Aplicabilidad de la Caracterización de Benchmarks a Modelos de Variabilidad

Daniel-Jesus Munoz y Lidia Fuentes

Universidad de Málaga, Andalucía Tech, España
{danimg,lff}@lcc.uma.es,
http://caosd.lcc.uma.es

Resumen Los benchmarks utilizados para comparar el rendimiento de diferentes sistemas presentan una alta variabilidad que puede ser representada en modelos de variabilidad como los Feature Models. En este artículo presentamos los problemas de escalabilidad y complejidad de selección por objetivos de los Feature Models, y una posible solución, mostrando las ventajas de la caracterización de benchmarks (dada por sus cargas de trabajo). Para esta mejora formalizamos un modelo de caracterización de paquetes de cargas de trabajo para Feature Models, basándonos en ocho atributos abstractos (operaciones matemáticas, memoria, ...). Este modelo y sus ventajas son evaluados en el eco-asistente HADAS, junto a un benchmark PHP, y al benchmark de sistemas empotrados BEEBS, obteniendo una capacidad de selección más intuitiva, y un decremento en el tiempo de obtención de configuraciones válidas y sus métricas en HADAS, con respecto a la representación estándar.

Keywords: Variabilidad, benchmarking, Caracterización, Software, Eficiencia, Escalabilidad, Features, Atributos

1. Introducción

Los programadores de software procuran desarrollar aplicaciones eficientes, y para ello tratan de establecer comparativas de diferentes soluciones para cargas de trabajo variables (del inglés *workloads*). Una carga de trabajo consiste en una serie de operaciones ejecutándose en un sistema (ej.: servidor, ordenador, smartphone, empotrado...), que se completarán de forma satisfactoria en un determinado tiempo [6]. Ese tiempo nos indicará lo eficiente que es el sistema para ese conjunto de operaciones (para esa carga de trabajo). Si un sistema A tarda menos tiempo que otro sistema B en completar esa misma carga de trabajo de manera exitosa, concluimos que el sistema A es más eficiente en tiempo. Esto es aplicable a la ecoeficiencia (eficiencia energética) del software, donde si una operación X es programada de manera diferente (C y D), y ejecutada en un sistema Y, resultando en un consumo energético menor de la manera C, concluimos que la manera C es más energéticamente eficiente que la manera D para un sistema Y. Si esto, además, se cumple para varios sistemas, se concluiría que C siempre es más energéticamente eficiente que D. Por tanto, un desarrollador que busque el mínimo consumo energético optaría siempre por C.

En el mundo real, una aplicación suele contener varias operaciones en conjunto. Para poder comparar la eficiencia entre aplicaciones reales se crearon los *benchmarks*. Un benchmark es un conjunto de cargas de trabajo (operaciones) buscando evaluar el rendimiento relativo de un sistema completo (hardware y/o software) [8]. Un benchmark se desarrolla de la manera más independiente, para poder ser lanzado en el mayor número de sistemas posibles, enriqueciendo las comparativas con una mayor cantidad de medidas (de tiempo de ejecución, de consumo energético...). benchmarks populares como SPEC [4] son muy genéricos, pero existen otros más específicos para dominios más concretos como, por ejemplo, la automoción (e.g., AutoBench [5]). Para representar tanto la variabilidad de los distintos sistemas y operaciones, como la de los distintos tipos de benchmarks, proponemos utilizar los *Modelos de Variabilidad*.

Los Modelos de Variabilidad son ampliamente utilizados desde su primera definición formal denominada *Feature Model* (FM) en 1990 [9]. Un FM se usa generalmente para representar la variabilidad y los elementos comunes en líneas de productos y en software reutilizable. Un FM define todas las posibles configuraciones válidas de las distintas *Features* (características). Features en el contexto de benchmarking abarcan desde distintos algoritmos (ej: Quicksort, Encriptación DSA, ...), hasta parámetros como la longitud de la clave (Key Size) o el tamaño del Array. Una ventaja importante de utilizar FMs es que podemos mejorar el grado de reutilización de características comunes, y la eliminación de las características no usadas, reduciendo la complejidad del diseño de sistemas. Aun así, uno de los principales problemas que comparten FMs y benchmarks muy completos es su gran tamaño y complejidad, implicando cientos de miles de características y decenas de miles de parámetros (variables). Los problemas de satisfacibilidad booleana (SATs) binarios [12] son una solución enfocada a los FMs que trata de reducir la complejidad mediante un sistema de Hash binario (1/0 implica característica seleccionada/deseleccionada) y una función lógica. Por otra parte, la caracterización de benchmarks [3] también trata de reducir la complejidad, facilitando el entendimiento del conjunto de cargas de trabajo y de los resultados, sin requerir conocimientos muy específicos y detallados de mencionadas cargas de trabajo.

