opcionalmente puede portar un brazo robótico para desempeñar ciertas acciones. *TurtleBot 2* es la evolución de la plataforma *iRobot Create*. Soporta la arquitectura ROS y sus aplicaciones principales son el uso en *Ambient Asisted Living* (AAL), en labores de investigación para actividades de localización y mapeo, manipulación móvil, y tele-operación.

El software del *TurtleBot* se compone de diferentes funcionalidades. En primer lugar, tenemos los aspectos de navegación de robot, que puede implementarse usando algoritmos de *Planificación* cuyo cometido es localizar una ruta sin obstáculos entre dos puntos del mapa. Dependiendo de la manera en que representemos el entorno para crear el mapa, podemos usar diferentes algoritmos tales como *Geometría*, basado en coordenadas espaciales que definen áreas libres de colisiones, o *Topología*, las cuales representan un área como un conjunto de nodos interconectados.

El problema de la planificación reside en la selección del algoritmo de búsqueda de grafos más adecuado (por ejemplo, *AMCL* está basado en una versión adaptada de algoritmo de localización de *Monte Carlo* el cual se usa en *TurtleBot* cuando se selecciona una navegación de tipo *Geometría*). *TurtleBot* tiene una característica de auto localización que puede estar también basada en *Geometría* o *Topología* al igual que la *Planificación*.

Otra funcionalidad importante del robot es el mecanismo de *detección de obstáculos*. Como usa un algoritmo de *seguimiento del contorno* que construye un mapa en base al recorrido que realiza el robot detectando los objetos presentes, *TurtleBot* utiliza una gran cantidad de *sensores* tales como: detección de superficies, detección de objetos, cálculo de distancia a los objetos o sensores de movimiento para detectar si hay otros robots u objetos moviéndose en el recinto y en qué dirección. Por tanto, el robot puede reaccionar a objetos que se mueven cercanos a él. El robot posee una función de mapeado para crear los mapas y que le ayuda a ubicarse y a encontrar salidas a otras áreas, y también incorpora un mapa geométrico llamado *GMapping,* que construye el mapa 2D geométrico en base al odómetro del robot, y otro *Topológico,* trazando la superficie como una serie de nodos interconectados.

## 5.2.2 Modelo de variabilidad del robot

A partir de las características software del punto anterior, se ha elaborado el árbol de variabilidad mostrado en la Figura 5.7, resultando un árbol que contiene 18 características, y 7 relaciones, siendo 5 de tipo AND, 1 de tipo OR y 1 de tipo XOR. A través de este modelo se explora la variabilidad de *TurtleBot*, describiendo cómo se moverá a través de la *Planificación*, cómo creará los mapas, a través del *Mapeado*, cómo se ubicará a sí mismo, a través de la *Auto-Localización* y cómo realizará las exploraciones de las áreas, a través de la *Exploración* y a qué tipos de señales y eventos reaccionará, a través de los *Sensores*.

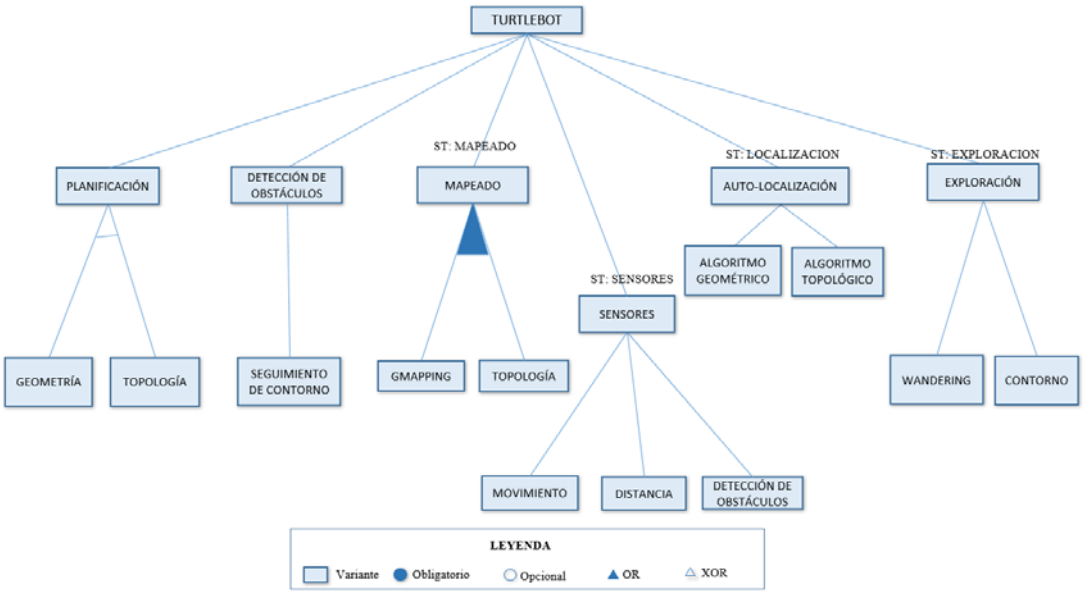



Figura 5.7. Modelo de variabilidad del robot

La lista de restricciones (los requerimientos y exclusiones entre características) que hemos identificado en el modelo es la mostrada en la Figura 5.8.

Wandering REQUIRE Detección de Obstáculos
Mapping REQUIRE Auto-Localización
Planificación REQUIRE Mapeado
Mapeado REQUIRE Sensores
Detección de Obstáculos REQUIRE Sensores
Geometría EXCLUDE Topología
Mapeado Geométrico EXCLUDE Mapeado Topológico
Wandering EXCLUDE Contorno
GMapping EXCLUDE Topología

Figura 5.8. Restricciones del Modelo de Variabilidad

Una vez que hemos descrito la representación del modelo de variabilidad del robot y las restricciones, pasamos a discutir cómo hemos realizado la integración entre nuestro algoritmo y la información que proviene del robot.

### 5.2.3 Integración de TurtleBot con RuVa

Con el fin de poder realizar la experimentación con TurtleBot y nuestro algoritmo de variabilidad dinámica RuVa, hemos tenido que realizar un preparatorio para conectar ambos sistemas. Para ello, es preciso comunicar el robot con el servidor que almacena nuestro algoritmo de manera que los cambios producidos por el robot se transmitan al algoritmo de variabilidad. En este sentido, es necesario disponer de un formato de datos común que permita a *RuVa* leer las actualizaciones que se produzcan en tiempo real en el robot.

Para comunicarse *TurtleBot* con *RuVa* hemos utilizado el protocolo *HTTP* con peticiones *GET* para que se puedan enviar los datos de reconfiguración de TurtleBot a un servidor *PHP* que recibe las peticiones del robot y se las transmite a *RuVa*. El formato de las peticiones enviadas a *RuVa* es como muestra la Figura 5.9

```
http://dominio/ruva.php?featureModel=<nombre-del-modelo>
&newFeature=<nueva-característica>
&st=<supertipo-nueva-característica>
```

Figura 5.9. Esquema de llamada URL de TurtleBot a RuVa

Donde el parámetro *featureModel* identifica el nombre del modelo de características utilizado, *newFeature* hace referencia a la nueva característica que se va a intentar insertar en el modelo y *st* hace referencia al supertipo de la característica a insertar. En la Figura 5.10 se muestra un ejemplo de petición a *RuVa*.

```
http://dominio/ruva.php?featureModel=TurtleBot
&newFeature=Octomap
&st=mapping
```

Figura 5.10. Ejemplo de llamada URL de TurtleBot a RuVa

## 5.2.4 Caso de estudio

Para el caso de estudio con *TurtleBot* hemos utilizado la planta baja de uno de los edificios de la *Universidad Rey Juan Carlos* consistente en dos áreas diferentes separadas con marcadores visuales que puede leer *TurtleBot*. Hemos utilizado el software *FloorPlaner*[11] para diseñar el mapa tal y como mostramos en la Figura 5.11. Con el fin de generar los mapas del recorrido realizado por *TurtleBot* se ha utilizado *ROS* (*Robot Operating System*) juntamente con *RViz*[12], herramienta de visualización de los mapas generados por ROS.

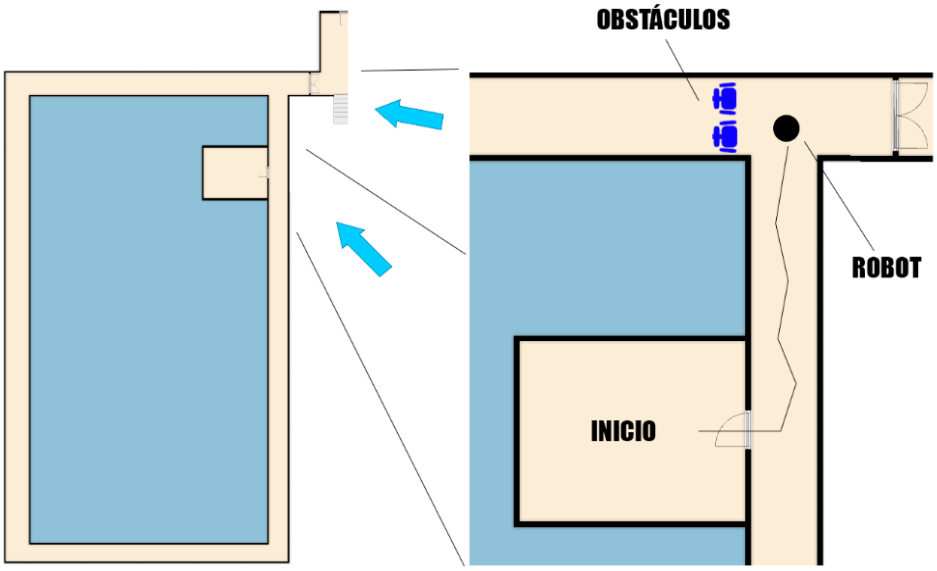



Figura 5.11. Planta del edificio Departamental II (URJC Móstoles)

El experimento comienza en la sala marcada como *INICIO* en la Figura 5.11. En esta sala se encuentra el robot, donde inicialmente tiene cargado un mapa para poder moverse por el área coloreada en color claro. El robot inicia el recorrido y decide girar hacia la izquierda y recorrer el pasillo. El mapa generado por *TurtleBot* así como el área que va recorriendo se pueden ver en la Figura 5.12.

[11] http://floorplanner.com
[12] http://wiki.ros.org/rviz

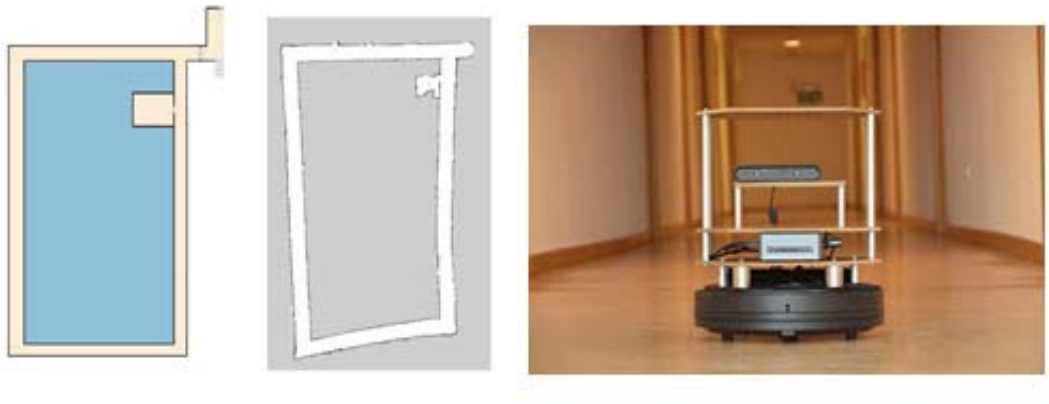


Figura 5.12. Mapa de la planta baja generado por TurtleBot

Tras recorrer algunos metros encuentra una bifurcación, donde puede ir a izquierda o derecha. Inicialmente el robot intentará ir hacia la izquierda ya que ese camino está contemplado en los mapas de los que dispone. Sin embargo, los sensores de los que dispone le indican que en el camino hay obstáculos que le impiden recorrer ese pasillo, por lo que esa ruta deja de ser una alternativa viable.

En la Figura 5.13 podemos ver tanto los obstáculos en la *zona 1* como las marcas que separan la *Zona 1* de la *Zona 2*.

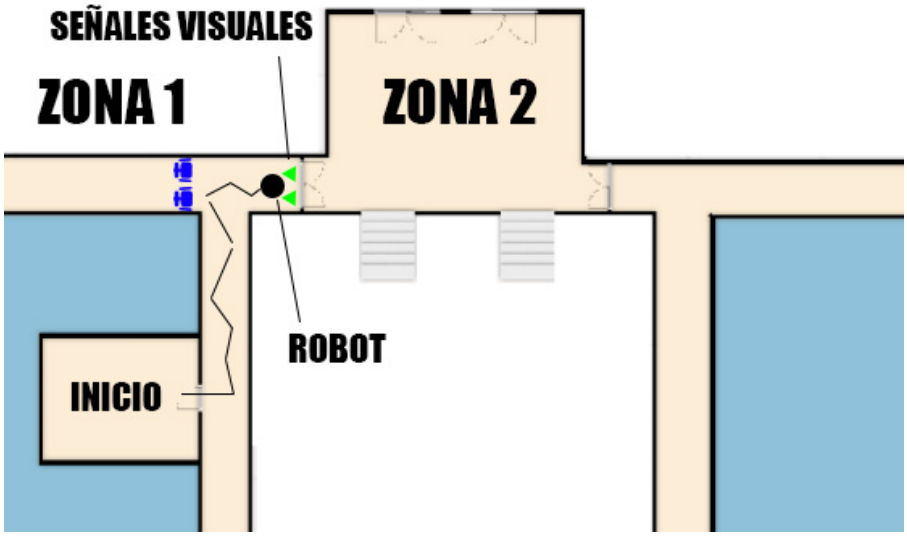



Figura 5.13. Llegada a las marcas visuales en suelo.

Dado que el robot se encuentra en modo exploración, va construyendo el mapa a medida que recorre la *Zona 1*. En un momento dado, el robot se encuentra un obstáculo a la izquierda de la zona y decide explorar la parte derecha del pasillo. En ese momento, al recorrer el inicio de ese pasillo detecta unas marcas visuales en el suelo que le indican un cambio de zona y dado que el robot debe construir un nuevo mapa, conecta con *RuVa* vía *HTTP* para enviarle los datos de la nueva característica. Podemos observar en la Figura 5.14 el paso a la nueva área llamada *Zona 2* a la que se ha movido el robot.

