HADAS: Asistente de eco-eficiencia con repositorio de consumo energético

Daniel-Jesus Munoz, Mónica Pinto y Lidia Fuentes

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

Resumen El interés por la Ingeniería del Software "verde", o sea, sensible al consumo de energía, es relativamente reciente. Su objetivo es concienciar a los desarrolladores de software de la influencia que tienen sus decisiones de diseño e implementación en el gasto energético del producto final. Hasta el momento se han publicado muchos resultados experimentales que comparan el consumo de energía de varias soluciones alternativas, y que demuestran que se puede reducir dicho consumo hasta en un 70 %. Aunque estos resultados sean de libre disposición, no es sencillo que un desarrollador aplique este conocimiento a sus aplicaciones. En consecuencia, en este artículo presentamos el eco-asistente HADAS cuya utilidad es: (i) los investigadores almacenarán sus resultados en un repositorio de libre disposición; (ii) los desarrolladores podrán razonar y obtener las configuraciones que menos energía consuman y que satisfaga sus requisitos. Nos centraremos en mostrar los elementos principales de nuestra propuesta y como se aplica a casos de estudio reales.

Keywords: CVL, Energía, Línea de Productos Software, Variabilidad, Java, Servidor.

1. Introducción

Las políticas eco-eficientes, que persiguen la eficiencia energética, son una preocupación creciente, debido a las emisiones de CO_2 y al efecto invernadero, provocado en parte por los sistemas de información de uso personal e industrial. Aunque el software no consume directamente energía, sí que tiene impacto en el gasto energético del hardware donde se ejecuta (ej.: un dispositivo móvil). Este impacto del software en el consumo de energía no es precisamente despreciable, por lo que recientemente ha surgido un interés por concienciar al desarrollador de software, de la influencia que tienen sus decisiones de diseño e implementación en el gasto energético del producto final. Por tanto, el desarrollador de software no solamente debe tener en cuenta la modularización, rendimiento o mantenibilidad del software, sino que también debe tener en cuenta el ahorro energético.

El interés por las tecnologías de información verdes y en especial por la Ingeniería del Software sensible al consumo energético (Energy-aware Software Engineering [10]) es relativamente reciente. La mayoría de los trabajos sobre eficiencia energética del software son estudios empíricos que se centran en estudiar

qué partes del código son las que consumen más energía [7,14,15], o en comparar el consumo energético de varias soluciones alternativas [2,5,8,10,16]. Existen artículos que comparan el consumo de energía, tanto con métricas reales; en redes con dispositivos Android [13], en servidores [3] o en colecciones Java [8]; como con simulaciones, de aplicaciones Android [11] con eCalc, de aplicaciones Java Enterprise Edition (EE) [4], o de compresión de audio [10,17] con Palladio PCA. De estos trabajos se deduce que vale la pena optimizar el consumo de energía a nivel de código ya que se puede ahorrar entre un 2 % y un 70 %. Sin embargo, aunque todos estos trabajos empíricos muestran resultados muy interesantes, estos no son fáciles de trasladar a los procesos de desarrollo software industrial. Por ejemplo, en [8] se muestra el consumo de energía de las diferentes colecciones de datos (ej.: ArrayList, HashSet, Map, etc.) implementadas utilizando diferentes *frameworks*. Los autores han puesto de libre disposición sus resultados almacenados en ficheros .csv, pero solo con estos ficheros no es sencillo conseguir que un desarrollador reutilice este conocimiento en sus aplicaciones.

En consecuencia, en este trabajo nos planteamos la definición de un eco-asistente (HADAS) que contenga los elementos funcionales que más influyen en el consumo de energía de la aplicación (los *consumidores de energía*), junto con sus alternativas de diseño (ej.: usar el algoritmo de cifrado AES) e implementación (ej.: Apache Commons Crypto). El objetivo del eco-asistente HADAS es el de concienciar al desarrollador de que debe analizar con cuidado las diferentes alternativas de diseño/implementación de estos consumidores, desde el punto de vista del consumo energético, y ayudarle a elegir la mejor. La utilidad del asistente es doble. Por un lado, los investigadores podrán almacenar sus resultados experimentales en un repositorio que quedarán disponibles para el resto de la comunidad. Por otro lado, los desarrolladores podrán utilizar estos para razonar acerca de las distintas versiones de los consumidores de energía, y elegir aquella solución que les satisfaga, siendo conscientes de la energía que podría consumir.

Existen repositorios como el proporcionado por la herramienta de minería GreenMiner [12], orientado a almacenar resultados de pruebas experimentales relativos al consumo de energía. Sin embargo, este repositorio sirve únicamente para la ejecución y consulta de pruebas concretas y queda fuera de su alcance el aconsejarle al desarrollador sobre la solución más eco-eficiente. De hecho, desarrollar un asistente como el que planteamos en este artículo es una tarea muy ambiciosa, a la vez que novedosa. En este artículo mostraremos los elementos principales de nuestra propuesta, centrándonos en presentar la estructura interna del asistente y en cómo este le permite a un desarrollador razonar sobre el consumo de energía de las diferentes configuraciones de los consumidores de energía. Concretamente, este razonamiento se basa en explotar un modelo de variabilidad que almacena las diferentes alternativas de los consumidores de energía y nos permite generar diferentes configuraciones válidas, y los datos de energía de cada una de ellas. Hay que tener en cuenta que el objetivo del asistente no es dar valores concretos de consumo de energía, ya que estos dependen mucho del hardware, sino mostrar en forma de gráficas cómo varía el consumo de energía de dos o más soluciones. Por tanto, se pretende que el desarrollador

identifique tendencias de consumo y no asegurarle que su aplicación consumirá una determinada cantidad de Julios. De hecho, ninguna de las mediciones de energía realizadas tanto con herramientas software como hardware (ej.: Monsoon Power Monitor, osciloscopios) pueden tomarse como valores absolutos dado que el consumo real de una aplicación en ejecución depende de multitud de factores que no pueden controlarse (ej.: temperatura, carga del dispositivo, recolectores de basura, etc.).