En el proceso de desarrollo y uso del *Eco-Asistente HADAS* [14] hemos trabajado tanto con FMs (modelos Clafer [1]), como con benchmarking. HADAS [1] es un eco-asistente con un repositorio de mediciones colaborativo, que pretende aconsejar a investigadores, diseñadores y programadores sobre el rendimiento y la eficiencia energética de las distintas posibilidades para una propiedad dada (ej: diseño de un servidor web PHP [15] o primitivas criptográficas). Las gráficas, análisis estadísticos y consejos que ofrece el servicio HADAS pretenden sustentar análisis ricos sin necesidad de que el usuario conozca los detalles de las distintas alternativas. El perfil más común de los usuarios de HADAS es, partiendo de un caso de estudio concreto, es decir, de unas propiedades funcionales, obtener con el menor conocimiento la mejor configuración posible (consumo energético o tiempo de ejecución), o la más equilibrada (*trade-off*). Por tanto, a nivel

[1] `https://hadas.caosd.lcc.uma.es`

de definición de Features del árbol se nos presenta el problema de hasta que detalle se ha de llegar, es decir, que nivel de configurabilidad es permitido. El mismo caso se da en los consumidores de benchmarks; pretenden conocer como de eficiente será su configuración en base a los resultados obtenidos por otras configuraciones, sin tener que programar y ejecutar las aplicaciones de manera real. En cuanto este perfil de usuario encuentra cierta complejidad en la definición del caso de uso (ya sea mediante selección de Features en HADAS, o de un benchmark apropiado según las operaciones que realice), pierde el interés. En este artículo abordamos las siguientes *Research Questions*:

- **RQ1**: ¿La caracterización de benchmarks puede reducir la complejidad de un modelo de variabilidad? Proponemos aplicar la técnica de caracterización de benchmarks para reducir la complejidad (estructura y/o tiempo requerido), diseño y entendimiento (conocido como Design Space Search) de los FMs.
- **RQ2**: ¿Existe un nivel de detalle en la caracterización de benchmarks acorde a los distintos tipos de consumidores (usuarios, diseñadores y programadores)? Proponemos definir una serie de características comunes con el suficiente potencial para describir de una manera sencilla y exacta la variabilidad presente en benchmarking.

Este artículo presenta la siguiente estructura. En la sección 2 realizamos una revisión del trabajo relacionado en tres ramas, modelos de variabilidad, su complejidad, y la caracterización de benchmarks. En la sección 3 detallamos el benchmarking y sus opciones de caracterización. En la sección 4 profundizamos en los FMs, proponiendo un modelo de caracterización. En la sección 5 evaluamos nuestra propuesta aplicándola a dos casos y benchmarks, comenzando por un trabajo de servidores web PHP, y siguiendo con el benchmark BEEBS [17] en dispositivos con microprocesador de arquitectura ARM. En la sección 6 presentamos las conclusiones principales y su continuación como trabajo futuro.

2. Trabajo Relacionado

En este trabajo nos hemos basado en tres áreas de investigación. En primer lugar, analizamos los trabajos sobre la complejidad y claridad de los modelos de variabilidad. En segundo lugar, discutimos los trabajos sobre la complejidad computacional y analítica de los FMs. Por último, listamos los trabajos enfocados en la caracterización de benchmarks.

En Metzger et al. [13] encontramos un trabajo sobre logros y desafíos de los modelos de variabilidad y de las líneas de productos. En él, se especifica como desafío pendiente de resolver el «*Establecer un Equilibrio entre Expresividad y Analizabilidad*», es decir, ¿cómo facilitar la selección para objetivos concretos?. Concluyen que el análisis del equilibrio debe basarse en ejemplos de industria reales, validados mediante experimentación. Además, deben seguirse enfoques inteligentes que no afecten a la interrelación entre el modelo de variabilidad y sus artefactos software. En Melo et al. [11] encontramos un trabajo sobre la

variabilidad desde el punto de vista de los programadores, donde llegan a usar un sensor de visión, registrando cómo reaccionan los ojos de los programadores. Resaltan que, aunque una mayor variabilidad aporta beneficios, también tiene un coste, complicando, a veces innecesariamente, la lógica de un programa y/o código. Para solventar el coste en complejidad, se necesita de una curva de aprendizaje notable, la cual disuade tanto a diseñadores como a programadores. En nuestro propio trabajo con HADAS [14] también hemos lidiado con un nivel óptimo de detalle que satisfaga a los distintos usuarios. Básicamente, son los expertos en medición de energía de un caso concreto los que deciden qué nivel de detalle es el apropiado. Hemos añadido incluso una etiqueta meta-datos donde será incluida toda la información que, siendo importante, no lo es lo suficiente para ser representada como una Feature (ej: el equipo de investigación que realiza las mediciones, el medidor usado, etc.).

En Liang et al. [10] se analiza la facilidad de resolución de modelos de variabilidad grandes y reales con *SAT solvers*. Mediante experimentación, se comprueban satisfactoriamente diversas técnicas de simplificación de un modelo de variabilidad real. Las principales técnicas son heurística, *Boolean constraint propagation* y *backjumping*. Por otra parte, también aplican reducciones y simplificaciones previas al razonamiento de las configuraciones, las cuales son técnicas que reducen la complejidad y el tiempo de resolución. En Ochoa et al. [16] comienzan con la premisa «*la escalabilidad y el rendimiento comienzan a ser un problema en ambos, modelos de variabilidad y líneas de producto, grandes*». En este artículo de *Estado del Arte* se analizan 21 publicaciones, y se concluye que es necesario trabajo adicional, incluyendo un mayor número de modelos industriales reales (no solo teóricos). Como trabajo futuro se resalta: (i) Investigar modelos intermedios que satisfagan requisitos de casos de uso, pero, a la vez, incremente la satisfacción de los usuarios y consumidores de los modelos, y (2) Mejorar la representación a diferentes niveles de detalle, así como la sencillez en la selección de Features en configuraciones parciales y/o completas. En Carbonnel et al. [2] se formaliza un método para la fusión de modelos de variabilidad, reduciendo así el número de Features del modelo resultante mediante la cancelación de Features duplicadas, reduciendo la variabilidad del modelo final.

