Multi-SPLOT: Un Configurador Web con soporte a Múltiples Usuarios

Sebastian Velásquez-Guevara
Universidad Piloto de Colombia, Colombia.
Email: rlxsebas@gmail.com

***Resumen*—En los últimos años, las empresas han pasado de contar con un único producto para todos sus clientes, a ofrecer productos personalizados y diferentes para cada uno de ellos. Estas compañías brindan al cliente Sistemas de Configuración que permiten al cliente decidir y descartar que características desean obtener en su producto final. Sin embargo, aunque estos sistemas soportan decisiones individuales, no ofrecen un soporte suficiente cuando varias personas desean decidir sobre el mismo producto. Este artículo presenta Multi-SPLOT, un sistema de configuración web que soporta decisiones simultaneas de múltiples usuarios. Este sistema usa sistemas de solver para determinar si las decisiones no tienen conflictos entre sí y para proponer soluciones cuando las decisiones de un usuario contradicen las de otros. El artículo muestra el diseño general de la aplicación y detalles sobre el desarrollo de la misma usando Angular, Firebase y el optimizador lineal en Google App Script.**

***Abstract*—Nowadays, companies have gone from offering a single product for all their clients, to offer different customized for each one. These companies provide Configuration Systems where a user can decide and discard which features she wants in her final product. However, although these systems support individual decisions, they do not offer an special support for decisions made by multiple users for the same product. This paper presents Multi-SPLOT, a Configuration System that supports simultaneous decisions from multiple users. This system uses off-the-shelf solvers to determine if these decisions are not conflicting among them, and to propose solutions when the decisions of an user conflict with decisions of the others. This paper shows the design of the solution and details of its implementation using Angular, Firebase and optimization library in Google App Script.**

***Palabras clave*—Modelos de Características, Sistemas de Configuración**

I. Introducción

Instalar y Configurar una aplicación compleja puede tomar varios meses e involucrar varios usuarios. La configuración y puesta en funcionamiento de un sistema como SAP toma en promedio de 6 a 12 meses [1]. Diversos usuarios, con diferentes áreas de experiencias y requerimientos, (e.g. ingenieros, contadores y expertos en manufactura) deben coordinar su trabajo para lograr una instalación exitosa de un software como este.

En un Sistema de Configuración (a.k.a. un Configurador), los usuarios seleccionan las opciones que desean incluir e ingresan valores para los diferentes parámetros exigidos en el producto. El Software revisa que las selecciones del usuario no tengan conflictos entre sí, es decir que se puedan incluir al mismo tiempo en el producto, y ofrece recomendaciones al usuario en caso que se presente algún problema [2]. Sin embargo, estos sistemas consideran todas las decisiones como si fuesen tomadas por la misma persona, y no tienen en cuenta que algunas veces varias personas toman decisiones sobre el mismo producto [3].

Normalmente, cuando varias personas toman decisiones sobre el mismo producto, las selecciones de un usuario pueden limitar las opciones que pueden escoger los otros usuarios. El sistema desactiva las opciones que entran en conflicto con las decisiones ya tomadas. Es posible que los usuarios que llegan después no puedan seleccionar todas las opciones que quieren sino solo aquellas que no generan conflictos con las decisiones de los otros. Es posible también que esos usuarios no entiendan porque no pueden seleccionar ciertas opciones o no se sientan satisfechos con las opciones que pueden seleccionar. En la actualidad, configuradores como SPLOT [1] y FaMa [2] permiten que varios usuarios tomen decisiones, pero las decisiones tomadas por los primeros usuarios limitan las que pueden ser tomadas por los siguientes.

Una mejor alternativa puede ser que cada usuario exprese sus selecciones y preferencias libremente. Un proceso automatizado puede tomar las decisiones de los diferentes usuarios y, en caso de encontrar algún conflicto, buscar una solución que intente satisfacer en una gran medida a todos los usuarios. Este artículo presenta Multi-SPLOT, una Sistema de Configuración que permite que cada usuario defina las características deseadas para el producto de forma independiente, revisa si las decisiones de cada usuario no tienen conflictos entre sí, y ofrece procesos automáticos que buscan una configuración que satisfaga en mayor medida las decisiones de todos ellos.

El resto de este artículo está organizado de la siguiente forma: La Sección II presenta antecedentes de los sistemas de configuración y los modelos de características. La Sección III describe los problemas que ocurren cuando varias personas participan del proceso de configuración, Las Secciones IV y V introducen nuestra propuesta y describen el diseño del software correspondiente. Finalmente, la Sección VI concluye el artículo.

II. Antecedentes

A. Proceso de Configuración

El proceso de especificar las características para un producto personalizado se conoce como *Configuración*. De acuerdo con Falkner et al. [2],"la configuración es una forma básica de la actividad de diseño, en el cual el producto-objetivo se arma

[1] http://www.splot-research.org/
[2] http://www.isa.us.es/fama/

a partir de un conjunto de partes predefinidas de una manera que es consistente con un conjunto dado de restricciones."