Las ventajas del eco-asistente HADAS serían: (i) el desarrollador no necesita disponer de, ni conocer, las herramientas hardware y/o software disponibles para medir el consumo de energía; (ii) el desarrollador no tiene que simular o hacer mediciones del consumo de energía de las diferentes alternativas de los consumidores de energía; (iii) HADAS ofrece una interfaz web sencilla que le ayuda al desarrollador a identificar los consumidores de energía de su aplicación y navegar para configurar las diferentes alternativas; (iv) HADAS devuelve gráficas comparativas que le mostrarán las tendencias de consumo de energía de varias implementaciones alternativas.

El artículo se organiza de la siguiente manera. La Sección 2 describe los requisitos y la visión general del eco-asistente HADAS. La Sección 3 detalla la estructura y funcionamiento de HADAS. En la Sección 4 evaluamos nuestra propuesta mediante dos escenarios diferentes, y mostramos la mejora del análisis de sostenibilidad al usar HADAS. La Sección 5 describe el trabajo relacionado. Finalmente, la Sección 6 discute las conclusiones y el trabajo futuro.

2. Requisitos y Visión General del Eco-Asistente HADAS

En esta sección vamos a describir los requisitos que el repositorio HADAS debe cumplir para que sea útil, tanto a los investigadores que llevan a cabo los experimentos de consumo de energía, como a los desarrolladores que quieren reutilizar los resultados de dichos experimentos en sus aplicaciones. Daremos también una visión general de las partes más relevantes del repositorio.

En cuanto a los requisitos que el repositorio HADAS debería tener en cuenta, hemos identificado los descritos a continuación. Los requisitos R1-R3 son comunes tanto a investigadores como a desarrolladores, mientras que el requisito R4 es específico de los desarrolladores de aplicaciones y el requisito R5 específico de los investigadores en consumo de energía:

R1. Identificar los Consumidores de Energía. El primer paso para poder razonar sobre el consumo de energía de una aplicación es identificar aquellos elementos que de forma recurrente pueden tener un mayor impacto en su consumo energético. Estos elementos serían los que llamamos *consumidores de energía*. HADAS, al menos en esta primera fase, debería incorporar aquellos consumidores que más se utilizan en las aplicaciones, y dejar de lado aquellos más específicos de un dominio particular.

R2. Modelar la Variabilidad de los Consumidores de Energía. Para un mismo referente de consumo energético podría haber disponible distintas soluciones, tanto de diseño como de implementación. Dado que queremos poder comparar entre distintas alternativas para un mismo referente, éstas deben ser modeladas, y su variabilidad representada de forma explícita. Dentro de este modelo también será necesario identificar las dependencias entre distintos consumidores (ej: entre autenticación y distribución).

R3. Registrar el Consumo Energético de los Consumidores de Energía. Junto a cada referente, el repositorio HADAS debe almacenar información que permita calcular y razonar sobre el consumo energético de cada una de sus variantes, teniendo en cuenta los parámetros de configuración que afectan a dicho consumo.

R4. Dar soporte para realizar un Análisis de Eco-eficiencia. El desarrollador debe poder configurar los diferentes consumidores de energía (ej.: frameworks de desarrollo web PHP) de forma parcial, de manera que seleccione aquellas características variables sobre las que ya ha tomado una decisión, porque son imprescindibles de acuerdo a los requisitos de su aplicación (ej.: usar PHP plano) pero deje sin seleccionar otras características donde existe mayor flexibilidad a la hora de elegir entre las alternativas disponibles. De esta forma el repositorio HADAS podría devolverle gráficas comparativas con todas las soluciones posibles. El desarrollador podrá entonces realizar un análisis de la eco-eficiencia de las distintas soluciones y optar por una de ellas.

R5. Dar soporte para integrar los resultados de experimentación. Como ya hemos comentado anteriormente, no sería un objetivo viable el que los desarrolladores del eco-asistente HADAS fueran los encargados de medir y almacenar todas las configuraciones posibles de todos los consumidores de energía que existan en cualquier dominio de aplicación. Para esto, la idea es basarnos en un modelo colaborativo, donde el repositorio esté preparado para incluir las configuraciones y mediciones aportadas por profesionales, investigadores e, incluso, extraídas de artículos ya publicados. Dada la gran variedad de hardware existente y las distintas posibilidades de medición, HADAS debería permitir agrupar aquellas medidas que se hayan realizado con el mismo hardware.