En Conte at al. [3], que data de 1991, se presenta una formalización general de la caracterización de benchmarks. Está más enfocada al ámbito de la arquitectura de microprocesadores, por tanto, se proponen parámetros como el índice de computación de suma, logica, multiplicación..., para ALUs (Unidades Aritmético-Lógicas) Enteras, para las Punto Flotante, registros *Store* y *Load*, etc. Se aplica a benchmarks comerciales como Dhrystone [20] y Whetstone [7], concluyendo que la caracterización de benchmarks reducen el coste racional/computacional de entender/explorar el espacio de búsqueda, enfocando la solución a cargas de trabajo con un nivel de abstracción mayor, recortando en detalles de simulación y de rediseño de cargas de trabajo similares. En Poovey et al.(2007) [18] se presenta un nuevo modelo de formalización basado en Conte at

al. [3], reduciéndolo a cinco unidades funcionales representadas por los siguientes porcentajes:

1. Operaciones ALU: Operaciones aritméticas (ej: suma, resta, multiplicación, ...) y operaciones lógicas (si, y, o, no).
2. Uso de Memoria Principal: Memoria donde se almacenan temporalmente tanto los datos de los programas que la unidad central de procesamiento (CPU) está procesando, como las que va a procesar en un determinado momento.
3. Uso de Memoria Caché: Memoria de acceso rápido de una CPU, que guarda temporalmente los datos procesados recientemente.
4. Ratio Ramificación: Paralelización de las secuencias en serie de un programa.
5. Corrimiento Lógico: Operación binaria de corrimiento a derecha/izquierda.

Tras un proceso experimental con diversos benchmarks generalistas de la EEMBC [2], concluyen que, mediante el uso de métricas de diseño hardware y eficiencia, es ventajoso representar en conjunto, mediante las citadas características, la extensa variabilidad de las distintas cargas de trabajo de los distintos benchmarks. Al estar representadas las cargas de trabajo de una manera más realista (menos matemático-algorítmica), es más factible y sencillo valorar configuraciones más efectivas. En un trabajo posterior de Poovey et al.(2009) [19] se experimenta con una rango más amplio de benchmarks de diferentes enfoques (p. ej. automoción, entretenimiento digital, oficina). Se demuestra, mediante la caracterización, que los benchmarks de enfoque similar presentan una caracterización quasi-idéntica. Además, concluyen que es necesario comprobar el desempeño de una configuración en distintos enfoques para obtener una visión realista y generalista de la eficiencia de mencionada configuración. En Pallister et al. [17] se detalla el benchmark para sistemas empotrados BEEBS, y, en la evaluación, se detecta que la caracterización del benchmark puede variar levemente dependiendo de las optimizaciones del compilador y de la plataforma en la que se ejecute. No obstante, esas variaciones son utilizadas para comparar las diferencias en rendimiento de las distintas configuraciones de compilación y del sistema. El modelo de caracterización es reducido: (i)ALU, (ii)Memoria, (iii)Caché, y (iv)Ramificación

3. Benchmarking

La principal motivación del benchmarking es la búsqueda y comprobación de diseños eficientes para cualquier sistema. Un enfoque de Fuerza-Bruta es demasiado costoso e inconcluso en tiempo y desarrollo. El atractivo de un benchmark es la definición de un sistema abstracto que, por una parte, es lo suficientemente genérico para incluir variantes de diseño y alternativas tecnológicas, y por otra, es lo más independiente posible del sistema donde se ejecuta, estableciendo una métrica comparativa mediante los resultados de tiempo de ejecución y consumo energético. Un benchmark, independientemente de su enfoque (e.g., automoción, oficina), pertenece a una de las siguientes categorías [3]:

[2] `https://www.eembc.org/benchmark/products.php`

- Reales: Se adapta una aplicación comercial como benchmark. Actualmente puede abarcar desde aplicaciones como el editor de imágenes Gimp [3], a videojuegos como The Witcher [4].
- Nucleares: Solo se evalúa el núcleo de una aplicación real. Dicho núcleo se cree responsable de la eficiencia de la aplicación en general.
- Parciales: Formados por trozos de aplicaciones reales. Pertenecen a nichos muy específicos y presentan dificultades de reproducción y compatibilidad al desconocer que parte se ha eliminado de la aplicación, y por qué.
- Recursivos: Programas 100 % recursivos como el problema de 'Las 9 Reinas'.
- Locales: Específicos para un sistema y configuración muy concretos. Normalmente, solo se usan para evaluar la estabilidad de un sistema concreto.
- Sintéticos: Colección de pequeños programas especialmente diseñados para benchmarking. No realizan una computación útil, no tienen ningún otro fin que estresar el sistema de una manera concreta. La idea principal es que un conjunto de micro-operaciones puede representar una aplicación real de manera aproximada. A esta categoría pertenecen los más comerciales y generalistas como Aida64 [5], PCMark [6], etc.
- Híbridos: Aplicación real que contiene micro-operaciones sintéticas. Un ejemplo es la 'Evaluación del Sistema' de los sistemas operativos Windows.