Los procesos de configuración son ampliamente utilizado en sectores como los componentes electrónicos, el sector automotriz y el desarrollo de software. En estos sectores, la gran cantidad de variantes (i.e. la variabilidad) dificulta la especificación correcta de los productos personalizados. Por ejemplo,"Renault es un fabricante de automóviles francés que sirve 118 países con vehículos que tiene un total de 1021 variantes. Un catálogo impreso completo contiendo los 5'000.000 variantes en un formato de libreta de teléfonos de 4cm de espesor daría una pila del tamaño de la distancia entre la Tierra y Plutón. Encontrar el vehículo que coincide con todos los requisitos del cliente en un catálogo como ese sería bastante ¡una aventura! El deber del sistema de configuración es para hacer la selección de un producto que pueda realizarse fácilmente por los seres humanos [4]."

En la actualidad existen varias formas de construir estos Sistemas de Configuración [2]. Por ejemplo, se pueden construir a partir de conjuntos de reglas que determinan dependencias entre los componentes del producto o a través de modelos que representan la variabilidad. En el área de Ingeniería de Software, los modelos de variabilidad más utilizados son los modelos de características [5].

B. *Modelos de Características*

Los *Modelos de Características* (*Feature Models*, en inglés) especifican las diferentes características y propiedades de una familia de productos y sus posibles combinaciones [6].

Sintáxis: Un Modelo de Características se representa por una estructura jerárquica, i.e. un árbol. Cada elemento del árbol es una característica o un grupo de características. La raíz es conocida como el *Concepto* (i.e. el producto a configurar). Los otros elementos son *Características* del producto o *Grupos de Características*.

La Tabla I muestra los elementos que pueden componer un modelo de características:

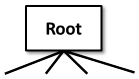
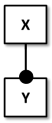
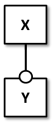
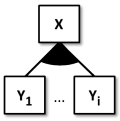
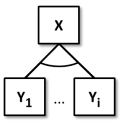
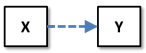
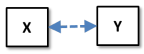
- La *Raíz*, es el *concepto* representado en el modelo,
- *Características Obligatorias* representan características que deben ser seleccionadas cuando la característica padre es seleccionada,
- *Características Opcionales* son características que pueden ser seleccionadas o no cuando la característica padre es seleccionada,
- *Grupos Or*, donde una o más características pueden ser seleccionadas, y
- *Grupos de Alternativas*, donde solo una de las características puede ser seleccionada al tiempo.

Además, el modelo puede contener relaciones entre las características para denotar restricciones:

- relaciones de *requiere*, que indican que una característica debe ser seleccionada cuando otra es seleccionada, y
- de *excluye*, representando que dos características no pueden ser seleccionadas al tiempo.

La Figura 1 muestra un ejemplo de modelos de características para Teléfonos Celulares. El modelo tiene una característica raíz *Cellular Phone* con tres características obligatorias: *LCD*, *Input Device* y *Battery* y una característica opcional *External Memory*. A su vez, un *LCD* puede ser *Normal* o *Touch Screen* pero no ambos. El *Input Device* puede ser *Keypad*, *Stylus*, o ambos. Hay una relación de *excluye* indicando que cuando un teléfono incluye un *Stylus*, no puede incluir un pantalla *Normal* (debe incluir una pantalla *Touch Screen*). Finalmente, la *Battery* puede ser *Small Size* o *Large Size*. Hay una relación *requires* indicando que incluir en el celular una pantalla *Touch Screen* requiere incluir también una *Large Size Battery*.

Tabla I. Elementos del Modelo de Características

Elemento	Figura	Descripción
Raíz		Concepto o Producto a Configurar. Debe ser seleccionado siempre.
Obligatorio		Si X es seleccionado, Y debe estar seleccionado
Opcional		Si X es seleccionado, Y puede ser (o no) seleccionado
Grupo *Or*		Si X es seleccionado, uno o más de $Y_1, ..., Y_n$ debe ser seleccionado
Alternativas		Si X es seleccionado, solo uno de $Y_1, ..., Y_n$ debe ser seleccionado
Relación	**Figura**	**Descripción**
Requiere		Si X es seleccionado, Y debe ser seleccionado
Excluye		Si X es seleccionado, Y no puede ser seleccionado

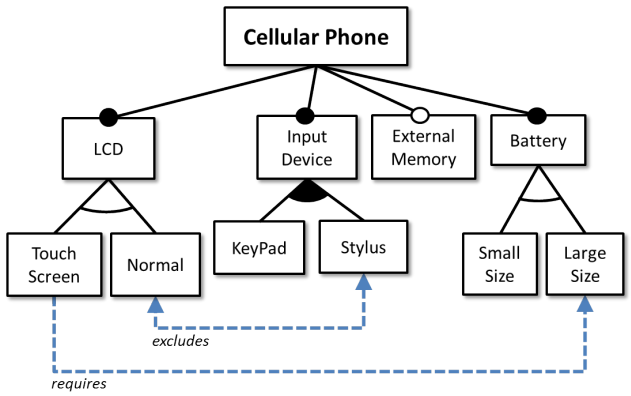


Figura 1. Ejemplo de un modelo de Features Básico

Semántica: Un Modelo de Características representa un conjunto de configuraciones.

