Implementar un módulo de YaST para la configuración del teclado del sistema

Trabajo fin de grado

Alumno: Joaquín Yeray Martín de la Nuez

Titulación: Grado en Ingeniería Informática

Tutor: José Miguel Santos Espino

Las Palmas de Gran Canaria, enero de 2019

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE

D/Dª Joaquín Yeray Martín de la Nuez, autor del Trabajo de Fin de Título Implementar un módulo de YaST para la configuración del teclado del sistema, correspondiente a la titulación Graduado en Ingeniería Informática,
en colaboración con la empresa/proyecto (indicar en su caso) _____

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 13 de diciembre de 2018.

El estudiante

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[✓] Positivamente

[ ] Negativamente
(la justificación en caso de informe negativo deberá incluirse en el TFT05)

Fdo.: _____

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Índice

1 Introducción....5
1.1 Qué es OpenSUSE....5
1.2 Qué es el software libre....5
1.3 Qué es YaST....7
1.4 Apoyo del equipo de YaST a este proyecto....9
2 Definición del proyecto....10
2.1 Objetivo de producto....10
2.2 Objetivos académicos, competencias....12
2.3 Costes económicos y planificación....14
3 Requisitos del software....16
3.1 Funcionalidad del módulo....16
3.2 Puesta en producción en openSUSE....16
3.3 Solución orientada a objetos....17
3.4 Otros requisitos....18
4 Desarrollo del módulo....20
4.1 Sobre systemd y cómo gestionar el teclado del sistema....20
4.2 Metodología de trabajo....22
4.3 Requisitos de los proyectos de YaST....26
5 Diseño e implementación....35
5.1 Uso del patrón Strategy para disminuir el acoplamiento....35
5.2 Interfaz gráfica de usuario....40
5.3 Cliente de YaST....46
5.4 Pasos seguidos para llegar a la implementación actual....46
6 Resultado y trabajos futuros....57
6.1 Resultado actual....57
6.2 Próximos pasos: paso a producción del módulo....57
6.3 Trabajos futuros....58
7 Conclusiones....59
7.1 Aportaciones a la comunidad....59

7.2 Aportaciones a mi desarrollo profesional.....59
8 Glosario de términos.....61
9 Índice de Figuras.....62
10 Referencias.....64

1 Introducción

Este proyecto parte de la propuesta de un colaborador de la comunidad de **openSUSE**, consiste en implementar desde cero un módulo de YaST para la configuración del teclado del teclado del sistema, que sustituya al actual [1].

Para poder explicar el origen de esta propuesta y el objetivo del proyecto en profundidad es necesario aclarar su contexto tecnológico y social.

1.1 Qué es OpenSUSE

openSUSE es un sistema operativo de software libre basado en GNU/Linux. Este proyecto posee una gran comunidad de personas involucradas en él. Comúnmente estos sistemas operativos son denominados *distribuciones* de GNU/Linux. Como su propio nombre indica procede de GNU y del uso de Linux. GNU es un proyecto colaborativo que tiene como objetivo crear un sistema operativo libre, es decir usando solamente software libre. Por su parte, Linux es un proyecto de software libre cuyo objetivo es construir un núcleo de un sistema operativo tipo Unix. Existen una gran variedad y diversidad de distribuciones GNU/Linux. Entre ellas, **openSUSE** destaca como una de las más reconocidas al lado de otras como Ubuntu, Fedora o Arch Linux.

openSUSE se trata de un proyecto mantenido por la comunidad abierta de **openSUSE**. Esto quiere decir que cualquier interesado puede participar en el desarrollo de este proyecto. Y no solamente desarrollando código: es posible participar de diversas maneras, ya sea traduciendo términos y documentación, realizando proyectos de diseño gráfico y muchas otras contribuciones.

1.2 Qué es el software libre

Para ser un poco más concretos, se dice que un proyecto software es software libre, cuando además de publicar el código fuente, posee una licencia que cumple con las cuatro libertades fundamentales del software libre [2]. Estas libertades fueron enunciadas por Richard Matthew Stallman (fundador del proyecto GNU y el movimiento software libre) en 1986. Las cuatro libertades son:

- *“La libertad de ejecutar el programa, con cualquier propósito (libertad 0)”.*

- *“La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.”*

- *“La libertad para redistribuir copias para que se pueda ayudar a tu vecino (libertad 2)”.*

- *“La libertad de mejorar el programa y publicar sus mejoras al público, de manera que beneficie a toda la comunidad (libertad 3). El acceso al código fuente es una condición previa para esto.”*

Estas cuatro libertades buscan garantizar el derecho de ejecutar, estudiar, modificar y distribuir el software por cualquier usuario del mismo. De estas cuatro libertades nacen las licencias de software libre. Una licencia que no respete y ampare todas estas libertades no puede ser considerada como una licencia de software libre.

En este proyecto se hará uso de la licencia denominada “GNU General Public License v2.0” o “GPL-2.0” [3].