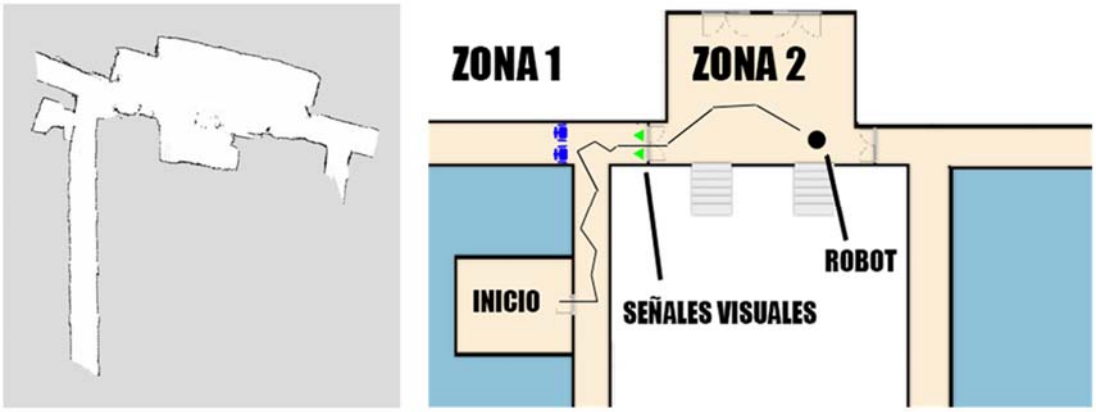



Figura 5.14. Mapa de la zona 2 incluyendo el mapa generado por TurtleBot.

Una vez que el *TurtleBot* cambia de la *Zona 1* a la *Zona 2* y se reconfigura construyendo el mapa de la nueva zona, establece una comunicación desde el servidor con RuVa enviando los datos de la nueva característica consistente en un nuevo mapa. Si RuVa procesa la solicitud satisfactoriamente, la nueva característica se agrega al modelo de variabilidad de acuerdo con el supertipo de la característica.

El modelo de variabilidad resultante que se muestra en la Figura 5.15 incluye la nueva característica añadida *Octomap* que se muestra en color naranja. Ejecutamos el caso de estudio dos veces y medimos el tiempo que dedica el robot a comunicar los cambios de contexto a RuVa. El tiempo promedio requerido por *TurtleBot* fue entre 1 y 1,2 segundos, incluido el tiempo para transmitir la solicitud a *RuVa* y el tiempo para reconfigurar el modelo de características, ya que se trata de un modelo de variabilidad muy pequeño.

En consecuencia, en aquellos casos en los que es posible agregar una nueva característica, la solución propuesta conectando un robot que comunique los cambios de contexto a un algoritmo de variabilidad en tiempo de ejecución permite reconfigurar el modelo de variabilidad del robot de forma dinámica sin intervención humana.

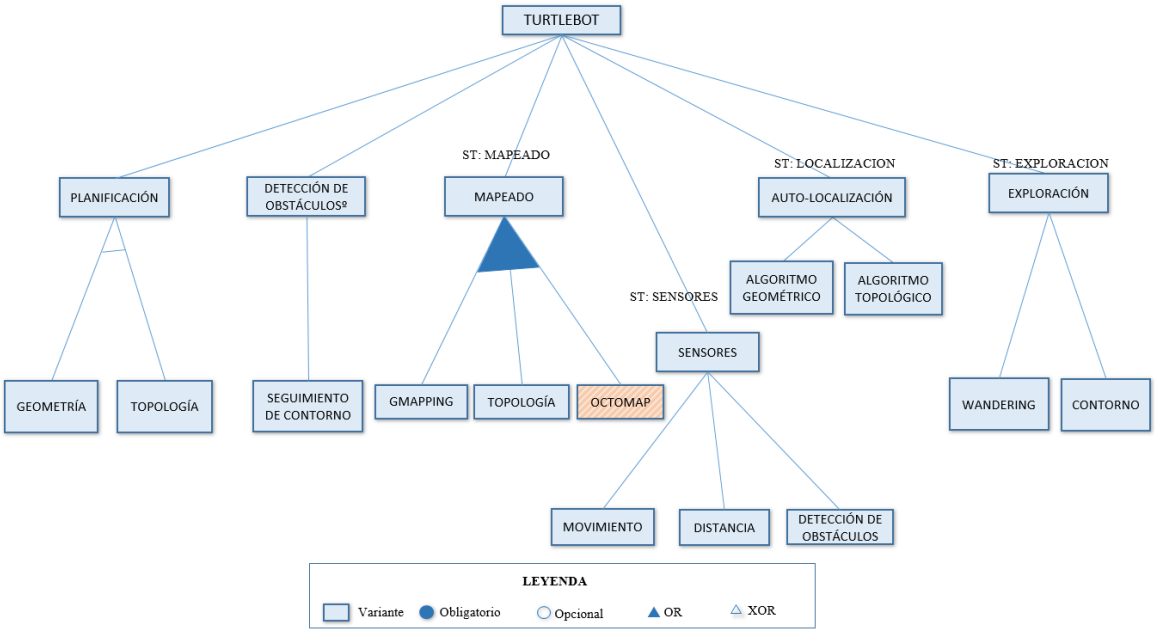



Figura 5.15. Modelo de variabilidad resultante después de insertar la nueva característica *Octomap*.

## 5.3 Conclusiones a la experimentación

Las conclusiones que podemos extraer en base a la experimentación realizada son las siguientes:

1. En primer lugar, el algoritmo de variabilidad de *runtime* se comporta de manera adecuada añadiendo nuevas características de manera dinámica a través de los supertipos definidos. Una selección apropiada de supertipos en las ramas oportunas en modelos reales de características puede facilitar las decisiones de los diseñadores de software para colocar las características en una ubicación adecuada. Además, el algoritmo sugiere la rama más adecuada según la cantidad de supertipos compatibles y la cantidad de características secundarias, evitando sobrecargar en exceso la misma rama. Por otra parte, la comprobación de restricciones en tiempo

de ejecución con un solver como *Choco* ayuda a detectar configuraciones incompatibles en modelos de características, ya que esta tarea de manera tradicional generalmente se realiza en modo *offline* y no durante *runtime*.

2. En segundo lugar, el algoritmo demostró ser altamente eficiente en la mayoría de los casos, incluso en modelos grandes de características. La selección de modelos de características generados aleatoriamente minimizó el sesgo en la creación de modelos de características *ad-hoc* para la simulación.

3. En tercer lugar, el uso de modelos grandes de características en las pruebas de rendimiento y escalabilidad demostró que el tiempo necesario para insertar diferentes características se mantiene bajo. Dado que redibujar el modelo de características no es una tarea crítica, los valores pueden considerarse más que aceptables en todos los casos. Sin embargo, estos valores pueden variar en función del tamaño del modelo de restricciones ya que son éstos precisamente los que consume más tiempo comprobar.

4. En cuarto lugar, el caso de estudio que utiliza un sistema real y dependiente del contexto como es un robot, demuestra la aplicabilidad de la solución propuesta para sistemas autónomos y sensibles al contexto que requieren algún tipo de reconfiguración. En esta experimentación integramos con éxito nuestra solución con un *TurtleBot* para redibujar un modelo de características en *runtime* en función de la información de los cambios de contexto proporcionada por el robot, agregando una nueva de forma dinámica. Como el modelo de características del robot es pequeño, el tiempo empleado para redibujar el modelo fue de alrededor de 1 segundo, incluido la llamada (aproximadamente 300 milisegundos) para enviar los datos a *RuVa*.

A pesar de que la solución propuesta resultó satisfactoria, existen algunas limitaciones en esta investigación. Una limitación aplica a la definición y ubicación de los supertipos. Esta es una tarea humana que puede influir en nuestros resultados, puesto que eligiendo un número diferente y haciendo una selección distinta de estos supertipos, es posible llegar a otros caminos de inserción de una característica o a un aumento de los resultados fallidos. En aquellos casos en los que una característica no se puede agregar automáticamente, es necesaria la intervención manual.

Otra limitación sería la realización de experimentos en sentido contrario, donde el diseñador incluye una nueva característica y el cambio se transmite al robot, aunque para ello puede ser necesario incluir un software adicional si la nueva característica no se encuentra almacenada en la memoria del robot.

En este caso, el tiempo necesario para la reconfiguración es importante, porque el sistema sensible al contexto debe reaccionar en poco tiempo, especialmente en sistemas críticos, o si un robot necesita descargar una función desde un servidor remoto o una nube. En sistemas críticos en tiempo real donde se requieren respuestas en términos de milisegundos, sugerimos usar nuestro enfoque sólo de la forma en que lo usamos en nuestro caso de estudio, teniendo en cuenta que depende del tamaño del modelo de variabilidad y el tiempo necesario para transmitir de los cambios.

Las conclusiones de esta experimentación destacan la importancia de los algoritmos de variabilidad dinámica para reconfigurar los modelos de variabilidad sobre la marcha, de acuerdo con los cambios de contexto. Dado que las soluciones de variabilidad en *runtime* son fundamentales para los enfoques de líneas de producto de software dinámicas (DSPL), nuestra solución integrada soluciona el problema entre los *solvers* de restricciones *offline* y los mecanismos de runtime requeridos por las soluciones de variabilidad dinámica. La adaptación de ambos formatos de datos entre el *solver* de *FaMa* y el algoritmo de variabilidad de *runtime*, así como la conexión entre el robot y el algoritmo, fueron claves para proporcionar una solución integrada capaz de modificar la variabilidad estructural en *runtime*. Nuestro enfoque puede ayudar también a reestructurar modelos grandes de características y reducir la carga de esfuerzo humano necesaria en el rediseño de variantes y que se complementa con un proceso semiautomático de gestión de puntos de variación en las nuevas características que se añaden.

# Capítulo 6. Conclusiones y Futuras Líneas de Investigación

Realizada la experimentación, en este capítulo exponemos las conclusiones extraídas del trabajo de investigación realizado, así como las propuestas de mejora y futuras líneas de investigación.

## 6.1 Conclusiones Generales

A lo largo de este trabajo hemos destacado la falta de mecanismos para reconfigurar los modelos de variabilidad de forma dinámica. De la solución aportada podemos arrojar las siguientes conclusiones:

- Un modelo integrado con un resolvedor de restricciones que proporciona la reconfiguración en tiempo de ejecución de un modelo de variabilidad cuando se añaden características nuevas y comprobando en tiempo real el modelo de restricciones.
- Un localizador que permite buscar el lugar óptimo para incluir una nueva característica basado en una comprobación de supertipos equivalentes y un optimizador para seleccionar la mejor rama posible del árbol de características.
- Una conexión del algoritmo propuesto con sistemas reales basados en el contexto que permiten comunicar los cambios de contexto y de nuevas funcionalidades y transmitirlas al algoritmo como datos de entrada para reconfigurar el modelo de variabilidad de forma autónoma.
- Una gestión semiautomática de puntos de variación que permite modelar ciertas relaciones lógicas de manera autónoma cuando se incluye una nueva característica.

## 6.2 Conclusiones a la Experimentación

Las conclusiones más visibles que podemos extraer sobre la experimentación realizada son las siguientes:

- Se han realizado pruebas de eficiencia para incluir características de manera satisfactoria con ratios que varían entre el 70 y el 95% de acuerdo con la topología de los modelos de variabilidad y a la distribución de supertipos, siendo los porcentajes de éxito más habituales en torno al 90%.
- Hemos simulado pruebas de escalabilidad y rendimiento con bajos valores de inserción para modelos grandes de características y cuyos resultados no sobrepasan los 16 segundos para modelos de 5.000 características generadas aleatoriamente y con diferentes juegos de pruebas de inserción.
- Hemos podido combinar de manera satisfactoria el algoritmo RuVa con el software de un robot para permitir la inclusión de una característica nueva en base a condiciones de contexto cambiantes. Como resultado de la prueba, se ha podido incluir la característica y reconfigurar el árbol de variabilidad del robot en menos de 2 segundos.
- Dado que el modelo de variabilidad del robot es pequeño, sería necesario realizar pruebas con diferentes sistemas basados en cambios en el contexto. Para sistemas en tiempo real no es un gran problema que los cambios en el modelo de variabilidad demoren algunos segundos, pero en el sentido inverso habría que evaluar el coste en tiempo de la transmisión de información de nuevas características teniendo lugar en un sistema autónomo o dependiente del contexto con requisitos de tiempo más exigentes.

## 6.3 Futuras Líneas de Investigación

Una vez completado nuestro trabajo, planteamos las futuras líneas de investigación:

- Búsqueda de nuevas formas de comprobar la inclusión de características sin utilizar supertipos predefinidos.
- Integración de nuestra solución con propuestas de líneas de producto software dinámicas y con algoritmos que se basan en el modelo *MAPE-K* para comparar su eficacia.
- Prueba de la reconfiguración de los sistemas desde el algoritmo hacia el sistema y medir los tiempos de respuesta.
- Análisis y clasificación de tipos de sistemas ciberfísicos y dependientes del contexto para ver cuáles son los más adecuados para utilizar algoritmos de variabilidad dinámica.
- Investigación de las topologías de modelos de variabilidad que son más adecuados y qué distribución y número de supertipos resulta más conveniente para alcanzar ratios de eficacia superiores al 90%.