Dado que las micro-operaciones sintéticas son factibles de caracterizar, y nuestro estudio pretende caracterizar, de manera óptima, en un modelo de variabilidad, la comunalidad de las Features del benchmarking, hemos centrado nuestra investigación en benchmarks Sintéticos e Híbridos. La caracterización de benchmarks intenta crear una descripción sencilla de las diferentes cargas de trabajo. Esto implica redescribir las cargas de trabajo como un conjunto cuantificable de atributos abstractos [18]. A posteriori, podemos usar estos atributos para clasificar benchmarks parecidos. A estos atributos los denominamos *características*. Un diseñador (i.e., hardware) y/o programador (i.e., software) que combine estas características con el cometido del sistema a desarrollar, puede obtener información de las *suites* de benchmarks más acordes a su caso de uso. En resumen, si esta variabilidad cuantificada se encuentra modelada, un desarrollador solo necesita conocer el objetivo del sistema a desarrollar, desconociendo por completo cualquier detalle de cualquier benchmark. La mayoría de los estudios como [19] se centran en definir un modelo de caracterización basándose en las distintas unidades funcionalidades físicas (i.e., hardware), y en el uso en conjunto que hacen de ellas las distintas cargas de trabajo de una *suite* (i.e., benchmark). Uno de los motivos de su enfoque hardware es la pertenencia de sus autores al consorcio de sistemas empotrados EEMBC. Este modelo ya detallado en el Trabajo Relacionado (sección 2) presenta una limitación; un programador necesita tener conocimiento hardware avanzado para poder caracterizar, siguiendo este modelo, su caso de uso.

[3] `https://www.gimp.org/`
[4] `http://thewitcher.com`
[5] `https://www.aida64.com/`
[6] `https://www.3dmark.com/`

Para solventar esta limitación, en nuestro trabajo pretendemos canalizar de manera más abstracta la definición de ese modelo, además de enfocarlo hacia el software. Además, la mayoría de los programas para sistemas con microprocesadores de propósito general (i.e., Raspberry Pi, smartphones, equipos de sobremesa, ...) no tendrán en cuenta, en la fase de desarrollo, cuantas unidades funcionales de punto flotante, caché, etc., contienen los dispositivos que los ejecuten, por tanto, los atributos deben ser independientes de la plataforma en la que se ejecute. Por último, debe ser una caracterización lo suficientemente amplia como para describir cualquier tipo de benchmark, pero, a la vez, sencilla, siendo fácilmente entendida por los desarrolladores. Para ello, basándonos en simulaciones de traza, análisis estadísticos y en tiempo de ejecución, proponemos los siguientes atributos abstractos:

1. Computación **Lógica** (Binaria): AND, OR, XOR...
2. Computación **Entera**: ADD, SUB, MULT...
3. Computación en **Punto Flotante**: ADD, SUB, MULT...(con decimales).
4. Operaciones en **Arrays**: SORT, SEARCH, MERGE....
5. Operaciones en Memoria **Caché**: SAVE, LOAD, MOVE...
6. Operaciones en **Memoria Principal**: SAVE, LOAD, MOVE...
7. **Ramificación**: Multihilo, Multiproceso, Distribuida.
8. **Recursividad**.

Para representar, seleccionar, razonar y evaluar estos atributos en benchmarking usamos FMs.

4. Benchmarks y Feature Models

Un FM es representado como un árbol donde cada nodo representa una característica distinta. Un nodo 'hijo' puede ser *obligatorio* (i.e., siempre presente en una configuración final si su 'padre' está presente) u *opcional*. Un nodo 'padre' puede restringir a sus nodos 'hijo' con operadores *OR* (dado un nodo 'padre', al menos un 'hijo' debe estar presente en esa configuración) y *alternativo* (dado un nodo 'padre', *solo un 'hijo'* puede estar presente en esa configuración). Adicionalmente, pueden definirse restricciones *cross-tree* entre nodos de ramas divergentes, como por ejemplo $X \rightarrow Y$ (X implica a Y) y $X \rightarrow \neg Y$ (X excluye a Y). En el caso de HADAS, se añade un tipo adicional de restricción, las relativas a la presencia o no de mediciones en su base de datos (una configuración puede ser válida, pero no se han realizado las métricas correspondientes). Usando la funcionalidad de *Instance Generator* de Clafer [1], podemos razonar sobre configuraciones tal que ambas restricciones (estructurales y cross-tree) sean satisfechas , obteniendo todas las configuraciones válidas posibles.

Para representar el modelo de atributos abstractos propuesto, y, posteriormente evaluarlo, usaremos el FM de HADAS (Figura 1), lo cual nos permite no solo modular el modelado, si no ejecutar los análisis automáticos del eco-asistente HADAS [14]. Tiene cuatro niveles y 3 sub-niveles:

1. **Referentes de Consumo Energético**(RCE): Seguridad, Compresión, Comunicación, Distribución, Memoria (no-volátil) y Caché.
2. **Alternativas de Diseño**: Cada RCE presenta un sub-árbol con distintas opciones de diseño. Son restricciones de tipo alternativo. Un ejemplo serían los diseños *Cliente/Servidor* y *P2P* en la RCE *Distribución*.
3. **Variantes Tecnológicas**: Se especifican diferentes implementaciones de un mismo diseño. Por ejemplo, los distintos frameworks o APIs.
4. **Contexto**: Este sub-árbol contiene tres sub-niveles
   a) **Tipo de Datos**: Por ejemplo, texto plano, XML, Objeto, Entero, Varchar(50),..., cuya selección dependerá del caso de uso a estudio.
   b) **Operaciones**: Cada diseño presenta diversas operaciones. En el caso de Bases de Datos, encontramos *CREATE*, *SELECT*, *UPDATE*, etc.
   c) **Parámetros**: Todos los parámetros deben tener un valor, como, por ejemplo, el número de usuarios (valor discreto) o la carga de computación del sistema (alta, media, baja).