Es muy frecuente la confusión entre los conceptos de software libre y de código abierto u *“open source”*. Y es que toda licencia de software libre es a su vez de código abierto. Pero aunque ambos tipos de software se basan en la misma premisa básica, que el código es público y cualquiera puede acceder a él, la diferencia radica en el cumplimiento de las cuatro libertades que posee el usuario sobre el código. Desde que alguna de estas libertades sea restringida o no sea contemplada en la definición de la licencia, dicha licencia no puede ser considerada como una licencia de software libre. Por lo tanto, comprobar si la licencia de software cumple con estas cuatro libertades es la mejor manera de discriminar si se trata de un proyecto de software libre.

Que **openSUSE** se trate de un proyecto de software libre no le impide tener el apoyo de algunas empresas, como puede ser el caso de SUSE. Dicha empresa es uno de los mayores promotores de este proyecto, aportando soluciones empresariales al proyecto con su versión SUSE Linux Enterprise Server. SUSE comercializa esta versión de **openSUSE**, algo que puede hacer gracias a la licencia que posee el software. La manera en la que obtiene rentabilidad no es obteniendo

un beneficio directo a cambio del código, sino ofreciendo un soporte y mantenimiento de los servidores que llevan instalada dicha distribución.

1.3 Qué es YaST

YaST acrónimo de inglés “**Y**et **a**nother **S**etup **T**ool” que se podría traducir como “otra herramienta de configuración”. YaST es denominada la herramienta de instalación y configuración de la distribución **openSUSE** en sus diferentes versiones incluyendo SUSE Linux Enterprise. Esta herramienta facilita al usuario la instalación del sistema operativo y además, una vez instalado el sistema, permite de manera muy fácil y sencilla la gestión y configuración del sistema operativo. Es por ello que se trata de una herramienta bastante conocida en el ámbito de las distribuciones GNU/Linux, llegando a ser muchas veces el punto de diferenciación y valor añadido clave que aporta **openSUSE** frente a otras de las más famosas distribuciones. Y es que YaST permite realizar configuraciones relacionadas con el hardware, las configuraciones de red, administrar los usuarios del sistema, servicios del sistema y seguridad, entre otras muchas, de una manera realmente fácil y sencilla, al alcance de la mayoría de los usuarios.

Yo mismo he podido comprobar cómo usuario de openSUSE que YaST facilita en gran medida la instalación y configuración del sistema operativo. Desde mi punto de vista, es en la gestión de la configuración del sistema donde más destaca esta herramienta. Pocas son las distribuciones que pueden presumir de disponer de una aplicación como YaST y para un usuario que no ha estado en contacto con distribuciones GNU/Linux es de gran ayuda disponer de una herramienta gráfica de fácil uso para administrar el sistema, debido a que hace más llevadera la curva de aprendizaje inicial que pudiera surgir.

Cuando un usuario final ejecuta la aplicación de YaST se encuentra con un menú principal en el cual se encuentra una larga lista de opciones, como se puede ver en la Figura 1.