# Capítulo 7. Bibliografía

**[ABBA 2011]** N. Abbas, J. Andersson, D. Weyns
Knowledge Evolution in Autonomic Software Product Lines. 15th International Software Product Line Conference (SPLC'11), Fifth International Workshop on Dynamic Software Product Lines (DSPL) ACM DL 36, 2011

**[ABBA 2015]** N. Abbas, J. Andersson
Harnessing variability in product-lines of self-adaptive software systems. 19th International Conference on Software Product Line (SPLC 2015), ACM DL 191-200, 2015

**[ABEL 2010]** A. Abele, Y. Papadopoulos, D. Servat, M. Törngren, M.Weber
The CVM framework—a prototype tool for compositional variability management. In: Variability Modelling of Software-Intensive Systems (VAMOS'10), ACM DL 101–105, 2010

**[ABMA 2014]** W. Aßmann, S. Götz, J-M., Jezéquel, B. Morin, M. Trapp, A Reference Architecture and Roadmap for Models@run.time Systems, Chapter in: Models@runtime Foundations, Applications and Roadmap Systems, LNCS 8378, 1-18, Springer, 2014

**[ACHE 2013]** M. Acher, P. Collet, P. Lahire, R. France
FAMILIAR: a domain specific language for large scale management of feature models. Science of Computer Programming 78, 657–681, 2013

**[ALI 2009]** R. Ali, R. Chitchyan, P., Giorgini
P. Context for goal-level product line derivation. In: Proceedings of 3rd Dynamic Software Product Lines (DSPL), Limerick, Ireland, 2009

**[ANAN 2020]** S. Ananieva, S. Greiner, T. Kühn, J. Krüger, L. Linsbauer, S. Grüner, T. Kehrer, H. Klare, A. Koziolek, H. Lönn, S. Krieter, C. Seidl, S. Ramesh, R.H. Reussner, B. Westfechtel
A conceptual model for unifying variability in space and time. SPLC '20: 24th ACM International Systems and Software Product Line Conference (SPLC (A)), ACM DL, 15:1-15:12, 2020

**[ANAN 2022]** S. Ananieva, S. Greiner, J. Krüger, L. Linsbauer, S. Grüner, T. Kehrer, T. Kühn, C. Seidl, S., R.H. Reussner
Unified Operations for Variability in Time and Space, 16th International Working Conference on Variability Modeling of Software-Intensive Systems (VAMOS 2022), 7:1-7:10, ACM DL, 2022

**[APPE 2013]** M. Appeltauer, R. Hirschfeld, J. Lincke
Declarative Layer Composition with the JCop Programming Language. Journal of Object Technology, 12(2), 1-37, 2013

**[AYAL 2016]** I. Ayala, J.M. Horcas, M. Amor, L. Fuentes
Using Models at Runtime to Adapt Self-managed Agents for the IoT Multiagent System Technologies - 14th German Conference (MATES 2016), LNCS 9872 Springer 155-173, 2016

**[BAK 2010]** K. Bak, K. Czarnecki, A. Wasowski
Feature and meta-models in Clafer: mixed, specialized, and coupled. Software Language Engineering (SLE'10), 102–122, 2010

**[BARE 2015]** L. Baresi, C. Quinton
Dynamically Evolving the Structural Variability of Dynamic Software Product Lines. SEAMS@ICSE, 57-63, 2015

**[BART 2012]** J. Bartholdt, D. Becker
Scope Extension of an Existing Product Line. 16th International Software Product Line Conference, 16th International Software Product Line Conference (SPLC '12), ACM DL 275-282, 2012

**[BASS 2012]** L. Bass, P. Clements, R. Kazman
Software Architecture in Practice (3 ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2012

**[BENA 2005]** D. Benavides, P. Trinidad, A. Ruiz-Cortés
Automated Reasoning on Feature Models, Advanced Information Systems Engineering, 17th International Conference (CAiSE 2005), LNCS 3250 Springer 491–503, 2005

**[BENA 2010]** D. Benavides, S. Segura, A. Ruiz-Cortés
Automated Analysis of Feature Models 20 Years Later: A Literature Review. Information Systems 35(6), 615-636, 2010

**[BENC 2012]** N. Bencomo, S. O. Hallsteinsen, and E. S. de Almeida, "A view of the dynamic software product line landscape," IEEE Computer, vol. 45(10), 36–41, 2012

**[BEEK 2019]** M.H. ter Beek, K. Schmid, H. Eichelberger
Textual variability modeling languages: an overview and considerations. 23rd International Systems and Software Product Line Conference, SPLC 2019, ACM DL, 82:1-82:7, 2019

**[BLAI 2009]** G. S. Blair, N. Bencomo, R. B. France, Models@ run.time. IEEE Computer 42(10), 22-27, 2009

**[BOSC 2000]** J. Bosch
Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. ACM Press/Addison-Wesley Publ. Co., New York, NY, USA, 2000

**[BOSC 2012]** J. Bosch, R. Capilla. Dynamic Variability in Software-Intensive Embedded System Families. IEEE Computer 45(10), 28-35, 2012

**[CAES 2019]** B. Caesar, M. Nieke, A. Köcher, C. Hildebrandt, C. Seidl, A. Fay, I. Scahefer
Context-sensitive reconfiguration of collaborative manufacturing systems. 9th IFAC Conference on Manufacturing Modelling, Management and Control, 331-336, 2019

**[CAMA 2013]** J. Cámara, P. Correia, R. de Lemos, D. Garlan, P. Gomes, B. R. Schmerl, R. Ventura
Evolving an adaptive industrial software system to use architecture-based self-adaptation. SEAMS 2013, IEEE DL, 13-22, 2013

**[CAPI 2001]** R. Capilla, J.C. Dueñas
Modelling Variability with Features in Distributed Architectures, 4th International Workshop on Software Product-Family *Engineering* (PFE '01), LNCS 2290 Springer 319-329, 2001

**[CAPI 2011]** R. Capilla, J. Bosch. The Promise and Challenge of Runtime Variability. IEEE Computer 44(12), 93-95, 2011

**[CAPI 2013]** R. Capilla, J. Bosch, K-C. Kang
Systems and Software Variability Management. Concepts, Tools and Experiences. Springer Verlag, 2013

**[CAPI 2014]** R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés, M. Hinchey. An Overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. Journal of Systems and Software, 91(5), 3-23, 2014

**[CAPI 2014A]** R. Capilla, O. Ortiz, M. Hinchey
Context Variability for Context-Aware Systems. IEEE Computer 47(2), 85-87, 2014

**[CAPI 2016]** R. Capilla, A. Valdezate, F.J. Díaz
A Runtime Variability Mechanism Based on Supertypes, 9th Dynamic Software Product Line Workshop, 10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems FAS*W, ACM DL, 2016

**[CAPI 2016A]** R. Capilla, J. Bosch
Dynamic Variability Management Supporting Operational Modes of a Power Plant Product Line, Tenth International Workshop on Variability, Modelling of Software-intensive Systems (VaMoS '16), ACM DL 49-56, 2016

**[CAPI 2018]** R. Capilla, J. Bosch, M. Hinchey
Cutting Edge Topics on Dynamic Variability. Software Technology: 10 years of Innovation in IEEE Computer. Wiley IEEE Press 247-270, 2018

**[CARD 2014]** N. Cardozo, W. de Meuter, K. Mens, S. González, P.Y. Orban
Features on demand. Eighth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '14), ACM DL 1-18, 2014

**[CETI 2009A]** C. Cetina, Ø. Haugen, X. Zhang, F. Fleurey, V. Pelechano
Strategies for variability transformation at run-time. 13th International Conference, SPLC 2009, ACM DL 61-70, 2009

**[CETI 2009B]** C. Cetina, P. Giner, J. Fons, V. Pelechano
Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. IEEE Computer 42(10), 37-43, 2009

**[CETI 2013]** C. Cetina, P. Giner, J. Fons, V. Pelechano. Prototyping Dynamic Software Product Lines to evaluate run-time reconfigurations. Science of Computer Programming 78(12), 2399-2413, 2013

**[CLASS 2011]** A. Classen, Q. Boucher, P. Heymans
A text-based approach to feature modelling: Syntax and semantics of TVL. Science of Computer Programming 76, 1130–1143, 2011

**[COLA 2016]** S. di Cola, K-K. Lau, C. Tran, C. Qian, R. Arshad, B. Ardyansiah
A Component Model for Defining Software Product Families with Explicit Variation Points. 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2016), IEEE DL, 79-84, 2016.

**[COST 2005]** P. Costanza, R. Hirschfeld
Language constructs for context-oriented programming: an overview of ContextL. Symposium on Dynamic Languages (DLS 2005) ACM DL 1-10, 2005

**[CZAR 2000]** K. Czarnecki, U. Eisenecker
Generative Programming: Methods, Tools, and Applications Addison-Wesley, Reading, MA, 2000

**[CZAR 2002]** K. Czarnecki, T. Bednasch, P. Unger, U. Eisenecker
Generative programming for embedded software: an industrial experience report. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS 2487, 156–172. Springer, 2002

**[CZAR 2004]** K. Czarnecki, S. Helsen, U. Eisenecker
Staged Configuration Using Feature Models in Software Product Lines, Software Product Lines, Third International Conference (SPLC 2004), LNCS 3154 Springer 266-283, 2004

**[CZAR 2005]** K. Czarnecki, S. Helsen, U. Eisenecker
Formalizing cardinality-based feature models and their specialization, Software Process Improvement and Practice 10(1), 7-29, 2005

**[DESM 2007]** B. Desmet, J. Vallejos, P. Costanza, W. de Meuter, T. D'Hont
Context-oriented Domain Analysis, CONTEXT 2007, Springer LNAI 4635, 178-191, 2007

**[DEUR 2002]** A. van Deursen, P. Klint
Domain-specific language design requires feature descriptions. Journal of Computing and Information Technology (10)1, 1–17, 2002

**[EICH 2015]** H. Eichelberger, K. Schmid
Mapping the design-space of textual variability modeling languages: a refined analysis. International Journal Software Tools Technology Transfer 17, Springer, 559-584, 2015

**[ELSN 2010]** C. Elsner, G. Botterweck, D. Lohman, W. Schroder-Preikschat Variability in Time - Product Line Variability and Evolution Revisited. 4th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '10) ACM DL 131-137, 2010

**[ERIK 2005]** M. Eriksson, M. Börstler, B. Jürgen, B. Kjell The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations, Software Product Line Conference (SPLC 2005) LNCS 3714, 33-44, Springer, 2005

**[FELF 2018]** A. Felfernig, R. Walter, J.A. Galindo, D. Benavides, S.P. Erdeniz, M. Atas, S.Reiterer Anytime diagnosis for reconfiguration. Journal of Intelligent Informatoon Systems 51(1), 161-182, 2018

**[FRIT 2002]** C. Fritsch, A. Lehn, T., Strohm Evaluating Variability Implementation Mechanisms. 2nd Workshop on Product Line Engineering – The Early Steps (PLEES), 2002

**[FROS 2009]** R. Froschauer, A. Zoitl, P. Grünbacher Development and adaptation of IEC61499 Automation and control applications with runtime variability models. In: 7th IEEE International Conference on Industrial Informatics (INDIN 2009), 905–910, 2009

**[GADE 2020]** R. Gadelha, L. Vieira, D. Monteiro, F. Vidal, P.H. Maia Scen@rist: an approach for verifying self-adaptive systems using runtime scenarios. Software Quality Journal, Springer, 2020

**[GAME 2011]** N. Gámez, L. Fuentes, M.A. Aragüez
Autonomic computing driven by feature models and architecture in FamiWare. In: 5th European Conference on Software Architecture (ECSA), LNCS 6903. Springer-Verlag 164–179, 2011

**[GAME 2012]** N. Gámez, J. Cubo, L. Fuentes, E. Pimentel
Configuring a Context-Aware Middleware for Wireless Sensor Networks. Sensors 12(7): 8544-8570 (2012)

**[GAME 2013]** N, Gámez, L. Fuentes
Architectural evolution of FamiWare using cardinality-based feature models. Information & Software Technology 55(3), 563-580, 2013

**[GARC 2019]** S. García, D. Strüber, D. Brugali, A. Di Fava, P. Schillinger, P. Pelliccione, T. Berger
Variability Modeling of Service Robots: Experiences and Challenges. 13th International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS 2019), ACM DL 1-6, 2019

**[GARL 2009]** D. Garlan, B. Schmerl, S.-W. Cheng, Software Architecture-Based Self-Adaptation. Autonomic Computing and Networking, Springer Science+Business Media, LLC, 31-55, 2009

**[GONZ 2013]** S. González, K. Mens, M. Colacioiu, W. Cazzola
Context Traits: dynamic behaviour adaptation through run-time trait recomposition. Intl. Conf. on Aspect- Oriented Software Development (AOSD'13). ACM Press, 2013

**[GRIS 1998]** M.L. Griss, J. Favaro, M. D'Alessandro
Integrating feature modeling with the RSEB, Fifth International Conference on Software Reuse. Fifth International Conference on Software Reuse, ICSR 1998, IEEE DL, 76-85, 1998

**[GROß 2015]** M. Große-rhode, M. Himsolt
The Variability Exchange Language. Fraunhofer FOCUS
Online: https://www.variability-exchange-language.org/, 2015

**[GURP 2001]** J. van Gurp, J. Bosch, M. Svahnberg
On the notion of variability in software product lines, Proceedings - Working IEEE/IFIP Conference on Software Architecture (WICSA 2001) IEEE DL 45-54, 2001

**[HALL 2008]** S. Hallsteinsen, M. Hinchey, S. Park, K. Schmid
Dynamic Software Product Lines, IEEE Computer 41(4), 93-95, 2008

**[HART 2008]** H. Hartmann, T. Trew
Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. Software Product Line Conference (SPLC'08), 12-21, IEEE CS, 2008

**[HAUG 2013]** Ø. Haugen, A. Wąsowski, K. Czarnecki
CVL: Common Variability Language, 17th International Software Product Line Conference (SPLC '13), ACM DL 277, 2013

**[HEIN 2000]** A. Hein, M. Schlick, R. Vinga-Martins
Applying Feature Models in Industrial Settings, Software Product Lines; Experiences and Research Directions, Proceedings of the First International Conference (SPLC 1) Kluwer 47-70, 2000

**[HELL 2009]** A. Helleboogh, D. Weyns, K. Schmid, T. Holvoet, K. Scheltfhout, W. van Betsbrugge
Adding variants on-the-fly: modeling meta-variability in dynamic software product lines. In: Proceedings of 3rd International Workshop on Dynamic Software Product Lines (DSPL 2009), San Francisco, CA, USA, 2009

**[HORC 2017]** J.M. Horcas, L. Fuentes
Extending the Common Variability Language (CVL) Engine. 21st International Systems and Software Product Line Conference, IEEE DL 32-37, 2017

**[HUEB 2008]** M.C. Huebscher, J.A. McCann
A Survey of Autonomic Computing—Degrees, Models, and Applications. ACM Computing Surveys 40(3), 1–28, 2008

**[INGL 2013]** J.F. Inglés-Romero, A. Lotz, C. Vicente-Chicote, C. Schlegel
Dealing with Run-Time Variability in Service Robotics: Towards a DSL for Non-Functional Properties. 3rd International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob-12). CoRR abs/1303.4296, 2013

**[JI 2015]** W. Ji, T. Berger, M. Antkiewicz, K. Czarnecki
Maintaining Feature Traceability with Embedded Annotations, 19th International Conference on Software Product Line (SPLC 2015), ACM DL 61-70, 2015