En *Operaciones* podríamos encontrar un benchmark, sus cargas de trabajo, los algoritmos que implementa cada carga de trabajo, o, incluso las instrucciones. Los *Parámetros* serán definidos en base a las variables configurables de las *Operaciones*. Este párrafo refleja la problemática de que nivel de detalle/variabilidad es necesario representar (RQ2). Adicionalmente, en el caso de HADAS, deberán realizarse más mediciones de consumo, o bien aumentar el número de restricciones, a medida que el nivel de detalle aumenta. Aplicando la metodología de la caracterización de benchmarks, los ocho atributos abstractos, y un índice sencillo de presencia (Nulo, Bajo, Medio, Alto) obtenemos las ventajas:

- **Formalización y consenso del nivel de detalle**, lo que nos permite representar/seleccionar una colección de cargas de trabajo, es decir, un benchmark, como una sola operación.
- **Representación de la variabilidad mediante atributos abstractos ya definidos**, en lugar de definir las cargas de trabajo, instrucciones y/o algoritmos, de manera individual, sabiendo que nos interesa evaluarlos como conjunto, representando un sistema real.
- **Modelos de variabilidad reducidos**:
  - **Menor tiempo de razonamiento de SAT solvers debido a un menor número de Features**, ya que los ocho atributos abstractos (y sus índices de uso) representan a todos los benchmarks de ese sub-árbol *Contexto*, sin importar cuantos sean.
  - **Representación más clara, concisa, y sencilla, en análisis y comparativas como las del Eco-Asistente HADAS**, ya que se selecciona y se obtienen medidas por objetivos (e.g., aplicación con uso intensivo de Arrays), en vez de por instrucción (e.g., insertar en Array por cabeza).

Aclarando conceptos, el *Dominio* de un parámetro del contexto es el tipo de dato inherente a esa variable. Con caracterización, tan solo se representan los parámetros externos al benchmarks (por ejemplo: número de usuarios, de tipo Entero). Sin caracterización, deberían definirse también los parámetros internos

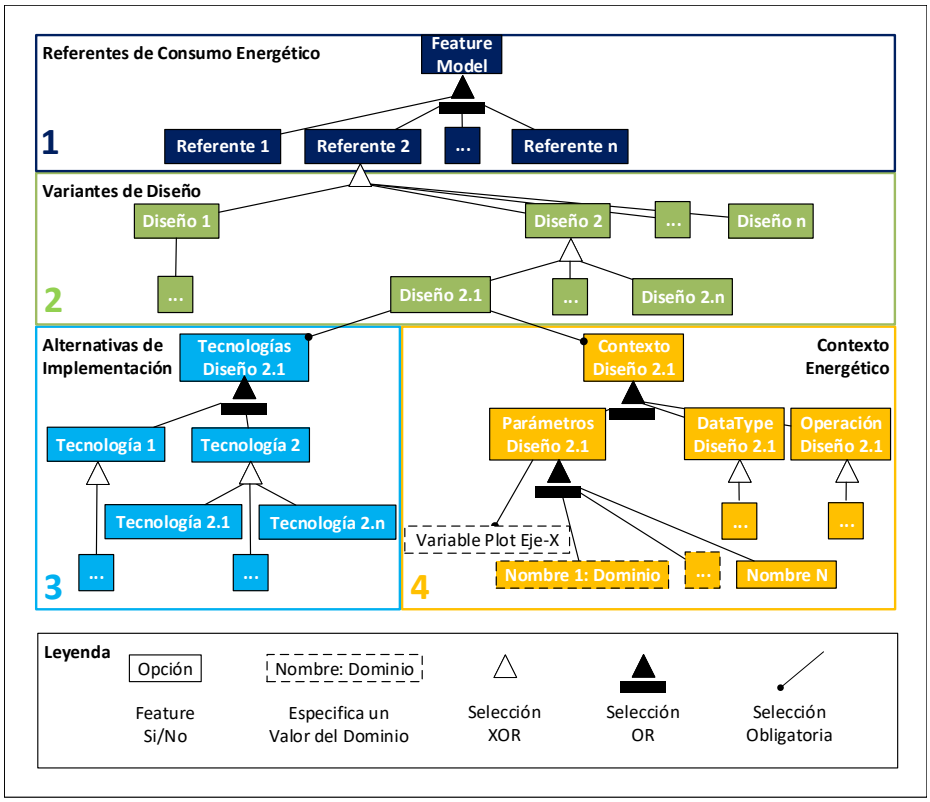

Figura 1. Extended Feature Model de HADAS

(e.g., tamaño de inicialización de un Array). Por otra parte, el DataType, si no se trabaja con Operaciones simples, representa el formato del código (i.e., ejecutable JAR, fichero web PHP, firmware, ...). En artículos anteriores como Munoz et al. [15] resolvimos esta problemática definiendo un benchmark PHP como una sola operación/nodo (igual al sistema que proponemos ahora), pero, sus parámetros relativos a la Operación se resumieron en uno: carga computacional del microprocesador (baja, media, alta). Esta opción, si bien era idónea para la claridad de aquella publicación, tan solo es capaz de representar de forma realista un rango muy bajo de benchmarks. Para solucionarlo, y evaluar nuestra propuesta, modelamos en este artículo la variabilidad de aquel caso de estudio sin y con la caracterización propuesta en este apartado, comparando ambos FMs.

5. Evaluación

En trabajos posteriores a [15], hemos realizado mediciones con un benchmark de amplio espectro (caso de uso genérico) mantenido por Rusoft Ltd [7].