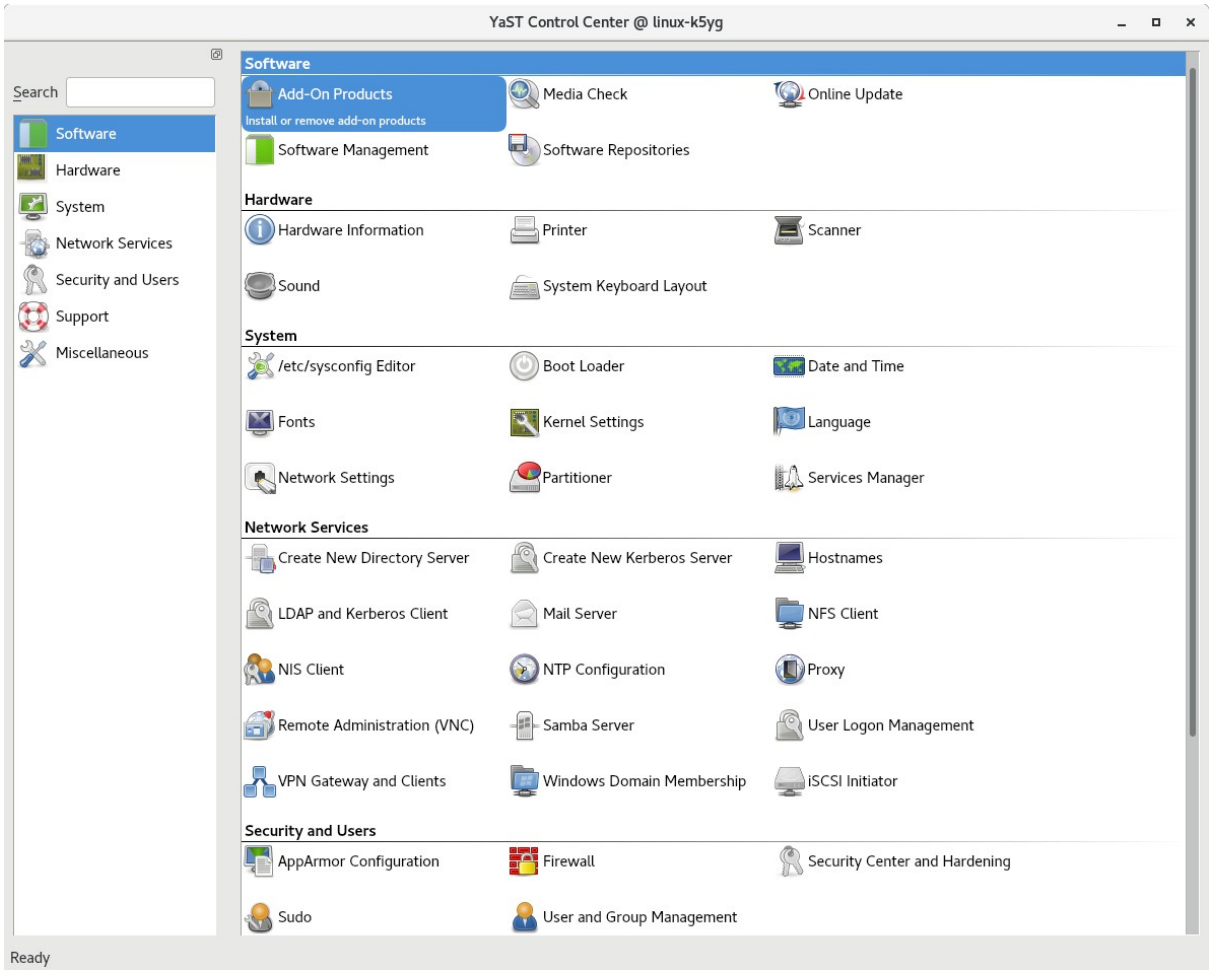


Figura 1: Menú de YaST.

Internamente, a nivel de arquitectura, YaST no es una aplicación que se pueda considerar un monolito, sino que se compone de de diferentes módulos. Cada uno de ellos aporta soluciones a necesidades diversas. Existe un menú principal de aplicación cuya función es ejecutar el módulo concreto que el usuario ha seleccionado. En la siguiente lista se muestran algunos de los módulos que posee YaST:

- **yast-firewall**: configuración y gestión del cortafuegos
- **yast-storage**: configuración y gestión del almacenamiento
- **yast-country**: configuración y gestión del idioma y localización
- **yast-bootloader**: configuración y gestión del arranque del sistema
- **yast-users**: configuración y gestión de los usuarios del sistema

A su vez, a nivel de desarrollo existe un conjunto de utilidades comunes que facilita en gran medida el desarrollo de un módulo de YaST. Un ejemplo es el paquete denominado “yast ruby bindings” que aporta *bindings* para facilitar el manejo de interfaz gráfica de usuario haciendo uso del lenguaje de programación Ruby. Con ello se consigue facilitar el desarrollo de los módulos. A lo largo de este documento se podrá ver con mayor detalle cómo ha influido esto en el desarrollo de este trabajo.

1.4 Apoyo del equipo de YaST a este proyecto

Algo a destacar en el desarrollo de este trabajo de fin de grado es que YaST ha influido en este proyecto no solo por su tecnología, sino también por el elemento humano, ya que los integrantes del equipo de YaST serán los encargados de realizar las revisiones de código y validar la solución antes de distribuirla en producción. Además han ofrecido ayuda técnica en los casos en los que se ha requerido.

2 Definición del proyecto

2.1 Objetivo de producto

Este proyecto tiene como origen una propuesta del desarrollador de YaST y colaborador con la comunidad de **openSUSE** Ancor González Sosa, publicada en el repositorio público de **openSUSE** llamando “mentoring”, el dos de febrero de 2017. En dicho repositorio se publican diferentes proyectos que pueden ser realizados por cualquier desarrollador que tenga interés. La finalidad es atraer a nuevos desarrolladores y promover entre ellos la colaboración con proyectos de software libre y con el proyecto **openSUSE** en concreto. En el siguiente extracto de la propuesta se puede observar cuál es el principal objetivo de este proyecto:

> “*The goal of this project is to rewrite the keyboard management from scratch in a proper object oriented way. For example, the interaction with systemd should be just one of many possible backends, so the module is ready for the next big change in Linux keyboard management or to be ported to systems without systemd.” [1]*

Como se puede leer, la propuesta se basa en reescribir desde cero un módulo existente de YaST, en concreto el módulo de gestión del teclado del sistema. Este módulo está siendo usado ahora mismo por los usuarios del sistema operativo **openSUSE** en sus diferentes versiones así como en la versión comercial de la compañía SUSE. Una propuesta tan radical de eliminar un código que está ahora mismo en producción y está siendo usado por miles de usuarios, tiene que tener unos argumentos de peso que la apoyen.

Pues bien, esta propuesta surge después de que el sistema operativo **openSUSE** comenzara a hacer uso de “**systemd**”. Concretamente **systemd** es un conjunto de software que proporciona elementos fundamentales para un sistema operativo Linux. Incluye demonios, librerías y utilidades para interactuar con el núcleo del sistema operativo Linux. Entre otras funcionalidades proporciona un “System and

Service Manager” [4] (administrador de sistema y servicio) que se ejecuta como primer proceso al arrancar el sistema y es el encargado de iniciar el resto del sistema. Pero aunque es una de sus funcionalidades principales, **systemd** no solo se encarga del arranque del sistema, sino que se comporta más bien como un framework que facilita la administración del núcleo del sistema. Un demonio que proporciona **systemd** es el denominado “**systemd-localed**” [5], se trata de un servicio que permite gestionar la configuración regional del sistema, así como la configuración del teclado del sistema.