**[KÄKÖ 2007]** T. Käkölä, J.C. Dueñas (Eds)
Software Product Lines: Research Issues in Engineering and Management. Springer Science & Business Media, 2007

**[KAMI 2013]** T. Kamina, T. Aotani, H. Masuhara
A Unified Context Activation Mechanism. 5th International Workshop on Context-Oriented Programming (COP'13), Montpellier, France, 2013

**[KAMI 2014]** T. Kamina, T. Aotani, H. Masuhara, T. Kamai
Context-oriented software engineering: a modularity vision. MODULARITY, 85-98, 2014

**[KANG 1990]** K-C, Kang, S. Cohen, J.A. Hess, W.E. Novak, A. Spencer Peterson
Feature-Oriented Domain Analysis, Feasibility Study. SEI Technical Report CMU/SEI-90-TR-21, 1990

**[KANG 1998]** K-C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh
FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering 5, 143-168, 1998

**[KEPH 2003]** J.O. Kephart, D.M. Chess. The vision of autonomic computing. Computer 36(1), 41–50, 2003

**[KRIE 2019]** S. Krieter, T. Thiem, Th. Liech
Using Dynamic Software Product Lines to Implement Adaptive SGX-enabled Systems. 13th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS 2019, ACM DL, 9:1-9:9, 2019

**[LEE 2004]** K. Lee, K-C. Kang
Feature dependency analysis for product line component design, 8th International Conference on Software Reuse (ICSR'04), LNCS 3107, 69-85, Madrid, Spain. Springer, 2004

**[LEEJ 2008]** J. Lee, D. Muthig
Feature-oriented Analysis and Specification of Dynamic Product Reconfiguration. 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems (ICSR '08), Springer-Verlag, 154-165, 2008

**[LIAN 2012]** J. Liang
Solving Clafer Models with Choco. (GSDLab-TR 2012–12-30), 2012

**[MATS 2007]** Y. Matsumoto
A Guide for Management and Financial Controls of Product Lines. 11th International Software Product Line Conference (SPLC 2007), IEEE DL 163–70, 2007

**[MAUR 2016]** J. Mauro, M. Nieke, C., Seidl, C., I.C. Yu
Context aware reconfiguration in software product lines. Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS ’16, 41–48. ACM DL, 2016

**[MAUR 2018]** J. Mauro, M. Nieke, C., Seidl, C., I.C. Yu
Context-aware reconfiguration in evolving software product lines. Science of Computer Programming 163, 139-159, Elsevier, 2018

**[MAUR 2021]** J. Mauro
Anomaly detection in Context-aware Feature Models. 15th International Working Conference on Variability Modelling of Software-Intensive Systems, 1-9, ACML, 2021

**[MEND 2009]** M. Mendonca, M. Branco, D. Cowan
SPLOT: software product lines online tools. Object Oriented Programming Systems Languages and Applications (OOPSLA'09), 761–762, 2009

**[MENS 2016]** K. Mens, R. Capilla, N. Cardozo, B. Dumas
A taxonomy of context-aware software variability approaches. MODULARITY (Companion), ACM DL 119-124, 2016

**[MENS 2017]** K. Mens, B. Duhoux, N. Cardozo
Managing the Context Interaction Problem: A Classification and Design Space of Conflict Resolution Techniques in Dynamically Adaptive Software Systems, first International Conference on the Art, Science and Engineering of Programming (Programming 2017) ACM DL 1-6, 2017

**[MICH 2020]** G.K. Michelon
Evolving System Families in Space and Time. SPLC '20: 24th ACM International Systems and Software Product Line Conference (SPLC (B)) ACM DL 104-111, 2020

**[MORA 2021]** N. Moraes do Nascimento, P.S.C. Alencar, D.D. Cowan
Context-aware data analytics Variability in IoT Neural-based Systems, IEEE International Conference on Big Data, 3595-3600, IEEE, 2021

**[MORI 2009]** B. Morin, O.r Barais, J-M. Jézéquel, F. Fleurey, A. Solberg,
Models@ Run.time to Support Dynamic Adaptation. IEEE Computer 42(10), 44-51, 2009

**[MOUR 2013]** M.L. Mouronte, Ó. Ortiz, A.B. García, R. Capilla
Using dynamic software variability to manage wireless sensor and actuator networks. IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), IEEE DL 1171-1174, 2013

**[MURG 2015]** A. Murguzur, S. Trujillo, H. Linh Truong, S. Dustdar, Ó. Ortiz, G. Sagardui,
Run-Time Variability for Context-Aware Smart Workflows. IEEE Software 32(3),52-60, 2015

**[ORTI 2012]** Ó. Ortiz, A.B. García, R. Capilla, J. Bosch, M. Hinchey
Runtime variability for dynamic reconfiguration in wireless sensor network product lines. 16th International Software Product Line Conference (SPLC '12), ACM DL 143-150, 2012

**[PASC 2013]** G. G. Pascual, M. Pinto, L. Fuentes
Run-Time Support to Manage Architectural Variability Specified with CVL. Software Architecture - 7th European Conference, ECSA 2013, Springer LCNS 7957, 282-298, 2013

**[PASC 2015]** G. G. Pascual, M. Pinto, L. Fuentes
Self-adaptation of mobile systems driven by the Common Variability Language. Future Generation Computer Systems 47, 127-144, 2015

**[PERE 2018]** J.A. Pereira, S. Schulze, S. Krieter, M. Ribeiro, G. Saake
A Context-Aware Recommender System for Extended Software Product Line Configurations. 12th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS 2018, ACM DL 97-104, 2018

**[PINT 2015]** M. Pinto, N. Gámez, L. Fuentes, M. Amor, J.M. Horcas, I. Ayala
Dynamic Reconfiguration of Security Policies in Wireless Sensor Networks. Sensors 15(3), 5251-5280, 2015

**[POHL 2005]** K. Phol, G. Böckle, F. van der Linden
Software Product Line Engineering. Foundations, Principles, and Techniques, Springer-Verlag, 2005

**[QUIN 2015]** C. Quinton, R. Rabiser, M. Vierhauser, L. Baressi, P. Grünbacher
Evolution in Dynamic Software Product Lines: Challenges and Perspectives, 19th International Conference on Software Product Line, SPLC 2015, ACM DL, 126-130, 2015

**[QUIN 2020]** C. Quinton, R. Rabiser, M. Vierhauser, L. Baressi, P. Grünbacher, C. Schumayer
Evolution in Dynamic Software Product Lines, Journal of Software Evolution and Process. Wiley, 2020

**[RIEB 2002]** M. Riebisch, K. Böllert, D. Streitferdt, I. Philippow
Extending feature diagrams with UML multiplicities, 6th World Conference on Integrated Design & Process Technology (IDPT 2002), Pasadena, CA, USA, 23–27 June 2002

**[ROME 2022]** A. Romero-Garcés, R. Salles de Freitas, R. Marfil, C. Vicente-Chicote, J. Martínez, J.F. Inglés-Romero, A. Bandera
QoS metrics-in-the-loop for endowing runtime self-adaptation to robotic software architectures. Multimedia Tools and Applications 81(3), 3603-3628, Springer, 2022

**[ROSE 2011]** M. Rosenmüller, N. Siegmund, M. Pukall, S. Apel
Tailoring dynamic software product lines. Generative Programming and Component Engineering, 10th International Conference on Generative Programming and Component Engineering (GPCE 2011), ACM DL, 3-12, 2011

**[SALL 2013]** K. Saller, M. Lochau, I. Reimund
Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems. 17th International Software Product Line Conference co-located workshops (SPLC 2013 Workshops), ACM DL 106-113, 2013

**[SANO 2021]** E. Barros dos Santos, R.M.C. Andrade, I. de Sousa Santos
Runtime testing of context-aware variability in adaptive software. Information and Software Technology 131(3), 106482, Elsevier, 2021.

**[SANT 2016]** I.S. Santos, L.S. Rocha, P.A. Santos Neto, R.M.C. Andrade
Model Verification of Dynamic Software Product Lines. Proceedings of the 30th Brazilian Symposium on Software Engineering (SBES), ACM DL, 113, 122. 2016

**[SCHR 2013]** R. Schröter, T. Thüm, N. Siegmund, G. Saake
Automated analysis of dependent feature models. Variability Modelling of Software-intensive Systems (VaMoS'13), ACM DL 9:1–9:5, 2013

**[SCHU 2015]** M. Schulze, R: Hellebrand
Variability Exchange Language - A Generic Exchange Format for Variability Data. Software Engineering Workshops, CEUR Workshop Proceedings, 71–80, 2015

**[SEGU 2012]** S. Segura, J.A. Galindo, D. Benavides, J.A. Parejo, A. Ruiz Cortés
BeTTy: benchmarking and testing on the automated analysis of feature models. Sixth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2012), ACM DL 63-71, 2012

**[SEGU 2014]** S. Segura, J.A. Parejo, R.M. Hierons, D. Benavides, A. Ruiz-Cortés
Automated Generation of Computationally Hard Feature Models Using Evolutionary Algorithms. Expert Systems with Applications 41(8): 3975–3992, 2014

**[SOUS 2017]** G. Sousa, W. Rudametkin, L. Duchien
Extending Dynamic Software Product Lines with Temporal Constraints. 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017), IEEE DL 129-139, 2017

**[SPREY 2020]** J. Sprey, C. Sundermann, S. Krieter, J. Mauro, T. Thüm, I. Schaefer
SMT-Based Variability Analyses in FeatureIDE, 14th International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS 2020), ACM DL 6:1-6:9, 2020

**[TAIN 2016]** N. Taing, T. Springer, N. Cardozo, A. Schill
A Dynamic Instance Binding Mechanism Supporting Run-time Variability of Role-based Software Systems, MODULARITY (Companion), 137-142, 2016

**[USMA 2014]** M. Usman Ftikhar, D. Weyns
Assuring system goals under uncertainty with active formal models of self-adaptation. 36th International Conference on Software Engineering, ICSE '14, ICSE Companion 2014:ACM DL 604-605, 2014

**[VILL 2019]** Á. Villota, R. Mazo, C. Salinesi
The high-level variability language: an ontological approach. 23rd International Systems and Software Product Line Conference, SPLC 2019 SPLC (B), ACM DL, 84:1-84:8, 2019

**[YE 2005]** H. Ye, H. Liu
Approach to modelling feature variability and dependencies in software product lines. IEEE Proceedings Software 152(3), 101-109, 2005

# Anexo I. Ficheros de Comandos de RuVa

Este anexo describe la lista de comandos aceptados por nuestro algoritmo de variabilidad dinámica (RuVa) ya que no hemos desarrollado una interfaz gráfica. Además, se incluye un ejemplo de ficheros **".init**" que son programas que encadenan secuencias de comandos para realizar diferentes operaciones con los modelos de variabilidad.

## I.1 Estructura de los Ficheros de Comandos

El conjunto de comandos que admite RuVa, su estructura y su descripción se enumeran a continuación.

1. ***ADD_FEATURE (w,x,y,z)*** - Descripción

   Invoca la inserción de una característica en un modelo de características.

   w – nombre de la característica (FX1, FX2,…)
   x – MANDATORY/OPTIONAL
   y – AND/OR/XOR/nulo
   z – [SUPERTIPOS] Deben ir entre corchetes y separados por espacios. No hay límite máximo.

   El resultado puede ser positivo si se lleva a cabo la inserción, o negativo si no es posible la inserción automática.

2. ***CHECK_FOR_CHANGES (x,y)*** - Descripción

   Lee el fichero `http://www.pclandia.net/ruva/reconf.inf` y reconfigura el modelo de características con los cambios que encuentre en el fichero ***reconf.inf***. Si no encuentra el fichero o no puede leerlo, RuVa lo intenta de nuevo pasados "x" segundos. Si tras "y" intentos no ha conseguido leerlo, entonces continúa la ejecución.

3. ***CLS –*** Descripción

Inicializa a cero los contadores de éxitos y fracasos al insertar. Estos contadores se pueden visualizar con ***SUMARIZE***.

***4. GENERATE_ADDFEATURE_SET(w,x,y,z) -*** Descripción

Genera un juego de pruebas con operaciones de inserción con un número de elementos determinado (w), usando un determinado conjunto de supertipos (x) y un número de supertipos máximo que puede haber (y). Este conjunto de pruebas generado se guarda en un fichero (z).

w – Tamaño del conjunto
x – Conjunto de supertipos (de 1 hasta x) de donde elegir. (p.e. 9)
y – Conjunto máximo de supertipos que puede tener la característica (p.e. 3)
z – Fichero donde se deja el set de ADDFEATURES.

5. ***INCLUDE_INIT(x) –*** Descripción

Incluye los comandos de otro fichero ".init" Este fichero puede a su vez invocar otros ficheros .init, con la finalidad de poder organizar los comandos en rutinas y subrutinas.

x – Nombre y ruta del fichero .init

6. ***LOAD_FM (x) -*** Descripción

Carga un modelo de características en formato .fm o .xml. Antes de cargar el modelo se ejecuta **RESET**.

x – Nombre y ruta del fichero .fm/.xml

7. ***LOAD_FM_FAMA (x)*** - Descripción

   Carga un modelo de características .fm. Antes de cargar el modelo se ejecuta **RESET**.

   x – Nombre y ruta del fichero .fm

8. ***LOAD_FM_XML (x)*** - Descripción

   Carga un modelo de características .xml. Antes de cargar el modelo se ejecuta **RESET**.

   x – Nombre y ruta del fichero .xml

9. ***FLUSH_LOG –*** Descripción

   Vuelca el contenido en memoria de log al fichero designado en ***LOG_SET_NAME***.

10. ***LOG_ACCUMULATIVE –*** Descripción

    Establece el log en modo acumulativo. Si el fichero de log ya existiera previamente, respeta el contenido que tuviera y añade el nuevo contenido al final.

11. ***LOG_ABSOLUTE –*** Descripción

    Establece el log en modo absoluto. Si el fichero de log existe, lo vacía y añade lo nuevo.

12. ***LOG_CLEAR –*** Descripción

    Vacía el contenido *del* log.

13. ***LOG_SET_NAME (x)*** - Descripción

    Establece el *fichero* log donde se volcará el log.

    x – Nombre y ruta del fichero log

14. ***RECORD_TIME_START*** - Descripción

Establece el momento de inicio. Si ya estaba establecido de antes, se sobrescribe el valor.

15. ***RECORD_TIME_STOP*** - Descripción

Establece el momento de fin. Si ya estaba establecido de antes, se sobrescribe el valor.