Nuestra visión general de cómo debería ser el eco-asistente HADAS puede verse en la Figura 1. Por un lado, el *Formulario* y el *Modelo de Variabilidad* modelan de forma explícita cada uno de los consumidores de energía contenidos en el asistente, haciendo hincapié en la variabilidad existente y en su contexto energético. El *Formulario* es la interfaz con los desarrolladores, que les hace transparente los detalles del modelo de variabilidad, haciéndoles más fácil su utilización, mientras que el *Modelo de Variabilidad* es una representación formal de la variabilidad de los consumidores de energía y su contexto energético, que permite configurarlos y razonar sobre ellos. Toda la estructura que nos indica el modelo de variabilidad, junto a las mediciones de energía para cada consumidor

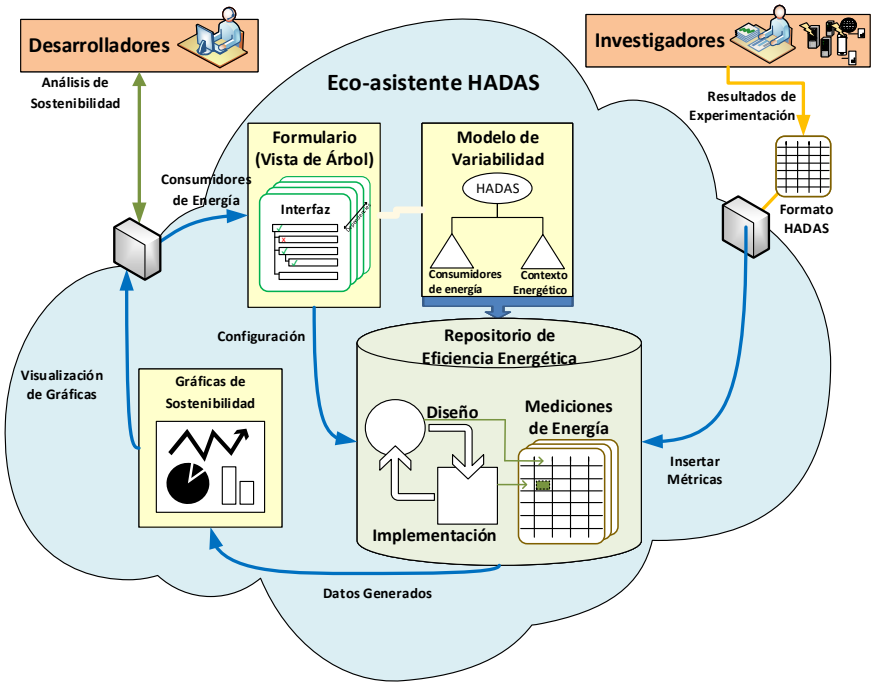

Figura 1. Visión general del eco-asistente HADAS

y cada contexto energético, estarán almacenados en el *Repositorio de Eficiencia Energética*. Una vez generada una *Configuración*, total o parcial, a través del *Formulario* se accederá al repositorio para obtener información sobre las mediciones de energía asociadas a esa configuración, y se generarán las *Gráficas de Sostenibilidad* necesarias para realizar el análisis energético de las variantes seleccionadas.

Como hemos mencionado anteriormente se distinguen claramente dos roles, el de los *Desarrolladores* y el de los *Investigadores*.

¿Cómo utiliza el eco-asistente HADAS un investigador? El investigador dispone de datos de experimentación sobre una variante determinada de un consumidor de energía y quiere integrarlos en el eco-asistente HADAS para que los desarrolladores de aplicaciones puedan utilizarlos. Básicamente, de acuerdo a la Figura 1, el investigador tendrá que representar los datos tomados en un formato entendible por HADAS, y habrá que actualizar de forma adecuada tanto el modelo de variabilidad del consumidor de energí,a como el repositorio con las métricas.

¿Cómo utiliza el eco-asistente HADAS un desarrollador? Un desarrollador de aplicaciones desea conocer cuáles son los posibles puntos de fuga de energía de la aplicación que va a desarrollar, y quiere realizar un análisis eco-eficiente de las distintas alternativas que le da el eco-asistente HADAS. Para ello, navegará por los formularios donde HADAS le muestra información de todos los consumidores de energía que tiene registrado junto a sus variantes. El desarrollador seleccionará qué consumidores de energía quiere analizar, qué variantes quiere pre-selecciona, cuales no para que el asistente le ayude a elegir. Con toda esa información el eco-asistente HADAS generará una configuración parcialmente

instanciada. Esto es posible porque por debajo de este formulario, transparente al desarrollador de aplicaciones, el eco-asistente tiene definido el modelo de variabilidad de todos los referentes de consumo y de todas sus variantes. Además, enlazado con este modelo de variabilidad, estará la información de las mediciones de energía que hay disponibles. Una vez definida una configuración de los consumidores de energía que se desean analizar, el eco-asistente generará las gráficas que permitirán al desarrollador realizar su análisis de sostenibilidad.

3. Estructura de HADAS

En la Figura 1 podemos visualizar que, tanto las métricas como su estructura (características de diseño e implementación), se encuentran en una base de datos, formando lo que hemos denominado repositorio de eficiencia energética. La estructura del repositorio viene dada por el modelo de variabilidad, que en nuestro caso, sería una representación del árbol de variabilidad en un formato manejable para el eco-asistente HADAS. Hemos elegido una representación en modo texto, concretamente el formato EFM (Famas Extended Model) [18]. De esta forma es fácil chequear su corrección utilizando la herramienta FaMa-FW (del inglés Feature automated Model analyses - FrameWork) [1]. Como se ve en