Es debido a este último servicio que la adopción por parte de **openSUSE** de **systemd**, provocó cambios en la manera en que se gestiona el teclado del sistema, dejando obsoleta la antigua implementación del módulo. Por ello se realizó un desarrollo para adaptar el código del módulo de YaST, con el propósito de comunicarse con la API que proporciona el nuevo demonio del sistema. Esta adaptación fue costosa debido a que el código está fuertemente acoplado a la implementación subyacente en el sistema. Además, la actual estructuración del código dificulta en gran medida la introducción de nuevos cambios. Como consecuencia de ello la adaptación con **systemd** tiene mucho margen de mejora a nivel de implementación. Como se puede ver a continuación estos argumentos se explicitan también en la propuesta inicialmente publicada en el repositorio de **openSUSE**:

> “*The current source code is poorly structured. Since the original implementation was highly tight to the underlying tools, the adaptation to systemd was hard and is still very far from perfection.” [1]*

Es por todo esto por lo que se plantea desarrollar un nuevo módulo que sustituya al actual. El objetivo que trasciende bajo esta idea es conseguir un módulo que sea fácilmente actualizable y sobre el que resulte menos costoso realizar nuevos cambios en el futuro. Recordemos que el mantenimiento del código es la fase que más costes suele implicar a lo largo de todo el ciclo de vida de un producto software. En consecuencia, el objetivo final es desarrollar un módulo de YaST para la gestión del teclado del sistema operativo **openSUSE**, con el principal requisito de obtener un coste de mantenimiento menor con respecto al que posee la implementación que

actualmente se encuentra en uso.

El sistema actual conlleva un elevado coste de mantenimiento. Esto se ve reflejado en lo dificultoso que resulta realizar cambios en su implementación. Los principales motivos de este elevado coste son la pobre estructuración del código, la no aplicación de las denominadas buenas prácticas de desarrollo y la falta de aplicación de los principios englobados en el paradigma de la programación orientada a objetos. Esto ha tenido como resultado que el módulo actual no permite cambiar fácilmente su implementación, y por tanto aunque la implementación actual está integrada **systemd**, esta posee un amplio margen de mejora, ya que se encuentra fuertemente acoplado. Para reducir este coste de mantenimiento es crucial disminuir el acoplamiento con la interfaz ofrecida por **systemd** y el sistema operativo, por otro lado también es importante conseguir una mejor organización del código. Obtener un menor acoplamiento permitiría realizar cambios más fácilmente en la implementación e incluso que el sistema tenga la capacidad de disponer de varias implementaciones, es decir que pueda usar una u otra implementación dependiendo de la configuración del sistema y por tanto sea menos costoso añadir otra implementación. Esto último puede darse en caso de que cambie la interfaz que proporciona **systemd** o que este sea reemplazado.

2.2 Objetivos académicos, competencias

A continuación se enumeran las competencias cubiertas por el desarrollo de este proyecto. La principal competencia cubierta es la **CII010**, cuyo texto es el siguiente:

«Conocimiento de las características, funcionalidades y estructura de los Sistemas Operativos y diseñar e Implementar aplicaciones basadas en sus servicios.»

Puedo afirmar que esta competencia queda cubierta tras el desarrollo de este proyecto, ya que ha implicado un profundo aprendizaje sobre el funcionamiento de parte de los sistemas operativos, en concreto la gestión del teclado para sistemas operativos GNU/Linux. Para poder realizar este proyecto han sido realmente útiles los conocimientos adquiridos sobre sistemas operativos durante el grado. A partir de estos ha sido posible implementar una aplicación basada en los servicios que proporciona el sistema operativo **openSUSE**.

- **CII016:** *«Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería de software.»*

La competencia **CII016** se justifica con la aplicación de diversos principios de orientación a objetos que se enumeran más adelante en este documento, así como por la metodología de desarrollo usada para el desarrollo de este proyecto, que se explica más detenidamente en el apartado 4.2 Metodología de trabajo

- **IS01:** *«Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.»*

- **IS07:** *«Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.»*

Cabe destacar tambien las competencias **IS01** e **IS07**, ya que durante el desarrollo del proyecto hubo que identificar y analizar los problemas del módulo ya existente, identificar y priorizar los requisitos de usuario, para posteriormente diseñar e implementar el módulo, aplicando para ello las teorías, principios y prácticas de la ingeniría de software.

- **CII017:** *«Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.»*

La competencia **CII017** se ha visto cubierta debido al análisis e implementación de la interfáz gráfica de usuario que se ha desarrollado.

- **CII06:** *«Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.»*

- **CII07:** *«Conocimiento, diseño y utilización de forma eficiente los tipos y estructuras de datos más adecuados a la resolución de un problema.»*

- **CII08:** *«Capacidad para analizar, diseñar, construir y mantener aplicaciones*

de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.»