16. ***RELOAD_FM*** - Descripción

Recarga el modelo de características previamente cargado.

***17.*** ***REMOVE_FEATURE (x)*** - Descripción

Elimina la característica dada en x.

***18.*** ***RESET*** – Descripción

Se reinician a cero todos los contadores y se vacía el log.

**19.** ***SHOW_FEATURE_LIST*** - Descripción

Muestra una lista ordenada de las características cargadas en el modelo.

**20.** ***SHOW_FEATURE_LIST_BY_LEVELS*** - Descripción

Muestra las características del modelo cargado ordenadas por niveles.

**21.** ***SHOW_FM*** - Descripción

Muestra el árbol de características en formato texto y lo vuelca en el log.

22. ***SHOW_FM_SCREEN*** - Descripción

Muestra el árbol de características en pantalla.

23. ***SHOW_TIME*** - Descripción

Inserta una línea en el log con la diferencia entre las variables START_TIME y STOP_TIME, que previamente tienen que haber sido establecidas en RECORD_START_TIME Y RECORD_STOP_TIME.

24. ***SHOW_TREE_VALUES*** - Descripción

Muestra una tabla con los nodos del árbol y los valores que tienen y lo vuelca en el log.

25. ***SHOW_BRANCHES*** – Descripción

Muestra las diferentes ramas del árbol agrupadas por características y lo vuelca en el log.

***26.*** ***SHOW_STATISTICS / STATISTICS*** – Descripción

Muestra estadísticas, números y ratios del árbol de características.

**27.** ***SUMARIZE*** - Descripción

Muestra un resumen de aciertos y descartes tras hacer realizado los intentos de inserción.

28. ***WRITE_LINE(x)*** - Descripción

Añade una línea al log con el texto especificado (x).

x – texto a imprimir

29. ***XML2FM (x,y)*** **-** Descripción

Transforma un fichero con un modelo de características en formato .xml a formato .fm.

x – Nombre y ruta del fichero .xml

y – Nombre y ruta del fichero .fm

# I.2 Ejemplos de Ficheros de Punto de Entrada

En este apartado se describe un ejemplo de fichero de entrada "**.init**" similar a los utilizados durante la experimentación. La funcionalidad de los comandos se muestra en la parte derecha de la figura, describiéndose sólo aquellas líneas más importantes:

```
01 log_set_name(fm1000.log)
02 cls
03 reset
04 load_fm(1000.fm)
05 show_statistics
06 record_time_start
07 add_feature(FX1,MANDATORY,AND,[ST1])
08 add_feature(FX2,MANDATORY,AND,[ST2 ST3])
09 add_feature(FX3,OPTIONAL,OR,[ST4 ST5])
10 record_time_stop
11 show_statistics
12 show_time
13 sumarize
14 show_tree_values
15 show_fm
16 cls
17 reset
```

```
04 Carga del modelo
05 Ver valores y detalles del modelo

07,08,09 Se añaden las características

12 Visionado del estado del modelo
13 Tiempo transcurrido durante inserción
14 Resumen resultados obtenidos
15 Visionado árbol modelo resultante
```

Figura I.1 Comandos del fichero de entrada a la izquierda y su descripción a la derecha

# Anexo II. Datos de Salida de la Ejecución de RuVa

Las ejecuciones de los programas en RuVa generan una salida en ficheros ".log" donde se almacena el resultado devuelto por los diferentes comandos procesados. Estos ficheros log permiten recoger los cambios aplicados al modelo de características objeto de las diferentes pruebas, así como comprobar el nivel de éxito de las inserciones aplicadas al modelo.

A continuación, se muestra el resultado de cargar un modelo de 100 características en RuVa, la inserción de un conjunto de 10 características en dicho modelo, y el resultado de cada intento de inserción. Posteriormente, se muestra el tiempo invertido en el proceso de inserciones, y se vuelcan los datos de cada nodo. Por último, se muestra el modelo de características resultante tras la inserción del conjunto de características.