Una *Configuración de Características* (i.e. una Configuración) es un conjunto de características de un modelo de características.

Una *Configuración Válida* es una selección de características que cumple con las restricciones definidas en el modelo de características respectivo. Por ejemplo, considerando el Modelo de Características mostrado en la Figura 1, la siguiente configuración C_v, es una configuración válida ya que satisface todas las reglas definidas en el modelo.

$$C_v = \{CellularPhone, LCD, TouchScreen, InputDevice, Stylus, Battery, LargeSize\}$$

Por el contrario, la siguiente configuración C_x es invalida. La configuración incluye la características *TouchScreen* y una batería *SmallSize*. Es una configuración inválida ya que *TouchScreen* requiere una batería *LargeSize*.

$$C_x = \{CellularPhone, LCD, TouchScreen, InputDevice, Stylus, Battery, SmallSize\}$$

C. Análisis de Modelos y Configuraciones de Características

Los Modelos de Características pueden ser analizados automáticamente usando técnicas para resolver problemas con restricciones [7]. Existen propuestas para determinar los errores en un modelo así como determinar la validez de una configuración usando programación basada en restricciones (CSP) [8][9][10], análisis de satisfabilidad (SAT) [11][12] y herramientas de optimización combinatoria y programación lineal [13].

Básicamente, estas propuestas toman los elementos del modelo de características y lo traducen a un problema analizable usando estas herramientas.

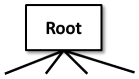
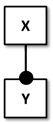
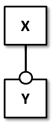
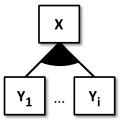
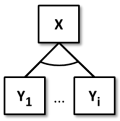
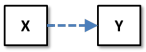
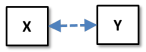
Programación basadas en Restricciones (CSP): Un problema de programación basada en restricciones es un problema definido en términos de variables, posibles valores para esas variables (i.e. dominios de las variables) y restricciones. Una solución para el problema comprende un conjunto, con un valor para cada variable, que satisface las restricciones.

Las herramientas solucionadoras, conocidas como *solvers*, son herramientas que permiten encontrar soluciones a problemas de programación basada en restricciones. Básicamente estas herramientas se centran en determinar la factibilidad del problema (*feasibility* en inglés), i.e. en encontrar una solución factible.

Análisis usando CSP: Es posible analizar un Modelo de Características convirtiendo los elementos del modelo a un problema de programación basada en restricciones o un problema de programación lineal:

- cada característica es convertida a una variable;
- el dominio para cada variable es el mismo: {0, 1}, donde 0 es usado para representar características no seleccionadas y 1 representa las características seleccionadas por el usuario; y
- los diferentes tipos de elementos, grupos y relaciones son traducidos a restricciones sobre esas variables.

Tabla II. Traducción del Modelo de Características a un Conjunto de Restricciones

Elemento	Figura	Traducción
Raíz		$r = 1$
Obligatorio		$x = y$
Opcional		$x \geq y$
Grupo *Or*		$x \leq \sum y_i$ (i.e. $x \rightarrow \sum y_i \geq 1$) $x \geq y_i$
Alternativas		$x = \sum y_i$ (i.e. $x \rightarrow \sum y_i = 1$) $x \geq y_i$
Relación	**Figura**	**Descripción**
Requiere		$x \leq y$
Excluye		$x + y \leq 1$

Los elementos del modelo se traducen a restricciones de la siguiente forma:

- **Características Hijas:** siendo X el padre y Y la hija, entonces la restricción equivalente es $X \leq Y$
- **Características Obligatorias:** siendo X el padre y Y la característica obligatoria, la restricción equivalente es $Y = X$
- **Características Opcionales:** siendo X el padre y Y la característica opcional, la restricción equivalente es $X \geq Y$
- **Grupo *Or*:** siendo X el padre de un grupo *Or* y $Y_1, ..., Y_i$ el conjunto de características hija, la restricción equivalente es $X = 1 \leq \sum Y_i$
- **Grupo de Alternativas:** siendo X el padre de un grupo alternativo y $Y_1, ..., Y_i$ el conjunto de características hijas, la restricción equivalente es $X = \sum Y_i$
- **Requiere:** Considerando la relación $X \xrightarrow{requires} Y$, la restricción equivalente es $X \leq Y$
- **Excluye:** Considerando la relación $X \xrightarrow{excludes} Y$, la restricción equivalente es $X + Y \leq 1$

La Tabla II muestra como se traducen los elementos de un modelo de características en ecuaciones que representa las restricciones correspondientes.