[7] `https://github.com/rusoft/php-simple-benchmark-script`

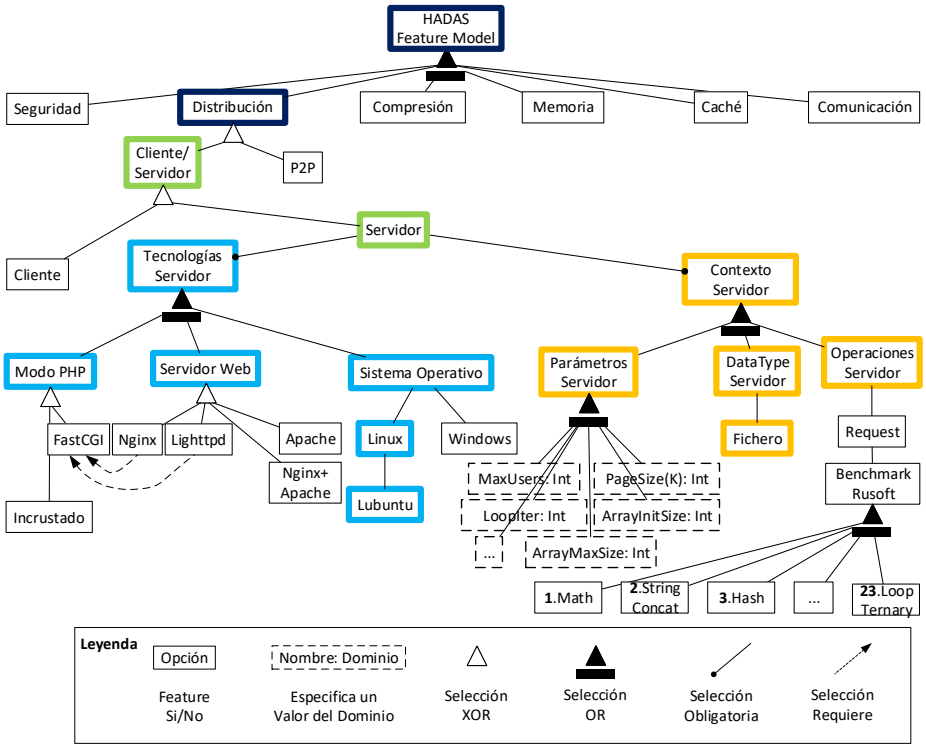

Figura 2. Feature Model de HADAS sin caracterización de benchmark PHP

Este benchmark de PHP está estructurado en cargas, recibiendo mediciones individuales de ejecución por cada carga de trabajo (segundos, operaciones por segundo, energía, uso de memoria, ...). Por tanto, en la Figura 2 podemos visualizar el Feature Model de HADAS correspondiente al caso de uso de servidores web PHP, incluyendo el benchmark Rusoft y sus 23 cargas de trabajo representadas por el nombre de la función que Rusoft asignó. Tras evaluarlo en HADAS, la representación estándar presenta las siguientes particularidades:

1. Un desarrollador, con un caso de uso, que proceda a realizar una selección y análisis, deberá hacerlo por carga de trabajo. Por ejemplo, el desarrollo de una calculadora en PHP, ¿solo seleccionaría *Math*?, ¿qué es *Math*? En un caso de una aplicación más completa, la incertidumbre sobre 'qué seleccionar' es aún mayor.
2. Si tenemos varios benchmarks, el número de nodos se incrementa según el número de operaciones en cada benchmark. Por otra parte, el incremento en número de configuraciones es exponencial. Esto también afecta a la incertidumbre sobre 'que seleccionar' (e.g., elegir nodo *Math* y similares en los sub-árboles de todos los benchmarks PHP).
3. El benchmark Rusoft es fácilmente modificable para añadir mediciones a nivel de instrucción. Esto incrementaría, de nuevo, el número de nodos y

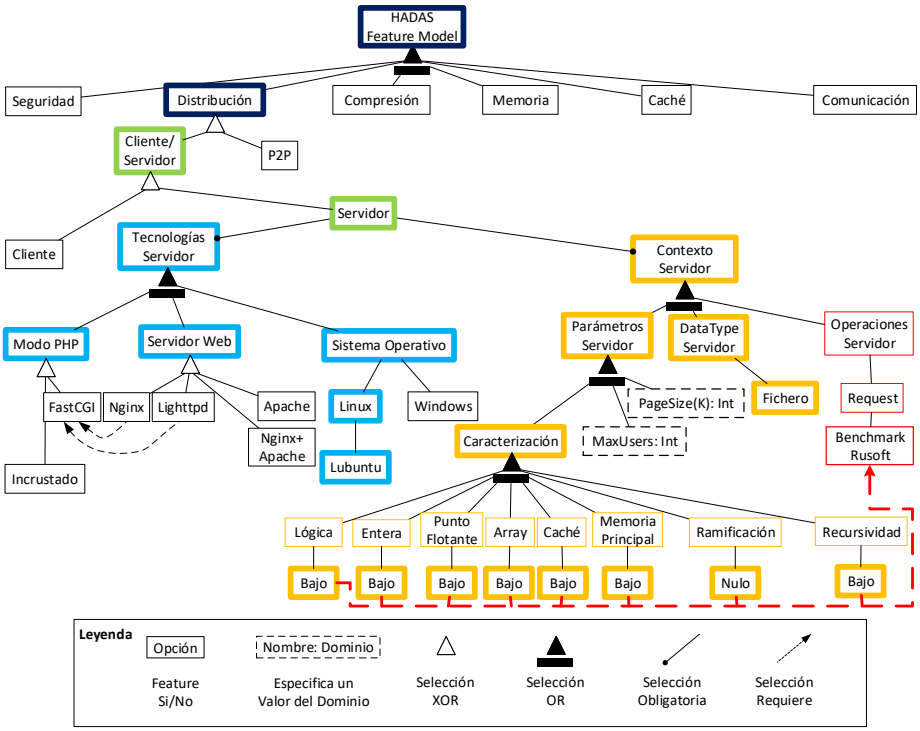

Figura 3. Feature Model de HADAS con caracterización de benchmark PHP

configuraciones, así como el número de parámetros. ¿Es necesario un nivel de detalle mayor?