Fichero Log

```
Processing: D:/workspace/201903/features/fm100/set10.init
Path log: D:/workspace/201903/features/fm100/logs/set10.log
Loaded D:/workspace/201903/features/fm100/0100.fm
==========================================
STATISTICS
-----------
Features: 100
Breeding features: 19
Childless features: 81
Nodes AND: 17 (58,62%)
Nodes OR: 5 (17,24%)
Nodes XOR: 7 (24,14%)
Nodes Mandatory: 69 (69%)
Nodes Optional: 31 (31%)
Depth tree: 6
Average Branching Factor: 1,23
Maximum Branching Factor: 10
Maximum number of Nodes in SET relationship: 7
Supertypes: 9
Nodes with supertypes: 10
Max supertypes in a node: 2
Constraints: 5
REQUIRE Constraints: 3
EXCLUDE Constraints: 2
---------------------
FX1 - adding - |RT->MANDATORY |RS->AND |ST->ST2 ST4
FX1 - adding - 1 SUCCESS - Feature FX1 inserted under F4 - Solution:1/1
FX2 - adding - |RT->MANDATORY |RS->AND |ST->ST8
FX2 - adding - 1 SUCCESS - Feature FX2 inserted under root - Solution:1/1
FX3 - adding - |RT->MANDATORY |RS->AND |ST->ST2
FX3 - adding - 1 SUCCESS - Feature FX3 inserted under F4 - Solution:1/2
FX4 - adding - |RT->MANDATORY |RS->AND |ST->ST4
FX4 - adding - 1 SUCCESS - Feature FX4 inserted under F4 - Solution:1/2
FX5 - adding - |RT->OPTIONAL |RS->OR |ST->ST3 ST10
FX5 - adding - 1 SUCCESS - Feature FX5 inserted under TO-F69_F70_F71_F72-SET -
Solution:1/2
```

```
FX6 - adding - |RT->OPTIONAL |RS->XOR |ST->ST3
FX6 - adding - 1 SUCCESS - Feature FX6 inserted under FX5 - Solution:2/2
FX7 - adding - |RT->MANDATORY |RS->AND |ST->ST1 ST9
FX7 - adding - 1 SUCCESS - Feature FX7 inserted under F16 - Solution:1/5
FX8 - adding - |RT->MANDATORY |RS->AND |ST->ST7
FX8 - adding - 1 SUCCESS - Feature FX8 inserted under F3 - Solution:1/1
FX9 - adding - |RT->OPTIONAL |RS->OR |ST->ST5 ST2
FX9 - adding - 1 SUCCESS - Feature FX9 inserted under TO-F69_F70_F71_F72-SET -
Solution:1/7
FX10 - adding - |RT->MANDATORY |RS->AND |ST->ST6
FX10 - adding - 1 SUCCESS - Feature FX10 inserted under F11 - Solution:1/1
================================================================================
INSERTION TIME USED: 7,77 second(s)
TOTAL: 10
SUCCESS: 10
FAILURES: 0
================================================================================
ID  |NAME                                |RS |RT |TYPE |IDPA|PARENT
|TOTS |TOTC |MIN C|MAX C|TAND |TOR  |TXOR |LEVEL|SUPERTYPES     |S|
--------------------------------------------------------------------------------
------------------------------------------------
1   |root                                |AND|M  |BASIC|0   |
|0    |9    |0    |0    |1    |0    |0    |0    |               |0|
2   |F1                                  |AND|O  |BASIC|1   |
|0    |0    |0    |0    |0    |0    |0    |1    |               |0|
3   |F2                                  |AND|O  |BASIC|1   |
|0    |10   |0    |0    |1    |1    |1    |1    |               |0|
4   |F3                                  |AND|M  |BASIC|1   |
|0    |1    |0    |0    |1    |0    |0    |1    |ST7            |1|
5   |F4                                  |AND|O  |BASIC|1   |
|0    |1    |0    |0    |1    |0    |0    |1    |ST2,ST4        |2|
6   |F5                                  |AND|O  |BASIC|1   |
|0    |0    |0    |0    |0    |0    |0    |1    |               |0|
7   |F6                                  |AND|M  |BASIC|1   |
|0    |8    |0    |0    |1    |1    |0    |1    |               |0|
8   |F7                                  |AND|M  |BASIC|1   |
|0    |6    |0    |0    |1    |0    |1    |1    |ST5            |1|
9   |F8                                  |AND|M  |BASIC|1   |
|0    |8    |0    |0    |1    |0    |1    |1    |ST10           |1|
10  |F9                                  |AND|O  |BASIC|1   |
|0    |5    |0    |0    |1    |0    |1    |1    |ST1,ST5        |2|
11  |F28                                 |AND|M  |BASIC|3   |
|0    |7    |0    |0    |0    |0    |1    |2    |ST1            |1|
12  |F29                                 |AND|O  |BASIC|3   |
|0    |0    |0    |0    |0    |0    |0    |2    |               |0|
13  |F30                                 |AND|O  |BASIC|3   |
|0    |0    |0    |0    |0    |0    |0    |2    |               |0|
14  |F31                                 |AND|M  |BASIC|3   |
|0    |0    |0    |0    |0    |0    |0    |2    |               |0|
15  |TO-F32_F33_F34-SET                  |OR |M  |SET  |3   |
|0    |3    |1    |3    |0    |0    |0    |2    |               |0|
16  |F32                                 |OR |M  |BASIC|15  |
|0    |10   |0    |0    |1    |1    |0    |3    |ST3,ST5        |2|
17  |F33                                 |OR |M  |BASIC|15  |
|0    |0    |0    |0    |0    |0    |0    |3    |               |0|
18  |F34                                 |OR |M  |BASIC|15  |
|0    |0    |0    |0    |0    |0    |0    |3    |               |0|
19  |TO-F35_F36_F37-SET                  |XOR|M  |SET  |3   |
|0    |3    |1    |1    |0    |0    |0    |2    |               |0|
20  |F35                                 |XOR|M  |BASIC|19  |
|0    |4    |0    |0    |1    |1    |0    |3    |               |0|
21  |F36                                 |XOR|M  |BASIC|19  |
|0    |1    |0    |0    |1    |0    |0    |3    |               |0|
22  |F37                                 |XOR|M  |BASIC|19  |
|0    |0    |0    |0    |0    |0    |0    |3    |               |0|
23  |F27                                 |AND|M  |BASIC|4   |
|0    |3    |0    |0    |0    |0    |1    |2    |               |0|
24  |F91                                 |AND|O  |BASIC|5   |
```

```
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
25  |F10                                      |AND|O  |BASIC|7   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
26  |F11                                      |AND|O  |BASIC|7   |
|0    |3    |0    |0    |1    |0    |0    |2    |ST1,ST6         |2|
27  |F12                                      |AND|M  |BASIC|7   |
|0    |2    |0    |0    |1    |0    |0    |2    |ST9             |1|
28  |F13                                      |AND|M  |BASIC|7   |
|0    |3    |0    |0    |1    |0    |0    |2    |                |0|
29  |F14                                      |AND|M  |BASIC|7   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
30  |F15                                      |AND|M  |BASIC|7   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
31  |TO-F16_F17-SET                           |OR |M  |SET  |7   |
|0    |2    |1    |2    |0    |0    |0    |2    |                |0|
32  |F16                                      |OR |M  |BASIC|31  |
|0    |9    |0    |0    |1    |1    |1    |3    |ST5,ST9         |2|
33  |F17                                      |OR |M  |BASIC|31  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
34  |F38                                      |AND|O  |BASIC|8   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
35  |F39                                      |AND|M  |BASIC|8   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
36  |F40                                      |AND|M  |BASIC|8   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
37  |TO-F41_F42_F43-SET                       |XOR|M  |SET  |8   |
|0    |3    |1    |1    |0    |0    |0    |2    |                |0|
38  |F41                                      |XOR|M  |BASIC|37  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
39  |F42                                      |XOR|M  |BASIC|37  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
40  |F43                                      |XOR|M  |BASIC|37  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
41  |F55                                      |AND|O  |BASIC|9   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
42  |F56                                      |AND|O  |BASIC|9   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
43  |F57                                      |AND|M  |BASIC|9   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
44  |F58                                      |AND|M  |BASIC|9   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
45  |F59                                      |AND|M  |BASIC|9   |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
46  |TO-F60_F61_F62-SET                       |XOR|M  |SET  |9   |
|0    |3    |1    |1    |0    |0    |0    |2    |                |0|
47  |F60                                      |XOR|M  |BASIC|46  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
48  |F61                                      |XOR|M  |BASIC|46  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
49  |F62                                      |XOR|M  |BASIC|46  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
50  |F78                                      |AND|M  |BASIC|10  |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
51  |F79                                      |AND|O  |BASIC|10  |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
52  |F80                                      |AND|O  |BASIC|10  |
|0    |0    |0    |0    |0    |0    |0    |2    |                |0|
53  |TO-F81_F82-SET                           |XOR|M  |SET  |10  |
|0    |2    |1    |1    |0    |0    |0    |2    |                |0|
54  |F81                                      |XOR|M  |BASIC|53  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
55  |F82                                      |XOR|M  |BASIC|53  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
56  |TO-F94_F95_F96_F97_F98_F99_F100-SET|XOR|M  |SET  |11  |
|0    |7    |1    |1    |0    |0    |0    |3    |                |0|
57  |F94                                      |XOR|M  |BASIC|56  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
58  |F95                                      |XOR|M  |BASIC|56  |
```

```
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
59  |F96                                    |XOR|M  |BASIC|56  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
60  |F97                                    |XOR|M  |BASIC|56  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
61  |F98                                    |XOR|M  |BASIC|56  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
62  |F99                                    |XOR|M  |BASIC|56  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
63  |F100                                   |XOR|M  |BASIC|56  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
64  |F63                                    |AND|M  |BASIC|16  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
65  |F64                                    |AND|O  |BASIC|16  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
66  |F65                                    |AND|O  |BASIC|16  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
67  |F66                                    |AND|O  |BASIC|16  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
68  |F67                                    |AND|M  |BASIC|16  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
69  |F68                                    |AND|M  |BASIC|16  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
70  |TO-F69_F70_F71_F72-SET                 |OR |M  |SET  |16  |
|0    |4    |1    |4    |0    |0    |0    |4    |                |0|
71  |F69                                    |OR |M  |BASIC|70  |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
72  |F70                                    |OR |M  |BASIC|70  |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
73  |F71                                    |OR |M  |BASIC|70  |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
74  |F72                                    |OR |M  |BASIC|70  |
|0    |5    |0    |0    |1    |0    |0    |5    |                |0|
75  |F83                                    |AND|M  |BASIC|20  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
76  |F84                                    |AND|M  |BASIC|20  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
77  |TO-F92_F93-SET                         |OR |M  |SET  |20  |
|0    |2    |1    |2    |0    |0    |0    |4    |                |0|
78  |F92                                    |OR |M  |BASIC|77  |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
79  |F93                                    |OR |M  |BASIC|77  |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
80  |F88                                    |AND|M  |BASIC|21  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
81  |TO-F52_F53_F54-SET                     |XOR|M  |SET  |23  |
|0    |3    |1    |1    |0    |0    |0    |3    |                |0|
82  |F52                                    |XOR|M  |BASIC|81  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
83  |F53                                    |XOR|M  |BASIC|81  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
84  |F54                                    |XOR|M  |BASIC|81  |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
85  |F46                                    |AND|M  |BASIC|26  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
86  |F47                                    |AND|M  |BASIC|26  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
87  |F48                                    |AND|O  |BASIC|26  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
88  |F44                                    |AND|O  |BASIC|27  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
89  |F45                                    |AND|M  |BASIC|27  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
90  |F49                                    |AND|O  |BASIC|28  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
91  |F50                                    |AND|O  |BASIC|28  |
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
92  |F51                                    |AND|M  |BASIC|28  |
```

```
|0    |0    |0    |0    |0    |0    |0    |3    |                |0|
93  |F18                                  |AND|O  |BASIC|32   |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
94  |F19                                  |AND|M  |BASIC|32   |
|0    |5    |0    |0    |1    |0    |0    |4    |                |0|
95  |F20                                  |AND|M  |BASIC|32   |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
96  |F21                                  |AND|O  |BASIC|32   |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
97  |F22                                  |AND|O  |BASIC|32   |
|0    |0    |0    |0    |0    |0    |0    |4    |                |0|
98  |TO-F23_F24-SET                       |OR |M  |SET  |32   |
|0    |2    |1    |2    |0    |0    |0    |4    |                |0|
99  |F23                                  |OR |M  |BASIC|98   |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
100 |F24                                  |OR |M  |BASIC|98   |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
101 |TO-F25_F26-SET                       |XOR|M  |SET  |32   |
|0    |2    |1    |1    |0    |0    |0    |4    |                |0|
102 |F25                                  |XOR|M  |BASIC|101  |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
103 |F26                                  |XOR|M  |BASIC|101  |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
104 |F73                                  |AND|M  |BASIC|74   |
|0    |0    |0    |0    |0    |0    |0    |6    |                |0|
105 |F74                                  |AND|O  |BASIC|74   |
|0    |0    |0    |0    |0    |0    |0    |6    |                |0|
106 |F75                                  |AND|O  |BASIC|74   |
|0    |0    |0    |0    |0    |0    |0    |6    |                |0|
107 |F76                                  |AND|O  |BASIC|74   |
|0    |0    |0    |0    |0    |0    |0    |6    |                |0|
108 |F77                                  |AND|M  |BASIC|74   |
|0    |0    |0    |0    |0    |0    |0    |6    |                |0|
109 |F85                                  |AND|O  |BASIC|94   |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
110 |F86                                  |AND|M  |BASIC|94   |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
111 |F87                                  |AND|M  |BASIC|94   |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
112 |F89                                  |AND|O  |BASIC|94   |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
113 |F90                                  |AND|O  |BASIC|94   |
|0    |0    |0    |0    |0    |0    |0    |5    |                |0|
114 |FX1                                  |AND|M  |BASIC|5    |
|0    |0    |0    |0    |0    |0    |0    |2    |ST2,ST4         |2|
115 |FX2                                  |AND|M  |BASIC|1    |
|0    |0    |0    |0    |0    |0    |0    |1    |ST8             |1|
116 |FX3                                  |AND|M  |BASIC|5    |
|0    |0    |0    |0    |0    |0    |0    |2    |ST2             |1|
117 |FX4                                  |AND|M  |BASIC|5    |
|0    |0    |0    |0    |0    |0    |0    |2    |ST4             |1|
118 |FX5                                  |OR |O  |BASIC|70   |
|0    |0    |0    |0    |0    |0    |0    |4    |ST3,ST10        |2|
119 |FX6                                  |AND|O  |BASIC|118  |
|0    |0    |0    |0    |0    |0    |0    |5    |ST3             |1|
120 |FX7                                  |AND|M  |BASIC|32   |
|0    |0    |0    |0    |0    |0    |0    |4    |ST1,ST9         |2|
121 |FX8                                  |AND|M  |BASIC|4    |
|0    |0    |0    |0    |0    |0    |0    |2    |ST7             |1|
122 |FX9                                  |OR |O  |BASIC|70   |
|0    |0    |0    |0    |0    |0    |0    |4    |ST5,ST2         |2|
123 |FX10                                 |AND|M  |BASIC|26   |
|0    |0    |0    |0    |0    |0    |0    |3    |ST6             |1|
================================================================================
================================
----------------------------
--- BEGIN FEATURE MODEL ---
|
```

```
|--> root TYPE=BASIC PR=M SR=AND ST:[] Level: 0
|----> F1 TYPE=BASIC PR=O SR=AND ST:[] Level: 1
|----> F2 TYPE=BASIC PR=O SR=AND ST:[] Level: 1
|------> F28 TYPE=BASIC PR=M SR=AND ST:[ST1] Level: 2
|--------> TO-F94_F95_F96_F97_F98_F99_F100-SET TYPE=SET PR=M SR=XOR ST:[]//
Level: 3
|----------> F94 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F95 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F96 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F97 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F98 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F99 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F100 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|------> F29 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F30 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F31 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> TO-F32_F33_F34-SET TYPE=SET PR=M SR=OR ST:[]// Level: 2
|--------> F32 TYPE=BASIC PR=M SR=OR ST:[ST3, ST5] Level: 3
|----------> F63 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> F64 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F65 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F66 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F67 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> F68 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> TO-F69_F70_F71_F72-SET TYPE=SET PR=M SR=OR ST:[]// Level: 4
|------------> F69 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|------------> F70 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|------------> F71 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|------------> F72 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|--------------> F73 TYPE=BASIC PR=M SR=AND ST:[] Level: 6
|--------------> F74 TYPE=BASIC PR=O SR=AND ST:[] Level: 6
|--------------> F75 TYPE=BASIC PR=O SR=AND ST:[] Level: 6
|--------------> F76 TYPE=BASIC PR=O SR=AND ST:[] Level: 6
|--------------> F77 TYPE=BASIC PR=M SR=AND ST:[] Level: 6
|------------> FX5 TYPE=BASIC PR=O SR=OR ST:[ST3, ST10] Level: 4
|--------------> FX6 TYPE=BASIC PR=O SR=AND ST:[ST3] Level: 5
|------------> FX9 TYPE=BASIC PR=O SR=OR ST:[ST5, ST2] Level: 4
|--------> F33 TYPE=BASIC PR=M SR=OR ST:[] Level: 3
|--------> F34 TYPE=BASIC PR=M SR=OR ST:[] Level: 3
|------> TO-F35_F36_F37-SET TYPE=SET PR=M SR=XOR ST:[]// Level: 2
|--------> F35 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|----------> F83 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> F84 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> TO-F92_F93-SET TYPE=SET PR=M SR=OR ST:[]// Level: 4
|------------> F92 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|------------> F93 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|--------> F36 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|----------> F88 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|--------> F37 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|----> F3 TYPE=BASIC PR=M SR=AND ST:[ST7] Level: 1
|------> F27 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|--------> TO-F52_F53_F54-SET TYPE=SET PR=M SR=XOR ST:[]// Level: 3
|----------> F52 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F53 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F54 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|------> FX8 TYPE=BASIC PR=M SR=AND ST:[ST7] Level: 2
|----> F4 TYPE=BASIC PR=O SR=AND ST:[ST2, ST4] Level: 1
|------> F91 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> FX1 TYPE=BASIC PR=M SR=AND ST:[ST2, ST4] Level: 2
|------> FX3 TYPE=BASIC PR=M SR=AND ST:[ST2] Level: 2
|------> FX4 TYPE=BASIC PR=M SR=AND ST:[ST4] Level: 2
|----> F5 TYPE=BASIC PR=O SR=AND ST:[] Level: 1
|----> F6 TYPE=BASIC PR=M SR=AND ST:[] Level: 1
|------> F10 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F11 TYPE=BASIC PR=O SR=AND ST:[ST1, ST6] Level: 2
|--------> F46 TYPE=BASIC PR=M SR=AND ST:[] Level: 3
|--------> F47 TYPE=BASIC PR=M SR=AND ST:[] Level: 3
|--------> F48 TYPE=BASIC PR=O SR=AND ST:[] Level: 3
```

```
|--------> FX10 TYPE=BASIC PR=M SR=AND ST:[ST6] Level: 3
|------> F12 TYPE=BASIC PR=M SR=AND ST:[ST9] Level: 2
|--------> F44 TYPE=BASIC PR=O SR=AND ST:[] Level: 3
|--------> F45 TYPE=BASIC PR=M SR=AND ST:[] Level: 3
|------> F13 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|--------> F49 TYPE=BASIC PR=O SR=AND ST:[] Level: 3
|--------> F50 TYPE=BASIC PR=O SR=AND ST:[] Level: 3
|--------> F51 TYPE=BASIC PR=M SR=AND ST:[] Level: 3
|------> F14 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> F15 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> TO-F16_F17-SET TYPE=SET PR=M SR=OR ST:[]// Level: 2
|--------> F16 TYPE=BASIC PR=M SR=OR ST:[ST5, ST9] Level: 3
|----------> F18 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F19 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|------------> F85 TYPE=BASIC PR=O SR=AND ST:[] Level: 5
|------------> F86 TYPE=BASIC PR=M SR=AND ST:[] Level: 5
|------------> F87 TYPE=BASIC PR=M SR=AND ST:[] Level: 5
|------------> F89 TYPE=BASIC PR=O SR=AND ST:[] Level: 5
|------------> F90 TYPE=BASIC PR=O SR=AND ST:[] Level: 5
|----------> F20 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> F21 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F22 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> TO-F23_F24-SET TYPE=SET PR=M SR=OR ST:[]// Level: 4
|------------> F23 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|------------> F24 TYPE=BASIC PR=M SR=OR ST:[] Level: 5
|----------> TO-F25_F26-SET TYPE=SET PR=M SR=XOR ST:[]// Level: 4
|------------> F25 TYPE=BASIC PR=M SR=XOR ST:[] Level: 5
|------------> F26 TYPE=BASIC PR=M SR=XOR ST:[] Level: 5
|----------> FX7 TYPE=BASIC PR=M SR=AND ST:[ST1, ST9] Level: 4
|--------> F17 TYPE=BASIC PR=M SR=OR ST:[] Level: 3
|----> F7 TYPE=BASIC PR=M SR=AND ST:[ST5] Level: 1
|------> F38 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F39 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> F40 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> TO-F41_F42_F43-SET TYPE=SET PR=M SR=XOR ST:[]// Level: 2
|--------> F41 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|--------> F42 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|--------> F43 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|----> F8 TYPE=BASIC PR=M SR=AND ST:[ST10] Level: 1
|------> F55 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F56 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F57 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> F58 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> F59 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> TO-F60_F61_F62-SET TYPE=SET PR=M SR=XOR ST:[]// Level: 2
|--------> F60 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|--------> F61 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|--------> F62 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|----> F9 TYPE=BASIC PR=O SR=AND ST:[ST1, ST5] Level: 1
|------> F78 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> F79 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F80 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> TO-F81_F82-SET TYPE=SET PR=M SR=XOR ST:[]// Level: 2
|--------> F81 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|--------> F82 TYPE=BASIC PR=M SR=XOR ST:[] Level: 3
|----> FX2 TYPE=BASIC PR=M SR=AND ST:[ST8] Level: 1
|
---- END FEATURE MODEL ----
```

A continuación describiremos cada sección por separado para explicar el contenido de cada una de ellas.

En la Figura II.1 se muestra el nombre del fichero de Punto de Entrada usado por RuVa, así como el lugar donde RuVa irá dejando el log generado (el mismo enumerado anteriormente como *Fichero Log*), y a continuación, muestra la confirmación al comando LOAD_FM, mostrando el nombre del modelo de características cargado con el que se trabajará a continuación.

```
Processing: D:/workspace/201903/features/fm100/set10.init
Path log: D:/workspace/201903/features/fm100/logs/set10.log
Loaded D:/workspace/201903/features/fm100/0100.fm
```

Figura II.1 Resultado tras ejecutar *LOG_SET_NAME* y *LOAD_FM*

En la Figura II.2 se muestra la información relativa al árbol cargado, con el número de características que integran el árbol, nodos padres y nodos sin hijos, factor de ramificación, número de nodos desglosados por tipo de relación (AND/OR/XOR), número de supertipos, supertipos distintos presentes en el modelo, y número de nodos que están relacionados con algún supertipo, así como el número de restricciones presentes en el árbol.

```
STATISTICS
-----------
Features: 100
Breeding features: 19
Childless features: 81
Nodes AND: 17 (58,62%)
Nodes OR: 5 (17,24%)
Nodes XOR: 7 (24,14%)
Nodes Mandatory: 69 (69%)
Nodes Optional: 31 (31%)
Depth tree: 6
Average Branching Factor: 1,23
Maximum Branching Factor: 10
Maximum number of Nodes in SET relationship: 7
Supertypes: 9
Nodes with supertypes: 10
Max supertypes in a node: 2
Constraints: 5
REQUIRE Constraints: 3
EXCLUDE Constraints: 2
```