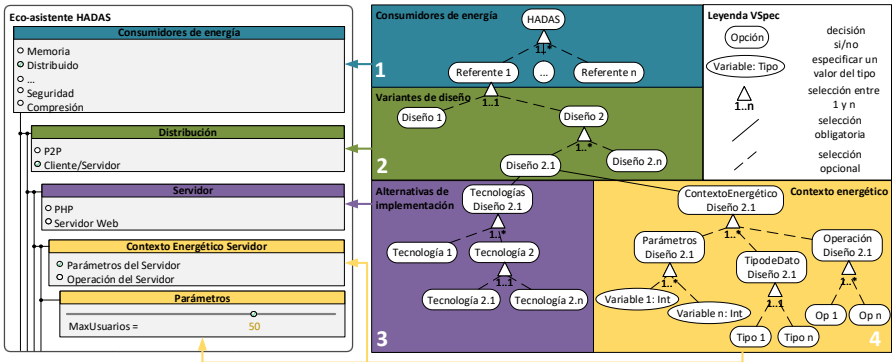

Figura 2. Estructura por niveles de la variabilidad del árbol y del formulario

la parte derecha de la Figura 2, y también en el ejemplo de la Figura 3, el árbol de variabilidad que usa HADAS se divide en tres niveles: (1) en este nivel se sitúan los consumidores de energía identificados no solamente por nosotros, sino en los diferentes trabajos de experimentación (ej.: seguridad, almacenamiento, distribución ...). en el primer nivel de la Figura 3); (2) para cada referente se definirá un sub-árbol con las opciones de diseño disponibles (ej.: en la Figura 3 para la distribución tenemos la arquitectura cliente/servidor). Dependiendo de la complejidad del consumidor de energía habrá uno o varios sub-niveles; (3) para cada hoja del sub-árbol de diseño (ej.: Diseño 2.1 en Figura 2) se define un sub-árbol *Tecnologías* y otro sub-árbol con el *Contexto Energético*. El sub-árbol *Tecnologías* define las diferentes alternativas que existen para implementar cada uno de los diseños. Aquí podemos distinguir entre diferentes frameworks, APIs,

etc., y sus respectivos lenguajes de programación. Esta información es la que nos permitirá darle al desarrollador el consumo de energía de un diseño concreto, implementado con diferentes tecnologías. El sub-árbol *Contexto Energético* contiene tres características hijas: (i) *Parámetros*: el consumo de energía de una implementación varía en función de algunos parámetros de su diseño, como por ejemplo la compresión de un fichero dependerá del tamaño del fichero a comprimir, por lo tanto, el desarrollador podría indicar un tamaño máximo; (ii) *Tipo de Datos*: normalmente los consumidores de energía manipulan datos de diferente tipo (ej.: texto plano, XML, etc.) y el tipo de datos suele influir en el consumo de energía; (iii) *Operaciones*: cada diseño de un referente ofrece una interfaz de uso, que es el conjunto de operaciones que se invocarán desde la aplicación (ej.: para las colecciones de datos, las operaciones son insertar, borrar, etc.). Tanto las operaciones, como el tipo de datos, no tienen por qué estar implementadas en todas las tecnologías, y esto lo expresamos mediante restricciones entre sub-árboles del tipo `Tecnología i implies NOT Op j`.

Una vez definimos nuestro modelo de variabilidad, este hay que representarlo mediante formularios dinámicos al desarrollador y permitirle que seleccione las diferentes alternativas, respetando las restricciones entre sub-árboles (ver parte izquierda de la Figura 2, que sigue la misma leyenda de colores que el esquema del árbol de variabilidad). Concretamente, el eco-asistente HADAS hace una representación web del modelo de variabilidad, en forma de árbol desplegable en profundidad. El formulario dinámico implementa las dependencias de manera dinámica, es decir, se marcarán y desplegarán las ramas objeto de restricciones, de manera automática, si son activadas.

Pero este árbol de variabilidad no contiene las métricas, sino que éstas se almacenan en una base de datos. Por lo tanto, conectamos el árbol de variabilidad a una base de datos que, conjuntamente, conforman el repositorio. La estructura de la base de datos se corresponde con la del modelo de variabilidad, donde cada nivel tiene su referente en una tabla concreta (tablas Referente, Diseño, Tecnología, TipodeDato, Operación, Variable). Además, la tabla Métrica, relacionará estas tablas con el consumo y el tiempo de ejecución. Para permitir la inclusión de datos de energía diferentes (de distintas fuentes) para la misma configuración, se incluye la tabla Metadato, donde se incluirá el sistema de medición y el hardware usado para obtener éstos datos, lo que le permitirá al desarrollador realizar un análisis más rico, al poder basarse en varias fuentes.

El desarrollador va generando diferentes configuraciones que se traducen a consultas en la base de datos del repositorio. Las configuraciones se generan con un razonador (*SAT solver*) que nos garantiza que se generan todas las configuraciones válidas. A continuación, se formaliza la correspondencia entre una configuración del modelo de variabilidad y la información en la base de datos. Esta correspondencia permite la generación automática de las consultas SQL, que le darán al desarrollador la información necesaria para realizar un análisis. Nos basamos en la formalización del árbol de variabilidad definida en [6] extendiéndola. Dada una configuración parcial $C_{Parcial}$ del modelo de variabilidad y después de aplicarle un razonador (FaMa-FW) obtendremos un conjunto con

todas las configuraciones correspondientes. Para cada configuración completa $C_{Completa}$ se definen los siguientes conjuntos y operaciones:

VSPEC. Conjunto finito, no vacío, de identificadores (nombres únicos) de todas las características (VSpecs) de una configuración completa.
$VSPEC = \{HADAS, Distribución, Cliente/Servidor, Servidor, TecnologíasServidor, PHP, PlainPHP, Lavarel, ...\}$

hijos $: VSPEC \rightarrow P(VSPEC)$. Función que dada una característica, devuelve todos sus hijos.

padre $: VSPEC \rightarrow VSPEC$. Función que dado una característica devuelve la característica padre, o $\emptyset$ si es la raíz.

antecesores $: VSPEC \rightarrow P(VSPEC)$. Función que dado una característica devuelve todos sus antecesores, o $\emptyset$ si es la raíz.