El hecho de analizar, diseñar y desarrollar la aplicación con el lenguaje orientado a objetos Ruby, aplicando algoritmos básicos y haciendo uso de los tipos proporcionados por Ruby así cómo algunos tipos propios desarrollados específicamente para este módulos, corrobora la aplicación de las competencias **CII06, CII07 y CII08**.

- **FB04:** *«Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación de la ingeniería.»*

Por último la competencia **FB04** se puede dar por cubierta con el uso de diversas herramientas específicas para el desarrollo de software, algunas de estas son: la terminal del sistema operativo, el editor de código **Visual Studio Code**, **Git** como sistema de control de versiones o **Github** como portal web para alojar el repositorio y realizar las peticiones de cambio.

2.3 Costes económicos y planificación

En tiempo invertido en el desarrollo propiamente dicho de la aplicación ha sido apróximadamente de unas 220 horas, incluyendo análisis del problema, de la aplicación ya existente, búsqueda de información sobre tecnologías implicadas como **systemd**, revisiones del código y su documentación.

Si hablamos de los recursos humanos necesarios, ha sido necesaria mi intervención como desarrollador de la aplicación, así como de los integrantes del equipo de **YaST** a la hora de revisar las peticiones de cambio. Algunos integrantes de **YaST** tambien han intervenido a la hora de resolver algunas cuestiones planteadas en las listas públicas de correo, notése que aunque no ha sido el caso, cualquier integrante de la comunidad openSUSE podría haber intervenido para ayudar en el proceso, ya que se trata de un proyecto de software libre.

Otros recursos necesarios para el desarrollo de este proyecto son:

- La dispocisión de un computador con el sistema operativo **openSUSE**, en su

versión **LEAP 15.0** o **Tumbleweed**.

- Una cuenta del portal web **Github** que nos permite interactuar con el repositorio del proyecto perteneciente a **openSUSE** y la creación de nuestra bifurcación del repositorio. Esta cuenta de **Github** no supone ningún coste económico ya que se tratan de repositorios públicos.
- Un editor de código, en mi caso la eleción ha sido **Visual Studio Code**, que tampoco conlleva ningún coste económico.
- **Git** como sistema de control de versiones.
- **Ruby** como lenguaje de programación.

3 Requisitos del software

3.1 Funcionalidad del módulo

Hay que tener en cuenta que de cara al usuario final el propósito de este software es administrar el teclado del sistema, no la configuración particular de cada uno de los usuarios del sistema operativo. Es decir, no se trata de una solución para que cada usuario del sistema operativo administre su configuración del teclado, sino para que aquel usuario con el rol de administrador del sistema, configure el teclado del sistema operativo. Aunque en un primer lugar pueda parecer la misma funcionalidad, en realidad se trata de problemas distintos. Además, la configuración del teclado por usuario ya posee diversas y exitosas soluciones en entornos GNU/Linux. Según la documentación ofrecida por SUSE [6]:

> «El módulo del teclado del sistema de YaST le permite definir la disposición del teclado por defecto del sistema (también usada para la consola)».

Esto quiere decir que la distribución del teclado configurada mediante el módulo de YaST será la que usará el sistema en casos en los que el usuario no haya iniciado sesión, en la terminal del sistema. YaST no cubre la necesidad de los usuarios del sistema de especificar qué distribución de teclado desean usar en vez de la definida por defecto en el sistema. Es aquí donde entran en juego otras soluciones software como por ejemplo las proporcionadas por los entornos de escritorio KDE y Gnome, que permiten a cada uno de los usuarios establecer que distribución de teclado usar en el entorno gráfico.

3.2 Puesta en producción en openSUSE

Una vez desarrollado el programa, el siguiente paso ha de ser su puesta en producción. Esto se debe realizar distribuyéndolo junto con el sistema operativo **openSUSE** en su versión denominada ‘Tumbleweed’. Para ello debe de ser desarrollado siguiendo las guías de desarrollo de YaST, es decir ha de cumplir con

ciertas convenciones establecidas por el equipo de YaST y la comunidad para poder ponerlo en producción, estas convenciones son principalmente la guía de estilo de YaST [7] y la convención para organizar la estructura de ficheros [8].

Para que el módulo pueda llegar a todos los usuarios de **openSUSE**, primero será publicado en los repositorios de software *YaST:Head* [9] y *openSUSE:Factory* [10]. Estos repositorios de software contienen aplicaciones software en sus versiones más recientes que deben ser probadas antes de ser publicadas en una versión de **openSUSE**, el objetivo es que el módulo sea probado por una cantidad reducida de usuarios. Una vez se haya probado entonces se publicará en la siguiete versión de **openSUSE Tumbleweed**.