Figura II.2 Resultado del comando *SHOW_STATISTICS*

Una vez cargado el modelo, es el momento en que se lanzan las inserciones de características. En la Figura II.3 se muestra la información relativa a los intentos de inserción, mostrándose si han tenido éxito o no. Cada intento consta de dos líneas, en la primera de ellas se muestra la característica que va a ser insertada y algunos detalles sobre la misma, como el tipo de relación que posee y los supertipos que tiene asociados. En la siguiente línea, se indica si ha tenido éxito la inserción o no con los valores *SUCCESS* o *FAILURE*. En caso de tener éxito, se muestra también el nodo en el que ha sido insertada la nueva característica, y si ha sido preciso crear una característica auxiliar para dar soporte a una nueva relación entre nodos hermanos, también es mostrado. Igualmente, en caso de haber varias posibilidades, se muestra el número de ellas y cuál de ellas ha sido seleccionada.

```
FX1 - adding - |RT->MANDATORY |RS->AND |ST->ST2 ST4
FX1 - adding - 1 SUCCESS - Feature FX1 inserted under F4 - Solution:1/1
FX2 - adding - |RT->MANDATORY |RS->AND |ST->ST8
FX2 - adding - 1 SUCCESS - Feature FX2 inserted under root - Solution:1/1
FX3 - adding - |RT->MANDATORY |RS->AND |ST->ST2
FX3 - adding - 1 SUCCESS - Feature FX3 inserted under F4 - Solution:1/2
FX4 - adding - |RT->MANDATORY |RS->AND |ST->ST4
FX4 - adding - 1 SUCCESS - Feature FX4 inserted under F4 - Solution:1/2
FX5 - adding - |RT->OPTIONAL |RS->OR |ST->ST3 ST10
FX5 - adding - 1 SUCCESS - Feature FX5 inserted under TO-F69_F70_F71_F72-SET -
Solution:1/2
FX6 - adding - |RT->OPTIONAL |RS->XOR |ST->ST3
```

Figura II.3 Resultado de la ejecución de comandos *ADD_FEATURE*

Tras realizar la inserción de características, es el momento de evaluar cuántos de los intentos han tenido éxito y cuántos han fallado, para ello se invoca el comando *SUMARIZE*, y si se previamente se han medido los tiempos con *RECORD_TIME_START* y *RECORD_TIME_STOP*, se mostrará, en Figura II.4, el número de segundos utilizados en realizar las inserciones, y el número de éxitos que han tenido lugar.

```
INSERTION TIME USED: 7,77 second(s)
TOTAL: 10
SUCCESS: 10
FAILURES: 0
```

Figura II.4 Resultado de la ejecución del comando *SUMARIZE*

Tras obtener estos resultados también es necesario conocer el contenido de cada nodo del árbol, hijos que tienen, tipos de relaciones de las que forman parte o que tienen bajo ellos, así como cuál es su nodo padre del que dependen, o supertipos con los que están relacionados. Esta información se obtiene invocando el comando *SHOW_TREE_VALUES* y una muestra de su contenido se puede ver en la Figura II.5.

```
ID  |NAME                                  |RS |RT |TYPE |IDPA|PARENT
|TOTS |TOTC |MIN C|MAX C|TAND |TOR  |TXOR |LEVEL|SUPERTYPES     |S|
-------------------------------------------------------------------------
-------------------------------------------------
1   |root                                  |AND|M  |BASIC|0   |
|0    |9    |0    |0    |1    |0    |0    |0    |                |0|
2   |F1                                    |AND|O  |BASIC|1   |
|0    |0    |0    |0    |0    |0    |0    |1    |                |0|
3   |F2                                    |AND|O  |BASIC|1   |
|0    |10   |0    |0    |1    |1    |1    |1    |                |0|
4   |F3                                    |AND|M  |BASIC|1   |
|0    |1    |0    |0    |1    |0    |0    |1    |ST7             |1|
5   |F4                                    |AND|O  |BASIC|1   |
|0    |1    |0    |0    |1    |0    |0    |1    |ST2,ST4         |2|
6   |F5                                    |AND|O  |BASIC|1   |
|0    |0    |0    |0    |0    |0    |0    |1    |                |0|
```

Figura II.5 Resultado de la ejecución de comando *SHOW_TREE_VALUES*

Por último, vamos a ver una parte de la salida generada por el comando *SHOW_FM*, que muestra una salida anidada en modo de texto del contenido del árbol del modelo de características, Figura II.6. Con este formato es posible conocer los hijos de cada nodo de forma visual, así como el nivel en el que se encuentra cada nodo. Cada nodo se describe en una línea, incluyendo en la misma el tipo de relación que contiene y los supertipos que tuviera relacionados.

```
|--> root TYPE=BASIC PR=M SR=AND ST:[] Level: 0
|----> F1 TYPE=BASIC PR=O SR=AND ST:[] Level: 1
|----> F2 TYPE=BASIC PR=O SR=AND ST:[] Level: 1
|------> F28 TYPE=BASIC PR=M SR=AND ST:[ST1] Level: 2
|--------> TO-F94_F95_F96_F97_F98_F99_F100-SET TYPE=SET PR=M SR=XOR ST:[]//
Level: 3
|----------> F94 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F95 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F96 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F97 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F98 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F99 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|----------> F100 TYPE=BASIC PR=M SR=XOR ST:[] Level: 4
|------> F29 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F30 TYPE=BASIC PR=O SR=AND ST:[] Level: 2
|------> F31 TYPE=BASIC PR=M SR=AND ST:[] Level: 2
|------> TO-F32_F33_F34-SET TYPE=SET PR=M SR=OR ST:[]// Level: 2
|--------> F32 TYPE=BASIC PR=M SR=OR ST:[ST3, ST5] Level: 3
|----------> F63 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> F64 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F65 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F66 TYPE=BASIC PR=O SR=AND ST:[] Level: 4
|----------> F67 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> F68 TYPE=BASIC PR=M SR=AND ST:[] Level: 4
|----------> TO-F69 F70 F71 F72-SET TYPE=SET PR=M SR=OR ST:[]// Level: 4
```

Figura II.6 Resultado de la ejecución de comando *SHOW_FM*

# Anexo III. Resultados de Eficiencia y Escalabilidad

En este anexo se van a presentar los resultados de las pruebas de *eficiencia* y *escalabilidad* que se han explicado en el Capítulo 5 de esta tesis.

## III.1 Resultados de Eficiencia

Durante las pruebas de eficiencia se han realizado una serie de pruebas consistentes en aplicar baterías con sets de inserción consistentes en grupos de 3, 5 y 10 características. Los resultados conseguidos se han descrito en el capítulo de experimentación. Cada batería de juegos de prueba consiste en 100 juegos. Para cada juego, se han anotado las inserciones válidas y las fallidas, y hemos calculado la ratio de éxito y el tiempo empleado en milisegundos.

A continuación, se muestran las tablas correspondientes a las inserciones contenidas en los juegos de prueba de 3 características. Se ejecutan 100 juegos de prueba.

Tabla III.1 Resultados de la ejecución del conjunto de 100 juegos de prueba de 3 inserciones

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-3-01.init | 2 | 1 | 67% | 1526 |
| set-3-02.init | 3 | 0 | 100% | 2328 |
| set-3-03.init | 3 | 0 | 100% | 2423 |
| set-3-04.init | 3 | 0 | 100% | 2787 |
| set-3-05.init | 3 | 0 | 100% | 2431 |
| set-3-06.init | 3 | 0 | 100% | 2464 |
| set-3-07.init | 3 | 0 | 100% | 2426 |
| set-3-08.init | 2 | 1 | 67% | 1558 |
| set-3-09.init | 2 | 1 | 67% | 1542 |
| set-3-10.init | 3 | 0 | 100% | 2307 |
| set-3-11.init | 2 | 1 | 67% | 1667 |
| set-3-12.init | 3 | 0 | 100% | 2428 |
| set-3-13.init | 1 | 2 | 33% | 892 |
| set-3-14.init | 1 | 2 | 33% | 770 |
| set-3-15.init | 3 | 0 | 100% | 2314 |
| set-3-16.init | 3 | 0 | 100% | 2378 |
| set-3-17.init | 3 | 0 | 100% | 2460 |
| set-3-18.init | 2 | 1 | 67% | 1586 |
| set-3-19.init | 3 | 0 | 100% | 2269 |
| set-3-20.init | 3 | 0 | 100% | 2269 |
| set-3-21.init | 3 | 0 | 100% | 2392 |
| set-3-22.init | 3 | 0 | 100% | 2487 |
| set-3-23.init | 3 | 0 | 100% | 2517 |
| set-3-24.init | 3 | 0 | 100% | 2391 |
| set-3-25.init | 2 | 1 | 67% | 1603 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-3-26.init | 2 | 1 | 67% | 1569 |
| set-3-27.init | 3 | 0 | 100% | 2285 |
| set-3-28.init | 2 | 1 | 67% | 1609 |
| set-3-29.init | 3 | 0 | 100% | 2466 |
| set-3-30.init | 2 | 1 | 67% | 1586 |
| set-3-31.init | 3 | 0 | 100% | 2358 |
| set-3-32.init | 2 | 1 | 67% | 1878 |
| set-3-33.init | 2 | 1 | 67% | 1611 |
| set-3-34.init | 3 | 0 | 100% | 2400 |
| set-3-35.init | 3 | 0 | 100% | 2494 |
| set-3-36.init | 3 | 0 | 100% | 2577 |
| set-3-37.init | 3 | 0 | 100% | 2354 |
| set-3-38.init | 2 | 1 | 67% | 1605 |
| set-3-39.init | 3 | 0 | 100% | 2177 |
| set-3-40.init | 2 | 1 | 67% | 1828 |
| set-3-41.init | 2 | 1 | 67% | 1625 |
| set-3-42.init | 2 | 1 | 67% | 1530 |
| set-3-43.init | 3 | 0 | 100% | 2218 |
| set-3-44.init | 3 | 0 | 100% | 2236 |
| set-3-45.init | 3 | 0 | 100% | 2170 |
| set-3-46.init | 3 | 0 | 100% | 2205 |
| set-3-47.init | 3 | 0 | 100% | 2186 |
| set-3-48.init | 3 | 0 | 100% | 2248 |
| set-3-49.init | 3 | 0 | 100% | 2216 |
| set-3-50.init | 3 | 0 | 100% | 2255 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-3-51.init | 3 | 0 | 100% | 2188 |
| set-3-52.init | 3 | 0 | 100% | 2143 |
| set-3-53.init | 3 | 0 | 100% | 2105 |
| set-3-54.init | 2 | 1 | 67% | 1477 |
| set-3-55.init | 3 | 0 | 100% | 2134 |
| set-3-56.init | 3 | 0 | 100% | 2239 |
| set-3-57.init | 3 | 0 | 100% | 2257 |
| set-3-58.init | 3 | 0 | 100% | 2187 |
| set-3-59.init | 3 | 0 | 100% | 2242 |
| set-3-60.init | 3 | 0 | 100% | 2250 |
| set-3-61.init | 3 | 0 | 100% | 2081 |
| set-3-62.init | 2 | 1 | 67% | 1415 |
| set-3-63.init | 3 | 0 | 100% | 2177 |
| set-3-64.init | 3 | 0 | 100% | 2150 |
| set-3-65.init | 3 | 0 | 100% | 2158 |
| set-3-66.init | 3 | 0 | 100% | 2059 |
| set-3-67.init | 3 | 0 | 100% | 2115 |
| set-3-68.init | 3 | 0 | 100% | 2131 |
| set-3-69.init | 3 | 0 | 100% | 2100 |
| set-3-70.init | 3 | 0 | 100% | 2136 |
| set-3-71.init | 3 | 0 | 100% | 2123 |
| set-3-72.init | 2 | 1 | 67% | 1435 |
| set-3-73.init | 3 | 0 | 100% | 2126 |
| set-3-74.init | 3 | 0 | 100% | 2178 |
| set-3-75.init | 2 | 1 | 67% | 1461 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-3-76.init | 3 | 0 | 100% | 2101 |
| set-3-77.init | 3 | 0 | 100% | 2135 |
| set-3-78.init | 3 | 0 | 100% | 2124 |
| set-3-79.init | 3 | 0 | 100% | 2227 |
| set-3-80.init | 3 | 0 | 100% | 2099 |
| set-3-81.init | 3 | 0 | 100% | 2174 |
| set-3-82.init | 2 | 1 | 67% | 1446 |
| set-3-83.init | 2 | 1 | 67% | 1412 |
| set-3-84.init | 3 | 0 | 100% | 2148 |
| set-3-85.init | 2 | 1 | 67% | 1422 |
| set-3-86.init | 3 | 0 | 100% | 2149 |
| set-3-87.init | 2 | 1 | 67% | 1404 |
| set-3-88.init | 3 | 0 | 100% | 2169 |
| set-3-89.init | 3 | 0 | 100% | 2069 |
| set-3-90.init | 2 | 1 | 67% | 1393 |
| set-3-91.init | 3 | 0 | 100% | 2079 |
| set-3-92.init | 3 | 0 | 100% | 2106 |
| set-3-93.init | 3 | 0 | 100% | 2091 |
| set-3-94.init | 3 | 0 | 100% | 2097 |
| set-3-95.init | 3 | 0 | 100% | 2146 |
| set-3-96.init | 2 | 1 | 67% | 1446 |
| set-3-97.init | 3 | 0 | 100% | 2130 |
| set-3-98.init | 2 | 1 | 67% | 1407 |
| set-3-99.init | 3 | 0 | 100% | 2230 |
| set-3-100.in | 3 | 0 | 100% | 2098 |

A continuación se muestran las tablas correspondientes a las inserciones contenidas en los juegos de prueba de 5 características. Se ejecutan 100 juegos de prueba.