Referentes $: C_{Completa} \rightarrow VSPEC$. Dada una configuración completa devuelve los referentes de consumo de energía de la configuración, es decir: $Referentes = \{c \in C_{Completa} \mid c \in hijos(HADAS)\}$

$\boldsymbol{TEC_{Raíz}} : C_{Completa} \rightarrow P(VSPEC)$. Dada una configuración completa devuelve las características raíz de las tecnologías: $TEC_{Raíz} = \{c \in C_{Completa} \mid c.startsWith(\text{"}Tecnologías\text{"})\}$

$\boldsymbol{V_{Diseño}} : C_{Completa} \rightarrow P(VSPEC)$. Dada una configuración completa devuelve todas sus variantes de diseño: $V_{Diseño} = \{c \in C_{Completa} \mid \exists t \in TEC_{Raíz}, c = padre(t)\}$

Tecnologías $: C_{Completa} \rightarrow P(VSPEC)$. Dada una configuración completa devuelve todas las tecnologías usadas en la configuración: $Tecnologías = \{c \in C_{Completa} \mid hijos(c) = \emptyset \wedge \exists a \in antecesores(c) \wedge a \in TEC_{Raíz}\}$

$\boldsymbol{PARAM_{Raíz}} : C_{Completa} \rightarrow P(VSPEC)$. Dada una configuración completa devuelve las características raíz de los parámetros: $PARAM_{Raz} = \{c \in C_{Completa} \mid c.startsWith(\text{"}Parámetros\text{"})\}$

Parámetros $: C_{Completa} \rightarrow P(VSPEC)$. Dada una configuración completa devuelve todos los parámetros usados en la configuración: $Parámetros = \{c \in C_{Completa} \mid hijos(c) = \emptyset \wedge \exists a \in antecesores(c) \wedge a \in PARAM_{Raíz}\}$

value $: p \rightarrow \mathbb{R}$. Dado un parámetro p devuelve el valor seleccionado por el desarrollador en la interfaz.

De manera análoga a la definición de las variantes de parámetros se definen los contextos energéticos, tipos de datos y operaciones.

Con esta información, se genera una consulta SQL para cada configuración completa $C_{Completa}$:

```
SELECT Energy, Time, Metadatos.* FROM TMétrica
JOIN Metadatos ON Metadatos.idMétrica = TMétrica.id
JOIN RefsMétrica ON RefsMétrica.idMétrica = TMétrica.id
JOIN TReferente ON TReferente.id = RefsMétrica.idReferente
⇑ Enlazamos una medida a uno o varios referentes
JOIN VsDisMétrica ON VsDisMétrica.idMétrica = TMétrica.id
JOIN TVDiseño ON TVDiseño.id = VsDisMétrica.idVDiseño
⇑ Enlazamos una medida a una o varias variables de diseño
JOIN TecsMétrica ON TecsMétrica.idMétrica = TMétrica.id
JOIN TTecnología ON TTecnología.id = TecsMétricas.idTecnologia
⇑ Enlazamos una medida a una o varias alternativas tecnológicas
... ⇒ De manera análoga enlaza para cada tabla/conjunto
WHERE ((∀r ∈ Referentes, TReferente.nombre = r) AND
(∀v ∈ VDiseño, TVDiseño.nombre = v) AND
(∀t ∈ Tecnologías, TTecnología.nombre = t) AND
... ⇒ De manera análoga filtra para cada tabla/conjunto
(∀p ∈ Parámetros, TParámetro.nombre = p AND
TParámetro.valor ≤ value(p))) ⇒ Acota por valores máximo
```

Una vez obtenidos los datos para cada configuración, el sistema hará uso de la API Google Charts para mostrar las gráficas de consumo y tiempo de ejecución pertinentes, permitiendo el análisis del caso de uso.

4. Evaluación

En esta sección presentamos dos escenarios de uso de HADAS. En el primero mostramos que es posible integrar los datos generados por otros investigadores en el eco-asistente, mejorando el análisis de consumo de energía que se puede realizar, y en el segundo seguimos el proceso completo generando y almacenando primero los datos y usando después el eco-asistente.

4.1. Colecciones en Java

En [8] se calcula el consumo de energía de distintas colecciones de Java (Lista, Mapa y Set) para diferentes operaciones (insertar al principio, en medio, al final, borrar, iterar...), utilizando distintas APIs (JCF, ACC, Trove) y distintas implementaciones (HashMap, TreeMap...). Se realiza la experimentación teniendo en cuenta distintos tipos de datos (Int, Object) y tamaño de los datos (hasta 5000 elementos). Toda esta información está accessible mediante archivos CSV en una web.

¿Cómo utilizaría entonces un desarrollador esta información sin usar HADAS? Puede consultar las gráficas estáticas proporcionados en la página web, pero no es trivial identificar todos los posibles análisis que existen, ya que no hay una forma fácil de conocer todas las variantes a analizar. Nosotros mostraremos como se pueden integrar estos datos en el eco-asistente y veremos que es posible realizar un análisis más rico del consumo de energía de las colecciones con dichas métricas.