Por ejemplo, considere el modelo presentado en la Figura 1, el modelo de características puede ser traducido al siguiente conjunto de restricciones:

$$
\begin{aligned}
&CellularPhone = 1\\
&CellularPhone = LCD\\
&LCD \leq (TouchScreen + Normal)\\
&TouchScreen \geq LCD\\
&Normal \geq LCD\\
&CellularPhone = InputDevice\\
&InputDevice \leq (KeyPad + Stylus)\\
&KeyPad \geq InputDevice\\
&Stylus \geq InputDevice\\
&ExternalMemory \geq CellularPhone\\
&CellularPhone = Battery\\
&Battery \leq (Small + Large)\\
&Small \geq Battery\\
&Large \geq Battery
\end{aligned}
$$

Usando un *solver*, es posible determinar si el problema tiene solución. Dado un problema P, decimos que $feasible(P)$ retorna $TRUE$ si el problema tiene solución o $FALSE$ si no lo tiene.

Es posible usar esta funcionalidad $feasible(P)$ para determinar si una configuración suministrada por el usuario es válida. Básicamente, se puede (1) convertir el modelo de características a un conjunto de variables y restricciones, (2) asignar uno a cada característica seleccionada en la configuración, y (3) asignar cero a cada característica no seleccionada. Si el problema resultante tiene solución, la configuración es válida. El algoritmo 1 muestra el proceso para validar una configuración.

Algoritmo 1 Validando una configuración

```
1:  procedimiento ValidaConfiguracion(c, fm)
2:      P = encode(fm)
3:      para todo f_s ∈ c hacer
4:          agregarRestriccion(P, f_s = 1)
5:      fin para
6:      para todo f_r ∈ (fm \ c) hacer
7:          agregarRestriccion(P, f_r = 0)
8:      fin para
9:      retornar feasible(P)
10: fin procedimiento
```

D. Configuración involucrando varios usuarios

En algunos casos, los procesos de configuración implican la participación de varios usuarios, cada uno de ellos decidiendo sobre algunas o la totalidad de las características del producto.

Los procesos de configuración involucrando varios usuarios se pueden clasificar de acuerdo a la forma como manejan las decisiones de los diferentes usuarios [14][1]:

Configuración por pasos (*Staged Configuration*, en inglés): Cuando las decisiones de todos los usuarios se consolidan en una sola. En estos procesos, las decisiones tomadas primero por alguno de los usuarios limitan las decisiones que pueden tomar los otros usuarios después [15]. Básicamente, funciona igual que los procesos de configuración interactiva con un solo usuario: Luego de tomar alguna decisión, el software propaga la decisión habilitando y deshabilitando otras características del producto de acuerdo a las restricciones en el sistema.

Configuración usando Vistas: Cuando cada usuario toma decisiones sobre un subconjunto del producto. A cada usuario se le asigna una vista de la configuración que contiene solo las características de son de su interés [3]. En este proceso, la configuración se realiza sobre "vistas"de un mismo modelo. Las decisiones que toma un usuario en una vista pueden afectar las decisiones que toman los otros usuarios en otras vistas. En este proceso de configuración, las decisiones se manejan igual que en la configuración por pasos. La única diferencia es que cada usuario solo puede ver un subconjunto de las características del producto.

Configuración Simultánea: Cuando cada uno de los usuarios toma sus decisiones de forma independiente. En este tipo de configuración, las decisiones tomadas por un usuario no afectan las decisiones con los demás. Después que todos los usuarios hayan terminado su propia configuración, un proceso automatizado adicional consolida las decisiones y busca resolver los conflictos que puedan existir entre las decisiones de los usuarios.

III. Conflictos en Configuraciones Simultáneas

En los procesos de configuración con varios usuarios es posible que las decisiones tomadas por un usuario entren en conflicto con las decisiones tomadas por otro.

Por ejemplo, es posible que un usuario decida incluir una característica mientras otro haya decidido no hacerlo. Considere un proceso donde dos usuarios están configurando un teléfono de acuerdo al modelo de características de la Figura 1. Suponga que el primero de los usuarios indica que quiere incluir una *ExternalMemory* mientras el otro usuario no lo incluye. No es posible contar con un mismo producto que satisfaga las dos configuraciones.

También puede ocurrir que un usuario decida incluir características que no se pueden seleccionar al mismo tiempo que las seleccionadas por alguien más. Por ejemplo, considerando el mismo proceso para configurar un teléfono, puede ser que uno de los usuarios decida incluir una pantalla *TouchScreen* mientras el otro usuario haya decidido usar una *Normal*.