4. El número de mediciones y ejecuciones a realizar es dependiente del punto anterior, por tanto, el trabajo de los investigadores también se incrementa.

Aplicando la caracterización propuesta, y ejecutando el benchmark Rusoft, obteniendo las operaciones por segundo y por Megaherzio de cada carga de trabajo ($\backsim$ 700 KOpS/Mhz), resultaría en el Feature Model y selección (objetivo: aplicación PHP genérica) de la Figura 3. Tras evaluarlo en HADAS, presenta las siguientes particularidades:

1. Un desarrollador, en este caso, realizará la selección por atributos (ocho, con hijos: nulo, bajo, medio y alto). En el caso de la calculadora PHP, la selección sería *Entera* y *Punto Flotante* con nivel de uso *Alto*, lo cual es intuitivo, e independiente del conocimiento del código del benchmark Rusoft. En resumen, la selección se basa únicamente en objetivos (caso de uso), y no en la implementación del benchmark.
2. La selección es la misma, independientemente del número de benchmarks. La complejidad computacional, por tanto, pasa a ser lineal. Al aumentar el número de benchmarks, aumentaran las restricciones, pero Clafer es más eficiente a medida que aumenta el número de restricciones [1].

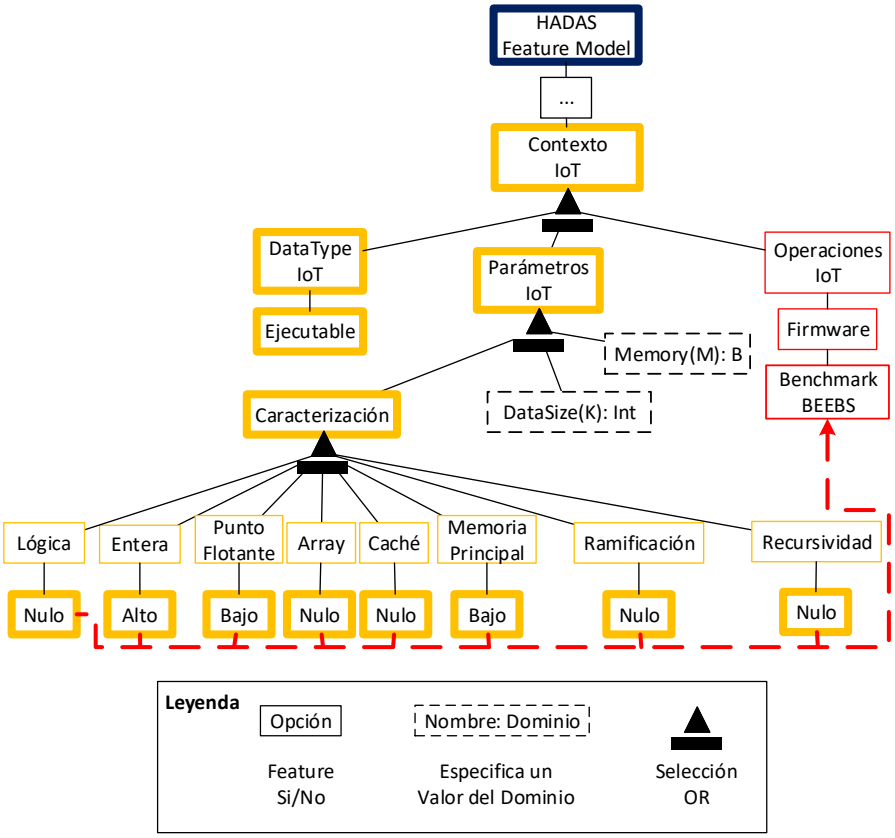

Figura 4. Feature Model de HADAS con caracterización de benchmark BEEBS

3. Un nivel de detalle mayor no es requerido cuando se evalúan aplicaciones grandes, no obstante, sería posible, y solo se incrementan las restricciones, lo cual es beneficioso para Clafer.
4. El número de mediciones y ejecuciones de los investigadores se incrementaría en la misma cantidad que el caso estándar (i.e., sin caracterización).

El modelo propuesto es también aplicable a estudios de ecoeficiencia con benchmarks que usen una caracterización diferente. De hecho, aun siendo modelos de caracterización distintos, suelen presentar similitudes, facilitando la caracterización. En Pallister et al. [17] se presenta BEEBS, un benchmark genérico para sistemas empotrados formado por 10 cargas de trabajo, específicamente diseñado para la ecoeficiencia de dispositivos IoT. En sus evaluaciones podemos visualizar como presenta 5 características y su porcentaje de uso, siendo una de ellas ambigua: (i) Entero, (ii) Punto Flotante, (iii) Memoria Principal, (iv) Ramificación, y (v) Otros. La categoría (v) es una mezcla de las categorías restantes que nosotros proponemos, y que suponemos que han sido categoriza-

das como 'Otros' por su menor importancia y presencia en las distintas cargas de trabajo. Para la obtención de los datos de este benchmark en HADAS (Figura 4, siguiendo nuestro modelo, tan solo tenemos que seleccionar un uso de *Enteros->Alto*, *Punto Flotante->Bajo* y *Memoria->Baja*, ahorrándonos incluso la lectura del artículo, y pudiendo obtener un mayor número de conclusiones que las publicadas (incluyendo conclusiones menores).