3.3 Solución orientada a objetos

Para lograr un menor acoplamiento a **systemd** y favorecer la mantenibilidad del módulo, se propone hacer uso del paradigma de programación orientada a objetos y las buenas prácticas que se prescriben dentro de él. En el paradigma de la programación orientada a objetos existe una serie de principios y patrones de diseño que buscan mejorar la calidad del código, repercutiendo así en una mejor mantenibilidad. Algunos de estos principios son los principios **SOLID**, ampliamente conocidos en parte gracias a la divulgación realizada por Robert C. Martin [11]. Los principios que forman **SOLID** son los enunciados a continuación:

- **S**ingle Responsibility Principle
- **O**pen Close Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Otros principios relacionados con la programación orientada a objetos son «Don't Repeat Yourself (DRY)» [12] o la «Law of Demeter» [13]. En cuanto a patrones de diseño también hay una larga lista. En el libro "Design Patterns: Elements of Reusable Object-Oriented Software" [14] escrito por Erich Gamma, Richard Helm,

Ralph Johnson y John Vlissides están documentados varios de estos patrones, tales como:

- Adapter
- Strategy
- Observer
- Factory method
- Command
- Singleton

Estos principios y patrones de diseño son de gran ayuda ya que por un lado los principios de diseño ayudan al desarrollador con unas pautas de organización del código que la experiencia de la industria ha demostrado que contribuyen a producir un código más mantenible.

3.4 Otros requisitos

Aparte de la descripción del objetivo, en la propuesta de **openSUSE** también se recogen algunos de los requisitos técnicos para el módulo:

> "*The module must conform to the YaST development guidelines … it must include unit tests (RSpec)"*

Concretamente se recoge especificamente en la propuesta inicial del proyecto [1] que el código desarrollado debe de llevar sus propios tests unitarios. Esto es así debido a la gran ventaja que proporcionan este tipo de tests, y es que proporcionan seguridad a los desarrolladores a la hora de realizar cambios en el código, pues son una manera rápida de probar el comportamiento del software. Por otro lado sirve de documentación del código pues la propia descripción de los tests aclara que es lo que debe hacer cada parte del código. De hecho se trata de una documentación actualizada, ya que el hecho de que un cambio en el código que hace que un test falle, obliga al desarrollador a adaptar el test, manteniendose actualizado. Esto se

explica con más detenimiento en el apartado denóminado Tests.

4 Desarrollo del módulo

Durante el desarrollo de este trabajo ha sido necesario invertir tiempo en investigar cómo funciona, a nivel interno, la gestión del teclado del sistema en entornos GNU/Linux. Para ello ha sido necesario investigar sobre los siguientes puntos:

- Conocer detalladamente la implementación actual del módulo
- Conocer qué es **systemd**
- Cómo gestionar la configuración del teclado con **systemd**
- Gestionar la configuración del teclado para entornos gráficos.

Hubo una pequeña primera parte del proyecto que fue de investigación para conocer el contexto. Pero por lo general es una tarea que se ha ido repitiendo durante el desarrollo de las distintas funcionalidades.

Tanto el flujo de trabajo como las herramientas que se han utilizado durante la realización del trabajo se han visto influenciadas por el contexto y la naturaleza del proyecto. Más concretamente ha sido el hecho de ser un proyecto de software libre y pertenecer a YaST lo que más ha influido.

4.1 Sobre systemd y cómo gestionar el teclado del sistema

Como se ha explicado con anterioridad, **systemd** es un software que incluye un conjunto de demonios, librerías y utilidades que permite interactuar con el núcleo del sistema operativo, Linux.

De entre este conjunto de utilidades la que afecta a este proyecto es el demonio **systemd-localed.** Como se puede leer en la página de ayuda de este servicio [5], se trata de un servicio del sistema que permite cambiar la configuración regional del sistema, así como la distribución de teclado tanto para la consola como para el entorno gráfico.

Este servicio provee de unos comandos de consola para hacer uso de él. En la siguiente lista se enumeran los comandos necesarios para el desarrollo de este

proyecto.

- "*localectl list-keymaps*"
- *"localectl status"*
- *"localectl set-keymap"*

El primer comando nos permite obtener un listado de todas las distribuciones de teclado disponibles en el sistema. En la Figura 2 se puede ver un extracto de la salida de consola que produce este comando.