Tabla III.2 Resultados de la ejecución del conjunto de 100 juegos de prueba de 5 inserciones

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-5-01.init | 4 | 1 | 80% | 2862 |
| set-5-02.init | 5 | 0 | 100% | 3521 |
| set-5-03.init | 4 | 1 | 80% | 2742 |
| set-5-04.init | 5 | 0 | 100% | 3512 |
| set-5-05.init | 5 | 0 | 100% | 3525 |
| set-5-06.init | 4 | 1 | 80% | 2776 |
| set-5-07.init | 5 | 0 | 100% | 3466 |
| set-5-08.init | 3 | 2 | 60% | 2151 |
| set-5-09.init | 5 | 0 | 100% | 3564 |
| set-5-10.init | 5 | 0 | 100% | 3525 |
| set-5-11.init | 5 | 0 | 100% | 3466 |
| set-5-12.init | 5 | 0 | 100% | 3543 |
| set-5-13.init | 5 | 0 | 100% | 3454 |
| set-5-14.init | 5 | 0 | 100% | 3478 |
| set-5-15.init | 5 | 0 | 100% | 3550 |
| set-5-16.init | 5 | 0 | 100% | 3537 |
| set-5-17.init | 5 | 0 | 100% | 3498 |
| set-5-18.init | 5 | 0 | 100% | 3402 |
| set-5-19.init | 5 | 0 | 100% | 3544 |
| set-5-20.init | 5 | 0 | 100% | 3505 |
| set-5-21.init | 5 | 0 | 100% | 3472 |
| set-5-22.init | 5 | 0 | 100% | 3455 |
| set-5-23.init | 4 | 1 | 80% | 2705 |
| set-5-24.init | 2 | 3 | 40% | 1363 |
| set-5-25.init | 4 | 1 | 80% | 2738 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-5-26.init | 5 | 0 | 100% | 3499 |
| set-5-27.init | 4 | 1 | 80% | 2746 |
| set-5-28.init | 5 | 0 | 100% | 3394 |
| set-5-29.init | 5 | 0 | 100% | 3419 |
| set-5-30.init | 5 | 0 | 100% | 3354 |
| set-5-31.init | 3 | 2 | 60% | 2012 |
| set-5-32.init | 4 | 1 | 80% | 2752 |
| set-5-33.init | 5 | 0 | 100% | 3446 |
| set-5-34.init | 5 | 0 | 100% | 3428 |
| set-5-35.init | 5 | 0 | 100% | 3505 |
| set-5-36.init | 5 | 0 | 100% | 3514 |
| set-5-37.init | 5 | 0 | 100% | 3464 |
| set-5-38.init | 5 | 0 | 100% | 3468 |
| set-5-39.init | 5 | 0 | 100% | 3423 |
| set-5-40.init | 5 | 0 | 100% | 3422 |
| set-5-41.init | 5 | 0 | 100% | 3449 |
| set-5-42.init | 5 | 0 | 100% | 3503 |
| set-5-43.init | 3 | 2 | 60% | 2081 |
| set-5-44.init | 4 | 1 | 80% | 3238 |
| set-5-45.init | 4 | 1 | 80% | 2759 |
| set-5-46.init | 4 | 1 | 80% | 2778 |
| set-5-47.init | 4 | 1 | 80% | 2828 |
| set-5-48.init | 5 | 0 | 100% | 3621 |
| set-5-49.init | 5 | 0 | 100% | 3408 |
| set-5-50.init | 4 | 1 | 80% | 2849 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-5-51.init | 4 | 1 | 80% | 2870 |
| set-5-52.init | 5 | 0 | 100% | 3499 |
| set-5-53.init | 4 | 1 | 80% | 2748 |
| set-5-54.init | 4 | 1 | 80% | 2746 |
| set-5-55.init | 4 | 1 | 80% | 2767 |
| set-5-56.init | 5 | 0 | 100% | 3432 |
| set-5-57.init | 5 | 0 | 100% | 3403 |
| set-5-58.init | 5 | 0 | 100% | 3429 |
| set-5-59.init | 5 | 0 | 100% | 3451 |
| set-5-60.init | 4 | 1 | 80% | 2790 |
| set-5-61.init | 3 | 2 | 60% | 2042 |
| set-5-62.init | 4 | 1 | 80% | 2754 |
| set-5-63.init | 5 | 0 | 100% | 3522 |
| set-5-64.init | 4 | 1 | 80% | 2705 |
| set-5-65.init | 5 | 0 | 100% | 3476 |
| set-5-66.init | 2 | 3 | 40% | 1407 |
| set-5-67.init | 5 | 0 | 100% | 3424 |
| set-5-68.init | 5 | 0 | 100% | 3488 |
| set-5-69.init | 5 | 0 | 100% | 3567 |
| set-5-70.init | 5 | 0 | 100% | 3442 |
| set-5-71.init | 5 | 0 | 100% | 3443 |
| set-5-72.init | 4 | 1 | 80% | 2702 |
| set-5-73.init | 4 | 1 | 80% | 2740 |
| set-5-74.init | 5 | 0 | 100% | 3427 |
| set-5-75.init | 2 | 3 | 40% | 1381 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-5-76.init | 4 | 1 | 80% | 2667 |
| set-5-77.init | 5 | 0 | 100% | 3427 |
| set-5-78.init | 5 | 0 | 100% | 3412 |
| set-5-79.init | 4 | 1 | 80% | 2818 |
| set-5-80.init | 4 | 1 | 80% | 2849 |
| set-5-81.init | 5 | 0 | 100% | 3412 |
| set-5-82.init | 5 | 0 | 100% | 3545 |
| set-5-83.init | 5 | 0 | 100% | 3526 |
| set-5-84.init | 4 | 1 | 80% | 2762 |
| set-5-85.init | 5 | 0 | 100% | 3607 |
| set-5-86.init | 5 | 0 | 100% | 3631 |
| set-5-87.init | 5 | 0 | 100% | 3581 |
| set-5-88.init | 5 | 0 | 100% | 3554 |
| set-5-89.init | 5 | 0 | 100% | 3467 |
| set-5-90.init | 4 | 1 | 80% | 2845 |
| set-5-91.init | 4 | 1 | 80% | 2776 |
| set-5-92.init | 5 | 0 | 100% | 3485 |
| set-5-93.init | 5 | 0 | 100% | 3512 |
| set-5-94.init | 3 | 2 | 60% | 2205 |
| set-5-95.init | 4 | 1 | 80% | 2866 |
| set-5-96.init | 4 | 1 | 80% | 2836 |
| set-5-97.init | 5 | 0 | 100% | 3612 |
| set-5-98.init | 5 | 0 | 100% | 3608 |
| set-5-99.init | 3 | 2 | 60% | 2162 |
| set-5-100.in | 5 | 0 | 100% | 3526 |

A continuación se muestran las tablas correspondientes a las inserciones contenidas en los juegos de prueba de 10 características. Se ejecutan 100 juegos de prueba.

Tabla III.3 Resultados de la ejecución del conjunto de 100 juegos de prueba de 10 inserciones

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-10-01.in | 10 | 0 | 100% | 7192 |
| set-10-02.in | 9 | 1 | 90% | 6497 |
| set-10-03.in | 10 | 0 | 100% | 7233 |
| set-10-04.in | 9 | 1 | 90% | 6551 |
| set-10-05.in | 10 | 0 | 100% | 7084 |
| set-10-06.in | 7 | 3 | 70% | 5019 |
| set-10-07.in | 10 | 0 | 100% | 7136 |
| set-10-08.in | 9 | 1 | 90% | 6424 |
| set-10-09.in | 6 | 4 | 60% | 4314 |
| set-10-10.in | 10 | 0 | 100% | 7086 |
| set-10-11.in | 9 | 1 | 90% | 6492 |
| set-10-12.in | 8 | 2 | 80% | 5757 |
| set-10-13.in | 10 | 0 | 100% | 7127 |
| set-10-14.in | 10 | 0 | 100% | 7306 |
| set-10-15.in | 9 | 1 | 90% | 6442 |
| set-10-16.in | 8 | 2 | 80% | 5713 |
| set-10-17.in | 10 | 0 | 100% | 7173 |
| set-10-18.in | 9 | 1 | 90% | 6300 |
| set-10-19.in | 10 | 0 | 100% | 7117 |
| set-10-20.in | 10 | 0 | 100% | 7328 |
| set-10-21.in | 10 | 0 | 100% | 7173 |
| set-10-22.in | 9 | 1 | 90% | 6681 |
| set-10-23.in | 10 | 0 | 100% | 7157 |
| set-10-24.in | 10 | 0 | 100% | 7205 |
| set-10-25.in | 8 | 2 | 80% | 5760 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-10-26.in | 10 | 0 | 100% | 7118 |
| set-10-27.in | 9 | 1 | 90% | 6325 |
| set-10-28.in | 10 | 0 | 100% | 7515 |
| set-10-29.in | 10 | 0 | 100% | 8639 |
| set-10-30.in | 8 | 2 | 80% | 6781 |
| set-10-31.in | 9 | 1 | 90% | 7426 |
| set-10-32.in | 8 | 2 | 80% | 6630 |
| set-10-33.in | 10 | 0 | 100% | 8345 |
| set-10-34.in | 10 | 0 | 100% | 8320 |
| set-10-35.in | 8 | 2 | 80% | 6719 |
| set-10-36.in | 10 | 0 | 100% | 8254 |
| set-10-37.in | 10 | 0 | 100% | 8112 |
| set-10-38.in | 8 | 2 | 80% | 6639 |
| set-10-39.in | 9 | 1 | 90% | 7509 |
| set-10-40.in | 8 | 2 | 80% | 6818 |
| set-10-41.in | 10 | 0 | 100% | 8393 |
| set-10-42.in | 10 | 0 | 100% | 8339 |
| set-10-43.in | 10 | 0 | 100% | 8161 |
| set-10-44.in | 10 | 0 | 100% | 8052 |
| set-10-45.in | 10 | 0 | 100% | 8922 |
| set-10-46.in | 10 | 0 | 100% | 8351 |
| set-10-47.in | 9 | 1 | 90% | 7467 |
| set-10-48.in | 10 | 0 | 100% | 7907 |
| set-10-49.in | 10 | 0 | 100% | 7646 |
| set-10-50.in | 10 | 0 | 100% | 7739 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-10-51.in | 9 | 1 | 90% | 7016 |
| set-10-52.in | 9 | 1 | 90% | 6876 |
| set-10-53.in | 9 | 1 | 90% | 7042 |
| set-10-54.in | 9 | 1 | 90% | 7406 |
| set-10-55.in | 9 | 1 | 90% | 7153 |
| set-10-56.in | 8 | 2 | 80% | 6318 |
| set-10-57.in | 10 | 0 | 100% | 7689 |
| set-10-58.in | 10 | 0 | 100% | 7828 |
| set-10-59.in | 10 | 0 | 100% | 7705 |
| set-10-60.in | 9 | 1 | 90% | 7157 |
| set-10-61.in | 9 | 1 | 90% | 6891 |
| set-10-62.in | 10 | 0 | 100% | 8078 |
| set-10-63.in | 10 | 0 | 100% | 7669 |
| set-10-64.in | 8 | 2 | 80% | 6092 |
| set-10-65.in | 9 | 1 | 90% | 6956 |
| set-10-66.in | 10 | 0 | 100% | 7955 |
| set-10-67.in | 10 | 0 | 100% | 7869 |
| set-10-68.in | 10 | 0 | 100% | 8025 |
| set-10-69.in | 8 | 2 | 80% | 6108 |
| set-10-70.in | 9 | 1 | 90% | 6965 |
| set-10-71.in | 9 | 1 | 90% | 6870 |
| set-10-72.in | 10 | 0 | 100% | 7965 |
| set-10-73.in | 9 | 1 | 90% | 7095 |
| set-10-74.in | 9 | 1 | 90% | 7450 |
| set-10-75.in | 10 | 0 | 100% | 8779 |

| SET | VALIDOS | NO VÁLIDOS | ÉXITO(%) | TIEMPO (ms) |
|---|---|---|---|---|
| set-10-76.in | 10 | 0 | 100% | 9659 |
| set-10-77.in | 8 | 2 | 80% | 7071 |
| set-10-78.in | 10 | 0 | 100% | 8800 |
| set-10-79.in | 9 | 1 | 90% | 8234 |
| set-10-80.in | 10 | 0 | 100% | 8901 |
| set-10-81.in | 9 | 1 | 90% | 7705 |
| set-10-82.in | 8 | 2 | 80% | 6977 |
| set-10-83.in | 8 | 2 | 80% | 7009 |
| set-10-84.in | 10 | 0 | 100% | 8723 |
| set-10-85.in | 10 | 0 | 100% | 8569 |
| set-10-86.in | 9 | 1 | 90% | 7876 |
| set-10-87.in | 9 | 1 | 90% | 7662 |
| set-10-88.in | 10 | 0 | 100% | 8656 |
| set-10-89.in | 9 | 1 | 90% | 7665 |
| set-10-90.in | 10 | 0 | 100% | 8808 |
| set-10-91.in | 10 | 0 | 100% | 8748 |
| set-10-92.in | 8 | 2 | 80% | 6892 |
| set-10-93.in | 9 | 1 | 90% | 7797 |
| set-10-94.in | 8 | 2 | 80% | 7072 |
| set-10-95.in | 7 | 3 | 70% | 6086 |
| set-10-96.in | 10 | 0 | 100% | 8700 |
| set-10-97.in | 9 | 1 | 90% | 8034 |
| set-10-98.in | 9 | 1 | 90% | 7784 |
| set-10-99.in | 9 | 1 | 90% | 7743 |
| set-10-100.i | 10 | 0 | 100% | 9199 |

# III.2 Resultados de Escalabilidad

Durante las pruebas de escalabilidad se han lanzado dos juegos de pruebas de inserción 2 características cada uno sobre cada uno de los modelos 1000, 2000, 3000, 4000 y 5000 características. Tras ejecutar cada juego de pruebas de inserción se han anotado el número de casos válidos, el número de no válidos, y el tiempo invertido en ejecutar el caso de pruebas (en segundos).

Tabla III.4 Resultados de la ejecución de los juegos de prueba en árboles de 1000 características

| SET | CARAC. A AÑADIR | VALIDOS | NO VÁLIDOS | TIEMPO (SEGUNDOS) |
|---|---|---|---|---|
| set-1000-10-1 | 10 | 10 | 0 | 7,90 |
| set-1000-10-2 | 10 | 10 | 0 | 8,06 |

Tabla III.5 Resultados de la ejecución de los juegos de prueba en árboles de 2000 características

| SET | CARAC. A AÑADIR | VALIDOS | NO VÁLIDOS | TIEMPO (SEGUNDOS) |
|---|---|---|---|---|
| set-2000-10-1 | 10 | 10 | 0 | 10,20 |
| set-2000-10-2 | 10 | 10 | 0 | 10,56 |

Tabla III.6 Resultados de la ejecución de los juegos de prueba en árboles de 3000 características

| SET | CARAC. A AÑADIR | VALIDOS | NO VÁLIDOS | TIEMPO (SEGUNDOS) |
|---|---|---|---|---|
| set-3000-10-1 | 10 | 10 | 0 | 12,21 |
| set-3000-10-2 | 10 | 10 | 0 | 13,01 |

Tabla III.7 Resultados de la ejecución de los juegos de prueba en árboles de 4000 características

| SET | CARAC. A AÑADIR | VALIDOS | NO VÁLIDOS | TIEMPO (SEGUNDOS) |
|---|---|---|---|---|
| set-4000-10-1 | 10 | 10 | 0 | 13,35 |
| set-4000-10-2 | 10 | 10 | 0 | 14,29 |

Tabla III.8 Resultados de la ejecución de los juegos de prueba en árboles de 5000 características

| SET | CARAC. A AÑADIR | VALIDOS | NO VÁLIDOS | TIEMPO (SEGUNDOS) |
|---|---|---|---|---|
| set-5000-10-1 | 10 | 10 | 0 | 15,56 |
| set-5000-10-2 | 10 | 10 | 0 | 16,58 |