Dado que no existe información de estos referentes energéticos en el actual repositorio, el primer paso es evolucionar el árbol de nuestro modelo de variabilidad. Por tanto, en la Figura 2, al nivel de variantes de diseño, bajo la rama Memoria->Volátil incluiríamos los tres tipos List, Map y Set), y, en cada uno, las alternativas tecnológicas, es decir, los tres frameworks usados en esta experimentación (JCF, ACC, Trove) y, de estos últimos, sus respectivas implementaciones (HashMap, TreeMap...). También se incluye el contexto energético de memoria volátil, donde contemplamos el máximo número de datos, el tipo de datos (Int, Object) y las operaciones. Para introducir los datos en el repositorio hemos implementado también una función de filtrado, la cual extrae los datos de diseño, implementación y energía que necesitamos de todos los CSV (tipo de dato, operación, clase, implementación, consumo en julios y tiempo de ejecución). A partir de estos datos se calcula la mediana de los Julios (ya que existen 32 pasadas para cada configuración completa), y esto es lo que añadiríamos a la base de datos de nuestro repositorio. Tras introducirlo en HADAS, junto a las restricciones necesarias (ej.: Map implica no Insert at the beginning|end), hemos evaluado el árbol de variabilidad EFM correspondiente a este caso de estudio con FAMA-FW, y nos devuelve 11904 configuraciones correctas. Este alto nivel de variabilidad demuestra que no es viable realizar el análisis de forma manual, y, también, que la información proporcionada en [8] es muy interesante, pero que se podría ampliar analizando muchas otras opciones no contempladas por los autores en [8].

¿Cómo utilizaría un desarrollador la información de consumo de energía usando HADAS? Primero seleccionaría en el formulario una configuración de las distintas opciones disponibles para las colecciones en las que esté interesado. Simplemente, haciendo esto, se le proporcionará de forma automática información de todas las variantes de las que se tiene información. Las gráficas de sostenibilidad las generará el eco-asistente también de forma automática, no limitándose a las proporcionas en la web con los datos de [8]. Ejemplos de alguno de los análisis que se podrían llevar a cabo y cuya información no está disponible directamente – i.e. nos obligaría a entrar en los CSV, entender su formato y generar nosotros las gráficas, podrían ser:

- El consumo, en una sola gráfica, de insertar al principio/medio/final para las distintas implementaciones del framework Trove, para Enteros.
- Una gráfica más detallada para, como máximo, 500 objetos, de una sola implementación pero para todas sus operaciones.
- El consumo para todas las implementaciones de Mapa, para la operación Iterar, para el tipo Objeto, junto al tiempo, para poder analizar el trade-off.

4.2. Servicio web

Un nuevo caso de estudio sería el desarrollo de servicios web. El primer paso es evolucionar nuestro árbol de variabilidad tras identificar los consumidores de energía y su contexto energético. El resultado lo podemos visualizar en la Figura 3, donde vemos que hemos podido modelar la información siguiendo la estructura en tres niveles definida en la Figura 2. En esta evolución se ha incluido `(Cliente/Servidor,{Servidor, Cliente}),(P2P,{})` como variantes de diseño del referente `Distribución`. Se han incluido también distintas alternativas tecnológicas del lenguage PHP (base o frameworks) y de servidores web. En el contexto energético el parámetro a tener en cuenta es el número de usuarios. Para `(Operación Servidor,{Gestión Datos})` hemos identificado una dependencia con el alojamiento de datos en el servidor. La base de datos también se ha actualizado con las características de nuestro modelo Nuestro rol comienza sien-

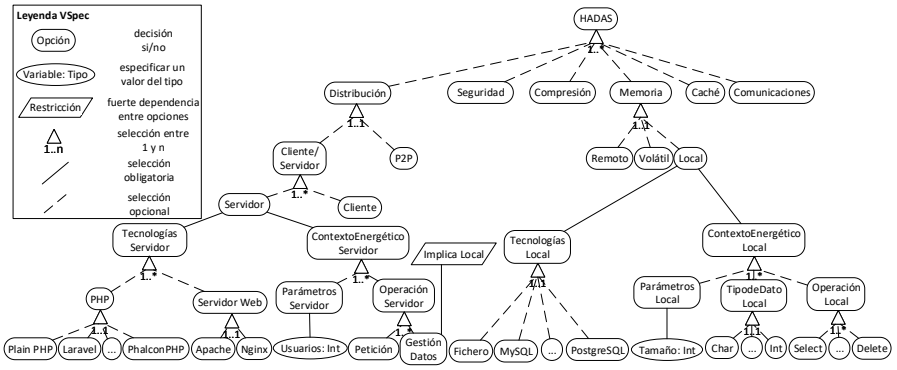

Figura 3. Caso de estudio de una aplicación web

do el de investigador, por tanto, se han realizado diversas mediciones de consumo de CPU con la herramienta Intel Platform Power Estimation Tool (IPPET), sobre un procesador Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz en modo alto rendimiento. Se han usado las versiones más recientes de cada software de 64 bits, y las peticiones al servidor se han realizado en localhost sobre Windows 10. La petición es un código web formado por la página de presentación Jumbotron de Twitter Bootstrap 4.0, a la que le hemos incrustado un benchmark de PHP que simula un escenario equilibrado genérico con operaciones matemáticas, manipulación de cadenas, bucles y sentencias de control, que inducen una carga moderada en PHP (superior a una página web estándar). Para aislar el efecto de un explorador web, en la realización de las pruebas de carga, se ha usado la herramienta JMeter para hasta 18 usuarios concurrentes. Se han usado las configuraciones por defecto de cada tecnología.