En ambos casos, es necesario revisar las decisions, detectar conflictos y proponer soluciones.

IV. Propuesta para el Manejo de Configuraciones Simultaneas

Nuestra propuesta se centra en proponer automáticamente soluciones que puedan resuelvan los conflictos.

A. Estrategias de Solución

Hemos identificado tres estrategias para proponer estas soluciones:

Maximizar las selecciones: Buscar la solución que incluya la mayor cantidad de características seleccionadas por todos los usuarios. En este caso, si un usuario selecciona *ExternalMemory* mientras otro usuario no lo hace, el sistema va a proponer incluir esa característica.

Minimizar las selecciones: Buscar la solución que incluya la menor cantidad de características seleccionadas por todos los usuarios. En este caso, si un usuario selecciona *ExternalMemory* mientras otro usuario no lo hace, el sistema va a proponer no incluirla.

Priorizar las decisiones de algunos usuarios: Si se asigna un *ranking* o un peso a las decisiones de cada usuario, la estrategia busca la solución que satsifaga en mayor medida a los usuarios con un peso mayor. En este caso, si un usuario con mayor peso selecciona *ExternalMemory* mientras otros usuarios no lo hacen, el sistema va a proponer incluir esa opción.

B. Proceso automatizado

Nuestra solución usa programación lineal para encontrar las soluciones a los conflictos.

Programación Lineal (LP): Un problema de *programación lineal* busca determinar una solución óptima a un problema de restricciones de acuerdo a algún criterio. En estos problemas, además de las variables, dominios y restricciones, es necesario definir una función objetivo. En los problemas de programación lineal, las restricciones y la función objetivo se definen en términos de ecuaciones lineales.

Implementación de las estrategias: Para implementar cada una de las estrategias de solución, es posible definir un problema de programación lineal usando las variables y las restricciones tal como se presentaron en la sección II. Solo hace falta definir la función objetivo en cada caso.

Considere un proceso de configuración en donde participan n usuarios. Cada uno de los usuarios provee una configuración C_i, donde $i \in \{1 \dots n\}$. El conjunto de características sin conflicto $\mathcal{F}_{nc}$ corresponde con las características seleccionadas al tiempo por todos los usuarios, i.e. $\mathcal{F}_{nc} = \bigcap_{i \in \{1 \dots n\}} C_i$. El conjunto de características en conflicto $\mathcal{F}_c$ son las restantes, i.e. siendo $\mathcal{F}_{fm}$ todas las características del modelo, $C_c = \mathcal{F}_{fm} \setminus C_{nc} = \{f \in \mathcal{F}_{fm} \mid f \notin \mathcal{F}_{fm}\}$.

El subconjunto de configuración en conflicto de un usuario CC_i corresponde a las características en la configuración C_i que no están en las características sin conflicto, i.e. $CC_i = C_i \setminus \mathcal{F}_{nc}$.

Maximizar las selecciones: Para buscar la solución con la mayor cantidad de características seleccionadas por todos los usuarios, la función objetivo busca maximizar la inclusión de los subconjuntos de configuración en conflicto. La función objetivo es:

$$\textit{maximize}: \quad \sum(1 * f),\ \forall f \in \bigcup_{i \in \{1 \dots n\}} CC_i$$

Minimizar las selecciones: En este caso, la función objetivo busca minimizar la inclusión de los subconjuntos de configuración en conflicto. La función objetivo es:

$$\textit{minimize}: \quad \sum(1 * f),\ \forall f \in \bigcup_{i \in \{1 \dots n\}} CC_i$$

Priorizar las decisiones de algunos usuarios: En este caso se busca maximizar ponderadamente las decisiones. Si tomamos r_i como el peso asignado a las decisiones del usuario i, la función objetivo es:

$$\textit{maximize}: \quad \sum_{i \in \{1 \dots n\}}(r_i * f_i),\ \forall f_i \in CC_i$$

V. Diseño e Implementación de la Solución

Multi-SPLOT es la implementación de nuestra propuesta. Es una aplicación web basada en Javascript, Angular y Firebase, herramientas que permiten poner a funcionar el software sin contar con infraestructura propia, o con muy bajo costo. Con el uso de estas tecnologías se busca poder publicar el software en internet y lograr que otras personas puedan usarla y darnos retroalimentación.

A. Descripción General

Nuestra aplicación se compone de dos partes:

1. **Una aplicación web para la configuración de productos** que permite (1) a usuarios administradores definir proyectos de configuración y asignar tareas a usuarios configuradores, y (2) a usuarios configuradores seleccionar opciones de los modelos de features.
2. **Un servicio para el análisis de las configuraciones** que permite (1) determinar la válidez de una configuración realizada por un usuario, (2) determinar si existen conflictos entre varias configuraciones individuales, y (3) proponer configuraciones que permitan soluciones conflictos entre varias configuraciones individuales.