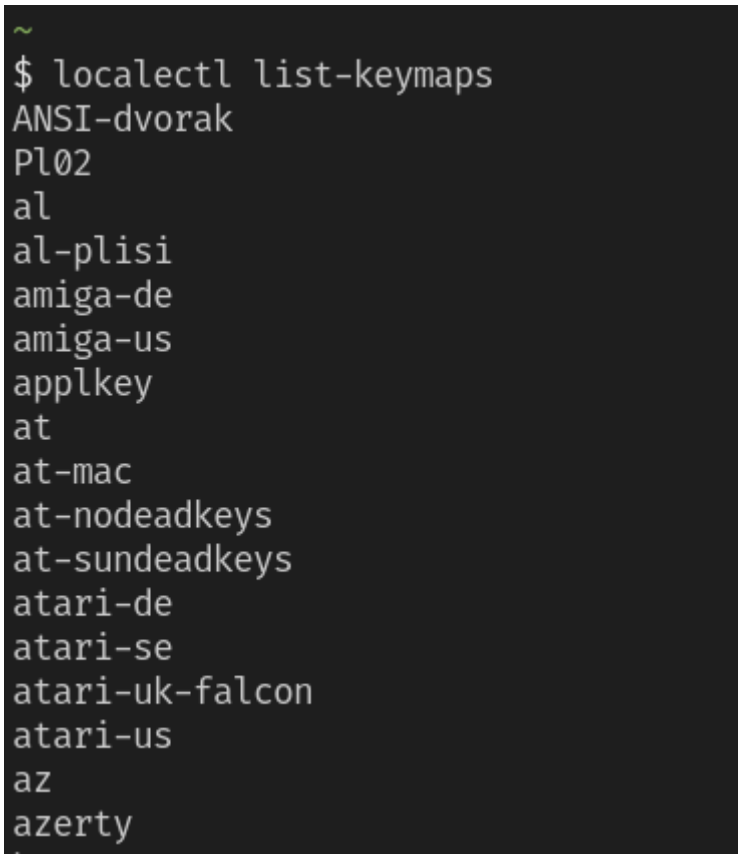


Figura 2: Parte del resultado obtenido al ejecutar el comando "localectl list-keymaps".

Por otro lado *"localectl status"* aporta información de la configuración actual del sistema, podemos ver el resultado de ejecutar este comando en la Figura 3.

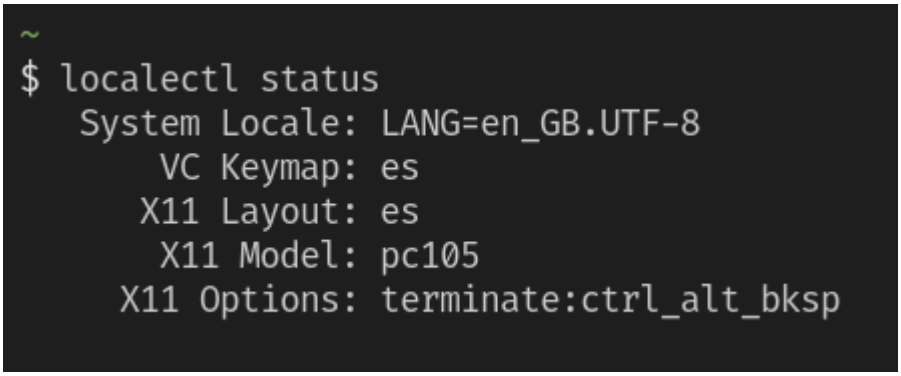


Figura 3: Información obtenida al ejecutar el comando "localectl status".

De este comando han resultado especialmente útiles los valores "*VC Keymap*" y "*X11 Layout*". El primero hace referencia a la distribución actual para la consola y el segundo para el entorno gráfico. Nótese que aunque en el ejemplo de la Figura 3 el valor de dichas propiedades coincida, puede ser diferente en cada una.

Por último se ha hecho del comando "*localectl set-keymap*" para cambiar la distribución actual del teclado. Este comando recibe como parámetro el código de la distribución de teclado que se quiere establecer, como por ejemplo '*es*', '*qwerty*' o '*azerty*'.

4.2 Metodología de trabajo

Para realizar el proyecto la primera acción fue ponerme en contacto con los desarrolladores de YaST que habían publicado la propuesta del proyecto en el repositorio *mentoring* de **openSUSE** [15]. Este repositorio está destinado a publicar posibles proyectos en los que cualquier interesado en introducirse en la comunidad **openSUSE** puede entrar y solicitar realizar alguno de los diferentes proyectos publicados. Aquí podemos ver una diferencia con el flujo habitual que se explica en el apartado 4.2.1 "Cómo contribuir a un proyecto de software libre": la *issue* no estaba publicada en el propio repositorio del proyecto yast-country [16]. Para ponerme en contacto con los desarrolladores de YaST he usado canal de irc "#YaST" en "freenode.net" [17].

En primer lugar me puse de acuerdo con el equipo de YaST que me encargaría del desarrollo de este proyecto. A continuación envié un correo electrónico a la lista de correos pública yast-devel [18], con el objetivo de dar a conocer esta decisión a todo

el equipo de YaST. El contenido del mensaje [19] se puede ver en la Figura 4.

[yast-devel] Starting a project: re-implement YaST keyboard

From: Joaquín Martín <joaquinmnsm@xxxxxxxxx>
Date: Thu, 28 Sep 2017 22:16:12 +0100
Message-id: <CAKVp-318k07VSQrAj6FSSGnjLbkGMF6CoQ9TZbZkqs3FGL7nFw@mail.gmail.com>

```
Hi, my name is Joaquin, maybe you remember me from 2016 Google Summer of Code
:)

In the lasts weeks I have been talking with Ancor, to do this project
(https://github.com/openSUSE/mentoring/issues/79)
The project is basically rewrite the YaST keyboard management module.
I want to start the project now so I had talked in the IRC channel
about how deal with project, and the best approach seems to have a
branch in yast-country repository for this project and make the pull
requests to this branch (called keyboard_rewrite) and when the project
is finished merge this branch into master.

So sometimes you will see me in the IRC channel and i will ask you
questions or by a review for pull request :P

If something is not clear, have any suggestion or anything else,
please don't hesitate to contact me.
That's all,
regards
--
To unsubscribe, e-mail: yast-devel+unsubscribe@xxxxxxxxxxxx
To contact the owner, e-mail: yast-devel+owner@xxxxxxxxxxxx
```

Figura 4: Mensaje enviado para comunicar el comienzo del proyecto.