¿Cómo se beneficia un desarrollador de estas métricas? En nuestro de rol de desarrollador, tenemos ya un servicio web en nuestro propio alojamiento (variante `Referentes=(Distribución, VDiseño=(Cliente/Servidor,{Servidor})`), pero tenemos un alto coste energético y buscamos reducirlo. Ya está desarrollado en PHP plano, por tanto incluir un framework no nos interesa. En cambio, no tenemos problemas con cambiar de servidor web, aunque actualmente usamos Nginx debido a ser una alternativa a Apache más liviana (seleccionaríamos `Tecnologías=(PHP,{PHP Plano},Servidor web,{}))`. Es un servicio de páginas y cálculos que solo depende de los datos del usuario, sin almacenamiento, por tanto se selecciona solo la operación de petición (`Operaciones=(Petición)`). No queremos acotar por número de usuarios, dado que la carga es variable (para ello `Parámetros=()`). Tras rellenar el formulario del eco-asistente con esta configuración parcial, FaMa-FW genera dos completas: `Tecnologías=(PHP,{PHP Plano}, Servidor web,{Apache})` y `Tecnologías=(PHP,{PHP Plano},Servidor web,{Nginx})`. Esto genera dos consultas SQL, cuyos resultados se envían a la API Google Charts, generando las gráficas de la Figura 4. Además, se adjuntan los metadatos en el resultado, tanto de las versiones usadas, como del hardware y sistema operativo usado. Un

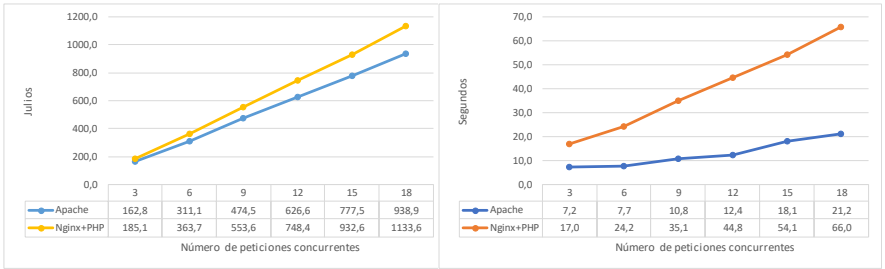

Figura 4. Consumo y tiempo de ejecución de Apache y Nginx, junto a PHP

análisis de sostenibilidad, a priori, podría ser el siguiente. Para varios usuarios hay una notable diferencia de tiempo de ejecución en ambos. Ésta resulta en un

consumo de energía mayor de Nginx con respecto a Apache. La explicación técnica, a posteriori del ánalisis, es que si bien Nginx es más liviano y trabaja mejor con servicios de baja carga PHP (por ejemplo, un blog realizado en Wordpress), para servicios más intensivos (que impliquen cálculos matemáticos complejos o manipulaciones de tipos de datos) Apache es mucho más veloz, compensando su mayor proporción de consumo con respecto al tiempo. Ésto es porque Apache carga en cada uno de sus procesos (en memoria RAM) un intérprete PHP, en cambio Nginx delega ese trabajo al intérprete PHP-CGI. Por tanto, en el rol de investigador, como puede verse en la Figura 4 el consumo de Apache es el consumo de sus procesos, dado que el proceso de PHP consume milésimas de julios. En cambio, para Nginx, se han realizado y sumado las medidas de los procesos de Nginx más los procesos de PHP.

La ventaja de HADAS es apreciable aquí, donde el desarrollador recibe el soporte necesario para decidir realizar un cambio en su servidor web, y comenzar a rebajar el consumo de energía de sus aplicaciones web. Todo esto al margen de las tablas de medición proporcionadas por los investigadores, y del análisis que estos pudieran realizar y publicar (ej.: en este caso el análisis indica que la clave del consumo energético está en si el intérprete reside o no en el proceso).

5. Trabajo relacionado

El eco-asistente HADAS que se presenta en este trabajo se basa en: (i) la creación de un repositorio de consumidores de energía; (ii) el procesamiento de las métricas que sustente un análisis del consumo de energía de diferentes alternativas de diseño e implementación. Existen algunos trabajos que resaltan la importancia de disponer de un repositorio con métricas de energía software como Selflab [9] que contempla como trabajo futuro desplegar un repositorio que ofrezca varias configuraciones al desarrollador. Uno de los más utilizados es el ofrecido por GreenMiner [12]. GreenMiner es una solución completa ya que proporciona herramientas de medición de energía, y también un repositorio donde almacena los datos recolectados y es posible aplicar técnicas de minería de datos. En este trabajo se comenta la dificultad de procesar los datos de energía ya que hay una falta de normalización de los datos extraídos y no se define un formato base en el que puedan casar los resultados de todos los experimentos. Además, este repositorio es local al sistema de medición que hay que desplegar de manera individual. En HADAS, en cambio, proponemos un enfoque colaborativo, donde se vuelquen los resultados de trabajos realizados por diferentes investigadores. Un enfoque similar se sigue en [7], donde se destaca la utilidad de un hipotético repositorio en la nube que ayude a identificar el diseño, arquitectura y componentes, a nivel software, culpables de fugas energéticas. Los autores comentan la dificultad de realizar un análisis de los datos de energía para un caso de uso concreto, para lo cual definen un *Energy modeller* que les permite restringir los resultados de búsqueda del repositorio. En [14] se propone un repositorio de componentes para servicios de la Internet de las Cosas (IoT) en vez de un repositorio genérico como el de HADAS. En este artículo usan uno

para buscar componentes que consuman poco y que proporcionen una interfaz requerida y proporcionada según lo especificado en la arquitectura de un servicio IoT. Este enfoque no incluye mediciones de implementaciones concretas como propone HADAS, sino que razona solamente a nivel de arquitectura, por lo que su utilidad real es limitada.