La Figura 2 muestra una descripción general de la aplicación.

La *aplicación web para la configuración de productos* fué desarrollada usando Javascript y el framework Google Angular[3]. Ella se encuentra organizada en módulos, controladores, vistas y servicios, siguiendo las recomendaciones del framework. La aplicación almacena la información en una base de datos Google Firebase[4] y ejecuta operaciones de análisis sobre las configuraciones usando un servicio web desarrollado por nosotros.

El *servicio web para anlizar configuraciones* está desarrollado usando Google App Script[5], una plataforma de desarrollo

[3]https://angularjs.org/

[4]https://firebase.google.com/

[5]https://developers.google.com/apps-script/

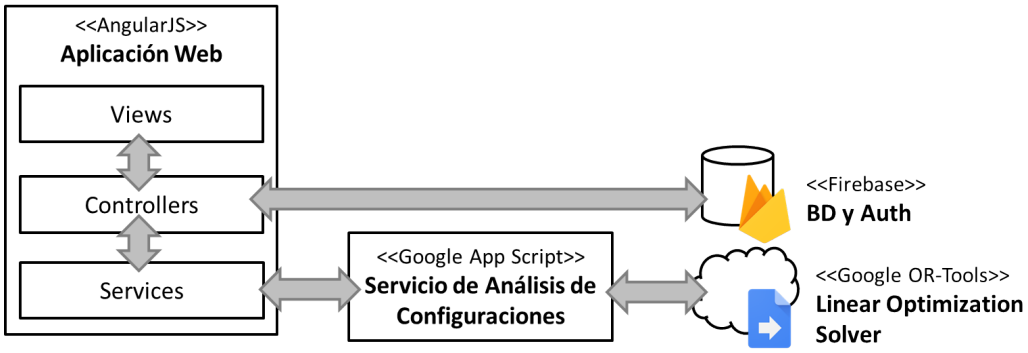


Figura 2. Esquema de la Arquitectura de la Aplicación

en la nube basada en Javascript, y la libreria incluida de Google Optimization Tools[6].

Las siguientes secciones muestran parte del diseño de la solución. En particular, se presentan: (1) el esquema de datos utilizado para representar los proyectos de configuración, los modelos de características y las configuraciones, (2) el diseño para la interfaz de usuario, y (3) la implementación del servicio en Google App Script.

B. Esquema de datos

Una de las decisiones de diseño se centró en la representación de los modelos de características y las configuraciones. El desarrollo del proyecto requirió la creación de objetos en Javascript que pudieran ser intercambiados entre la aplicación y el servicio web como mensajes REST y que pudiesen ser almacenados en la base de datos Firebase.

1) Información de los Proyectos y los Usuarios: La figura 3 muestra un diagrama de clases mostrando la estructura utilizada para almacenar los datos de los Proyectos y los Usuarios.

Project almacena la información de los proyectos, incluyendo un identificador y un nombre. Cada proyecto mantiene información del conjunto de líderes y miembros del proyecto, el modelo de características y las configuraciones hechas por los usuarios sobre ese modelo.

User mantiene la información de los usuarios. Incluye información del nombre, el correo y la imagen del usuario. Además, incluye un indicador que establece si el usuario esta activo o no. Usando este indicador, en lugar de eliminar los datos del usuario, la aplicación coloca el valor de activo en falso.

Feature Model mantiene información del modelo de características incluido en el proyecto. Todos los usuarios en el mismo proyecto establecen configuraciones para el mismo modelo.

Configuration contiene información de todas las configuraciones ingresadas por los diferentes usuarios del proyecto.

Los datos son almacenados en Firebase usando una estructura "aplanada"(i.e. *flatten structure*). Es decir, las relaciones entre las clases no son almacenadas como colecciones de objetos. En lugar de ellos, usamos listas de identificadores. Esta es la práctica recomendada en Firebase[7].