Los resultados finales tras aplicar la caracterización benchmark al Feature Model de HADAS son: (i) una reducción de tiempo de ejecución de aproximadamente un 10 % en la obtención de métricas en HADAS tras comparar las selecciones mostradas en la Figura 2 (sin caracterización) y la Figura 3 (con caracterización), (ii) una selección/clasificación enfocada en objetivos mediante ocho características básicas (i.e., objetivo obtener medidas para una aplicación web genérica, respondiendo a RQ2), también probado en el modelo de BEEBS [17], (iii) un sistema con mayor escalabilidad (e.g., de complejidad exponencial a lineal, respondiendo a RQ1) debido al menor número de nodos, y (iv) unos análisis de sostenibilidad más intuitivos (tras la selección y visualización por objetivos).

6. Conclusiones y Trabajo Futuro

Tras definir ocho atributos abstractos básicos para caracterizar benchmarks, hemos comprobado cómo es posible usarlos para realizar una selección por objetivos en un Feature Model, basándonos únicamente en el conocimiento del caso de uso a evaluar. Adicionalmente, tras la aplicación de la caracterización al Feature Model de HADAS, la complejidad, en la medida en que el número de benchmarks se incrementa, se reduce hasta ser lineal. Por tanto, concluimos que la caracterización de benchmarks presenta ventajas en su aplicación a modelos de variabilidad, entre ellas definiendo modelos más escalables e intuitivos. Estas conclusiones han sido corroboradas en HADAS, reduciendo el tiempo de respuesta en un 10 % para el objetivo de un benchmark para una aplicación generalista en PHP, y para el benchmark de sistemas empotrados BEEBS. Como trabajo futuro vislumbramos la aplicación de nuestra caracterización a un rango amplio de benchmarks, a diferentes categorías y a aplicarlo a diferentes modelos de procesos SPLE reales como HADAS, así como un estudio con una serie de diseñadores y desarrolladores, evaluando su aplicabilidad, escalabilidad y proceso de selección (restricciones) intuitivo.

Agradecimientos

Trabajo financiado por los proyectos MAGIC P12-TIC1814 y HADAS TIN2015-64841-R, y por la Universidad de Málaga.

Referencias

1. Bąk, K., Czarnecki, K., Wąsowski, A.: Feature and meta-models in clafer: mixed, specialized, and coupled. In: International Conference on Software Language Engineering. pp. 102–122. Springer (2010)

2. Carbonnel, J., Huchard, M., Miralles, A., Nebut, C.: Feature model composition assisted by formal concept analysis. In: ENASE: Evaluation of Novel Approaches to Software Engineering. pp. 27–37. SciTePress (2017)
3. Conte, T.M., Hwu, W.M.: Benchmark characterization. Computer 24(1), 48–56 (1991)
4. Dixit, K.M.: The spec benchmarks. Parallel computing 17(10-11), 1195–1209 (1991)
5. Ginsberg, M.: Autobench: A 21st century vision for an automotive computing benchmark suite. High Performance Computing in Automotive Design, Engineering, and Manufacturing, Cray Research, Inc., Eagan, MN pp. 67–76 (1997)
6. Gopher, D., Donchin, E.: Workload: An examination of the concept. (1986)
7. Harbaugh, S., Forakis, J.A.: Timing studies using a synthetic whetstone benchmark. ACM SIGAda Ada Letters 4(2), 23–34 (1984)
8. Jain, R.: The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. John Wiley & Sons (1990)
9. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990)
10. Liang, J.H., Ganesh, V., Czarnecki, K., Raman, V.: Sat-based analysis of large real-world feature models is easy. In: Proceedings of the 19th International Conference on Software Product Line. pp. 91–100. ACM (2015)
11. Melo, J., Narcizo, F.B., Hansen, D.W., Brabrand, C., Wasowski, A.: Variability through the eyes of the programmer. In: Program Comprehension (ICPC), 2017 IEEE/ACM 25th International Conference on. pp. 34–44. IEEE (2017)
12. Mendonca, M., Wąsowski, A., Czarnecki, K.: Sat-based analysis of feature models is easy. In: Proceedings of the 13th International Software Product Line Conference. pp. 231–240. Carnegie Mellon University (2009)
13. Metzger, A., Pohl, K.: Software product line engineering and variability management: achievements and challenges. In: Proceedings of the on Future of Software Engineering. pp. 70–84. ACM (2014)
14. Munoz, D.J., Pinto, M., Fuentes, L.: Green software development and research with the hadas toolkit. In: Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings. pp. 205–211. ECSA '17, ACM, New York, NY, USA (2017), http://doi.acm.org/10.1145/3129790.3129818
15. Munoz, D.J., Pinto, M., Fuentes, L.: Hadas and web services: Eco-efficiency assistant and repository use case evaluation. In: Energy and Sustainability in Small Developing Economies (ES2DE), 2017 International Conference in. pp. 1–6. IEEE (2017)
16. Ochoa, L., Pereira, J.A., González-Rojas, O., Castro, H., Saake, G.: A survey on scalability and performance concerns in extended product lines configuration. In: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems. pp. 5–12. ACM (2017)
17. Pallister, J., Hollis, S., Bennett, J.: Beebs: Open benchmarks for energy measurements on embedded platforms. arXiv preprint arXiv:1308.5174 (2013)
18. Poovey, J., et al.: Characterization of the eembc benchmark suite. North Carolina State University (2007)
19. Poovey, J.A., Conte, T.M., Levy, M., Gal-On, S.: A benchmark characterization of the eembc benchmark suite. IEEE micro 29(5) (2009)
20. Weicker, R.P.: Dhrystone: a synthetic systems programming benchmark. Communications of the ACM 27(10), 1013–1030 (1984)