Por otro lado, en relación al análisis sostenibilidad, existen varios trabajos. En [2] se compara el consumo de energía de diferentes tecnologías usadas en servicios web (ej.: Apache, PostgreSQL y PHP) basándose fundamentalmente en medir el tiempo en que los componentes del servicio web están en uso. En [5] se comprueba que, de todos los protocolos de red, HTTP/2 es el más energéticamente eficiente del lado del cliente, dada la reducción de peticiones al servidor web, independientemente del cliente web. Sin embargo, también resulta interesante evaluar este protocolo del lado del servidor como se hace en [16]. En este trabajo se comparan varios protocolos y servidores web, concluyendo que HTTP/2 es el que menos consume especialmente si se usa en conjunto con el servidor web H2O. En el caso de estudio de las colecciones Java [8], se realizan mediciones y se comparan las distintas implementaciones de los distintos frameworks, y se concluye que la configuración ideal depende del contexto energético (ej.: el tipo y tamaño de datos y la operación). Aunque en todos estos trabajos que comparan diferentes implementaciones se sacan conclusiones interesantes, este conocimiento no se almacena en ningún sitio, y por tanto es difícil que llegue a los desarrolladores. Con HADAS pretendemos suplir esta carencia.

6. Conclusión y trabajo futuro

En este artículo hemos presentado el eco-asistente HADAS. Se trata de una herramienta que explota las ventajas de los modelos de variabilidad para razonar y elegir entre diferentes alternativas de diseño e implementación teniendo en cuenta el consumo de energía. Desde el punto de vista de los investigadores, HADAS les permite que sus datos sean accesibles a través de un repositorio común. Para los desarrolladores, interesados en "medir" el consumo de energía de sus aplicaciones, HADAS les permite conocer mejor todas las variantes de las que dispone y realizar un análisis más rico de la sostenibilidad del software.

Este artículo demuestra que un asistente como HADAS es realmente útil tanto para investigadores como para desarrolladores. Como trabajo futuro nos planteamos mejorar el asistente, que de momento es un prototipo, hasta conseguir una versión completamente funcional, que pondremos a disposición de investigadores y desarrolladores de aplicaciones para que puedan probar sus ventajas.

Agradecimientos

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

Referencias

1. Benavides, D., Segura, S., Trinidad, P., Cortés, A.R.: FAMA: Tooling a Framework for the Automated Analysis of Feature Models. VaMoS) 2007-01, 129–134 (2007)
2. Bertini, L., Leite, J.C.B., Mossé, D.: Statistical QoS guarantee and energy-efficiency in web server clusters. Euromicro Conference on Real-Time Systems pp. 83–92 (2007)
3. Bianchini, R., R. Rajamony: Power and Energy Management for Server Systems. IEEE 0018-9162 04, 1–11 (2004)
4. Brunnert, A., Vögele, C., Krcmar, H.: Automatic performance model generation for java enterprise edition (EE) applications. Lecture Notes in Computer Science 8168 LNCS, 74–88 (2013)
5. Chowdhury, S.A., Sapra, V., Hindle, A.: Client-side Energy Efficiency of HTTP / 2 for Web and Mobile App Developers. International Conference on Software Analysis, Evolution, and Reengineering pp. 529–540 (2016)
6. CVL Submission Team: Common Variability Language (CVL), OMG revised submission. http://www.omgwiki.org/variability/ (2012)
7. Djemame, K., Armstrong, D., Kavanagh, R., Ferrer, A.J., Perez, D.G., Antona, D., Deprez, J.C., Ponsard, C., Ortiz, D., Macias, M., Guitart, J., Lordan, F., Ejarque, J., Sirvent, R., Badia, R., Kammer, M., Kao, O., Agiatzidou, E., Dimakis, A., Courcoubetis, C., Blasi, L.: Energy efficiency embedded service lifecycle: Towards an energy efficient cloud computing architecture. CEUR Workshop 1203, 1–6 (2014)
8. Fallis, A.: Energy Profiles of Java Collections Classes - PHD. Journal of Chemical Information and Modeling 53(9), 1689–1699 (2013)
9. Ferreira, M.A., Hoekstra, E., Merkus, B., Visser, B., Visser, J.: Seflab: A lab for measuring software energy footprints. Green and Sustainable Software (GREENS) pp. 30–37 (2013)
10. Gamez, N., Horcas, J.M., Pinto, M., Fuentes, L.: A green program lifecycle supporting energy-efficient applications. CoRR abs/1612.0, 12 (2016)
11. Hao, S., Li, D., Halfond, W.G.J., Govindan, R.: Estimating Android applications' CPU energy usage via bytecode profiling. Green and Sustainable Software (GREENS) pp. 1–7 (2012)
12. Hindle, A., Wilson, A., Rasmussen, K., Barlow, E.J., Campbell, J.C., Romansky, S.: GreenMiner: a hardware based mining software repositories software energy consumption framework. MSR pp. 12–21 (2014)
13. Kalic, G., Bojic, I., Kusek, M.: Energy consumption in android phones when using wireless communication technologies. MIPRO pp. 754–759 (2012)
14. Kim, D., Choi, J.Y., Hong, J.E.: Evaluating energy efficiency of Internet of Things software architecture based on reusable software components. International Journal of Distributed Sensor Networks 13(1), 155014771668273 (2017)
15. Manotas, I., Pollock, L., Clause, J.: SEEDS: a software engineer's energy-optimization decision support framework. In: International Conference on Software Engineering (ICSE). pp. 503–514. ACM Press, New York, New York, USA (2014)
16. Sapra, V., Hindle, A.: Web Servers Energy Efficiency Under HTTP/2. PeerJ Preprints (May), 1–18 (2016)
17. Stier, C., Koziolek, A., Groenda, H., Reussner, R.: Model-Based Energy Efficiency Analysis of Software Architectures. ECSA 9278, 221–238 (2015)
18. Vinueza, M., Rodas, J.L., Galindo, J.A., Benavides, D.: El uso de modelos de características con atributos para pruebas en sistemas de alta variabilidad: primeros pasos. CEDI 2016 (2016)