[6]https://developers.google.com/optimization/

[7]https://firebase.google.com/docs/database/web/structure-data

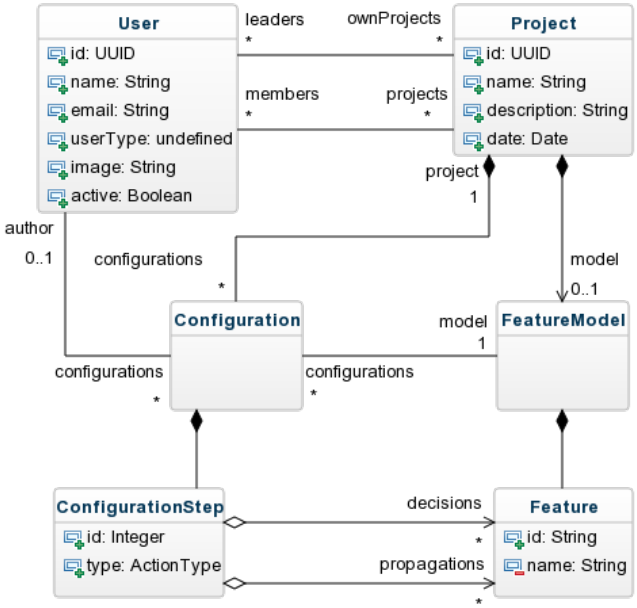


Figura 3. Diagrama de clases del Proyecto de Configuración

2) Información de los Modelos de Características: La información de los Modelos de Características se almacena de forma similar a la de los proyectos. Por cada Modelo se almacenan las diferentes características indicando su tipo y sus características hijas. Por ejemplo, considerando el modelo mostrado en la Figura 1, se guarda la información de la característica *CellularPhone* con un tipo *Root* e indicando que tiene como características hijas a *LCD*, *InputDevice*, *ExternalMemory* y *Battery*. Para el caso de *LCD*, el tipo es Mandatory y las características hijas son *Touch Screen* y *Normal*.

3) Información de las Configuraciones: Las configuraciones son listados de las características seleccionadas durante el proces de configuración. Nótese que cada vez que el usuario decide incluir una característica, la decisión es propagada automáticamente. Es posible que otras características sean incluidas o excluidas automáticamente. Cada configuración incluye no solo un listado de las características señaladas por el usuario, sino también las seleccionadas automáticamente.

(a) Pantalla de Inicio de Sesión

(b) Pantalla de Bienvenida

(c) Pantalla de Administración

(d) Pantalla de Administración de Proyectos

(e) Pantalla de Configuración

Figura 4. Imágenes de las Pantallas de Multi-SPLOT

C. *Interfaz de Usuario*

Multi-SPLOT se diseñó como una aplicación web de una sola página. La aplicación completa se carga desde el primer acceso al sitio web. Cada una de las pantallas está creada como una vista que se muestra o no de acuerdo a las opciones que selecciona el usuario en el menú.

Ya en funcionamiento, el usuario puede ingresar a la aplicación utilizando su correo electrónico o por medio de una cuenta de *Github*[8]. Antes de entrar por primera vez, la cuenta debe haber sido incluida en la lista de pre-registro por algun usuario con el rol de *Administrador*. El usuario debe registrarse agregando información adicional e indicando si va a ingresar usando su correo o usando Github. La Figura 4a muestra la pantalla de inicio de sesión.

Los usuarios tiene uno de tres posibles roles: Administrador, Líder de Proyecto y Configurador.

Los usuarios administradores tiene la posibilidad de administrar los usuarios del sistema. Por ejemplo, ellos pueden pre-registrar los usuarios, modificarlos o removerlos del sistema. La Figura 4c muestra la pantalla de administración de usuarios.

Los usuarios líderes del proyecto pueden crear proyectos y definir los usuarios que pueden participar. La Figura 4d muestra una pantalla de administración de proyectos. Allí pueden verse los usuarios que participan de un proyecto y la cantidad de configuraciones individuales que se tienen en un momento determinado.

Los usuarios configuradores pueden seleccionar opciones para uno o más proyectos. La Figura 4e muestra una pantalla de configuración. Allí el usuario puede ver las opciones de configuración y seleccionar las que desea para un producto determinado.

D. *Servicio de Análisis de Configuraciones*

Cada vez que un usuario realiza la configuración del producto o cuando se desean detectar conflictos entre configuraciones realizadas por varios usuarios, la aplicación web usa un servicio web desarrollado en Google App Script para analizar las configuraciones.

Google App Script es una plataforma web que permite la creación de aplicaciones y servicios web. Estas aplicaciones funcionan en la infraestructura de Google y tienen restricciones sobre las librerías que pueden usarse y la forma como se puede acceder a ellas. Para poder construir el servicio web se hizo necesario usar una librería especial que usa un API JSON para acceder a los datos en Firebase y una librería de Google que ofrece acceso limitado a las funciones de optimización de Google Optimization Tools.

E. *Implementación*

El software se desarrolló usando herramientas como Eclipse y Jetbrains WebStorm. Adicionalmente, se utilizó el esquema de control de versiones de Github. El código fuente se halla disponible en:

`https://github.com/SebasG22/SPLOT`.

[8]http://www.github.com

VI. Conclusiones

Este artículo presenta nuestra propuesta para el manejo de configuraciones simultáneas. Por un lado, ofrece estrategias automatizadas para proponer soluciones cuando la configuración proveída por un usuario genera conflicto con las configuraciones proveídas por otros. Por otro lado, ofrece una aplicación web que permite administrar usuarios, proyectos y configuraciones individuales.

En cuanto a la solución a conflictos para configuraciones simultáneas, este artículo muestra tres estrategias propuestas. Básicamente, estas estrategias buscan encontrar una configuración que mantengan todas las selecciones comunes y (1) maximicen el número de las otras selecciones individuales de los usuarios, (2) mantengan al mínimo el número de las otras selecciones, o (3) asignen una prioridad a las decisiones de algunos usuarios. Todas las estrategias se implementan usando nuestra propia librería para analizar configuraciones de modelos de características usando programación lineal [16].

Sobre el software, por un lado, se creó una aplicación web usando Javascript y Angular para implementar la interfaz de usuario y la funcionalidad. Esta aplicación usa Firebase para manejar los servicios de autenticación y almacenamiento a través de Computación en la Nube. Por otro lado, se creó un servicio web para el análisis de las configuraciones usando Google App Script y Google Optimization Tools. Estas tecnologías nos permiten desplegar la aplicación usando infraestructura gratuita o a muy bajo costo. Esperamos que esto potencie el uso de la aplicación en el futuro.

En la actualidad el software soporta modelos de características sin atributos. Es decir, no es posible definir valores para atributos como precio, peso o tamaño a las características del modelo. Esto impide que se puedan incluir restricciones más complejas en el modelo. Para el futuro estamos planeando extender la aplicación y soportar modelos de características con atributos e interfaces de usuario para la configuración que manejen estos atributos. Igualmente, queremos realizar casos de estudio con casos reales y explorar si las estrategias definidas para resolver conflictos son apropiadas o es necesario buscar otras nuevas.

Agradecimientos

Las librerías para procesar configuraciones se basan en un trabajo previo de Jaime Chavarriaga desarrollado en un proyecto conjunto de la Universidad de los Andes y Siemens Colombia.

Referencias

[1] A. Hubaux, T. T. Tun, and P. Heymans, "Separation of Concerns in Feature Diagram Languages: A Systematic Survey," *ACM Computing Surveys*, pp. 1–23, 2013.

[2] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, *Knowledge-based Configuration: From Research to Business Cases*. Morgan Kaufmann, 2014.

[3] A. Hubaux, P. Heymans, P.-Y. Schobbens, D. Deridder, and E. Abbasi, "Supporting Multiple Perspectives in Feature-based Configuration," *Software and Systems Modeling (SoSyM)*, pp. 1–23, 2011.

[4] A. Hubaux, "Feature-based configuration: Collaborative, dependable, and controlled," Ph.D. dissertation, University of Namur, Belgium, 2012.

[5] D. Benavides, A. Felfernig, J. A. Galindo, and F. Reinfrank, "Automated analysis in feature modelling and product configuration," in *13th International Conference on Software Reuse (ICSR 2013)*, 2013, pp. 160–175.

[6] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Software Engineering Institute, Carnegie-Mellon University, Tech. Rep., November 1990.

[7] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: a literature review," *Information Systems*, vol. 35, no. 6, pp. 615 – 636, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.is.2010.01.001

[8] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, "Using Java CSP Solvers in the Automated Analyses of Feature Models," in *Generative and Transformational Techniques in Software Engineering (GTTSE 2005)*. Springer Berlin Heidelberg, 2006, pp. 399–408.

[9] A. S. Karataş, H. Oğuztüzün, and A. Doğru, "Mapping extended feature models to constraint logic programming over finite domains," in *14th International Conference on Software Product Lines (SPLC'10)*. Springer-Verlag, 2010.

[10] R. Mazo, C. Salinesi, D. Diaz, and A. Lora-Michiels, "Transforming Attribute and Clone-Enabled Feature Models Into Constraint Programs Over Finite Domains," in *6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'11)*. Springer-Verlag, 2011.

[11] M. Mendonca, A. Wasowski, and K. Czarnecki, "SAT-based Analysis of Feature Models is Easy," in *13th International Software Product Line Conference (SPLC '09)*, 2009, pp. 231–240.

[12] M. Janota, "SAT Solving in Interactive Configuration," Ph.D. dissertation, University College of Dublin, 2010.

[13] P. van den Broek, "Optimization of product instantiation using integer programming," in *14th International Software Product Line Conference (SPLC 2011)*, 2011, pp. 107–111.

[14] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration through specialization and multilevel configuration of feature models," *Software Process: Improvement and Practice*, vol. 10, no. 2, pp. 143–169, 2005.

[15] ——, "Staged Configuration Using Feature Models," in *Software Product Lines*, ser. Lecture Notes in Computer Science, R. L. Nord, Ed. Springer Berlin Heidelberg, 2004, vol. 3154, pp. 266–283.

[16] J. C. Navarro and J. Chavarriaga, "Using Microsoft Solver Foundation to analyse Feature Models and Configurations," in *2016 8th Euro American Conference on Telematics and Information Systems (EATIS)*, 2016, pp. 1–8.