Al tratarse de un trabajo colaborativo con un proyecto de software libre, el método de desarrollo empleado ha sido muy parecido a el método que tradicionalmente siguen los proyectos de software libre [20] . A este método típico de proyectos de software libre se le ha realizado algunas adaptaciones, unas influenciadas por el equipo de YaST [21] y otras debido a las condiciones y el contexto del proyecto.

4.2.1 Cómo contribuir a un proyecto de software libre

Los proyectos de código abierto o de software libre suelen estar alojados de manera

pública en portales como **Github** o **GitLab.** En adelante se nombrará Github para referirse a estos servicios, ya que gran parte de los proyectos de código abierto y de software libre se alojan en sus servidores, y en concreto este proyecto, así como YaST y **openSUSE**.

Típicamente el flujo estándar para contribuir a cualquier proyecto es el siguiente:

1. Comúnmente en los proyectos de este tipo existen lo que se llaman "*issues*" o asuntos. Los tipos de *issues* pueden variar entre los diferentes proyectos debido a su distinta organización, pero suelen predominar ciertos tipos de *issues*. Algunos de estos tipos son:
    - errores de la aplicación que hay que solucionar
    - nuevas funcionalidades que se quieren implementar
    - mejoras a realizar en el código
2. La *issue* es asignada a un contribuidor, el cual se encargará principalmente del desarrollo de la misma. Es común que se genere un hilo de debate en la propia *issue* si hay varias personas implicadas, ya sea para aclarar cómo abordar la funcionalidad, el origen de un error, aclarar dudas, etc. Es en este punto donde un nuevo contribuidor puede aportar al desarrollo del proyecto, postulando para abordar el desarrollo de la *issue*.
3. Una vez asignado un contribuidor, este con su usuario de github, realiza lo que se denomina un *fork* o bifurcación. Entiéndase por "*fork*" a una copia íntegra del repositorio en la cuenta del contribuidor, duplicando de esta manera el repositorio.
4. El contribuidor trabajará en su copia del repositorio para realizar los cambios pertinentes. De esta manera el proyecto no se verá influenciado por los cambios realizados por el contribuidor durante esta fase.
5. Una vez el contribuidor considera que ha desarrollado la funcionalidad o corregido el error, realizará lo que se denomina un "*pull reques*t" o propuesta de cambio, al repositorio oficial del proyecto. Un *pull request* es un mecanismo para la revisión de código en el que se ve muy fácilmente cuales son los cambios realizados por el contribuidor. Una vez se crea el *pull request*, otro contribuidor será el encargado de revisar los cambios, con el objetivo de encontrar mejoras, comprobar que cumple con unos mínimos de

calidad exigidos, comprobar la documentación etc. Este sistema permite que todos los cambios introducidos en el proyecto sean al menos revisados por dos personas. A partir de esta revisión es posible se genere un debate a partir de la revisión. En caso de que sea necesario realizar alguna modificación en la implementación antes de aceptar el *pull request*, se vuelve a repetir el paso 4 explicado anteriormente. Una vez se ha realizado dicha mejora, el *pull request* es actualizado y vuelve a ser revisado, creando así un flujo de retroalimentación.

6. Una vez se ha llegado al consenso de que los cambios han sido los adecuados y el desarrollo requerido para finalizar la *issue* se ha realizado, el código está listo para ir a producción, para ello se realiza lo que se denomina un "*merge*". Esto es aceptar el *pull request*, que no es más que llevar esos cambios a la rama de producción para que sean distribuirlos en la siguiente versión del software. Con ésto quedaría el *pull request* y la issue, como cerrado o solucionado.

4.2.2 Cambios en la metodología para contribuir al proyecto

Debido a que el proyecto no consiste simplemente en añadir una nueva funcionalidad, sino que se trata de un proyecto con una mayor envergadura y con una duración más larga de lo habitual, se han aplicado algunos cambios en el flujo estándar de un proyecto de software libre para adecuarlo a dichos condicionantes.

El gran cambio que se ha realizado en el flujo típico para colaborar con un proyecto de software libre es que se creó una nueva "rama" (*branch*) en el repositorio antes de realizar su bifurcación (*fork*), con la idea de trabajar en esta rama sin interferir en el código principal.

Esta rama se ha llamado "*keyboard_rewrite*" y fue creada con la idea de realizar la fusión (*merge)* de los *pull request* en esta rama y no en la rama de producción denominada "master". Al crear un flujo de desarrollo paralelo al de producción, se consigue que este proyecto no afecte a los posibles desarrollos que se hagan en el código de producción. Una vez el desarrollo se haya completado en esta rama paralela, se fusionará con la rama "master" con el objetivo de poner definitivamente el producto en producción.

Otra ventaja de tener una rama paralela a la rama de producción es que es posible

realizar *pull request* con un menor número de cambios en el código. Así se consigue que los cambios sean más fáciles de revisar por parte de los integrantes del equipo de YaST y a su vez facilita llevar un flujo de desarrollo iterativo e incremental.

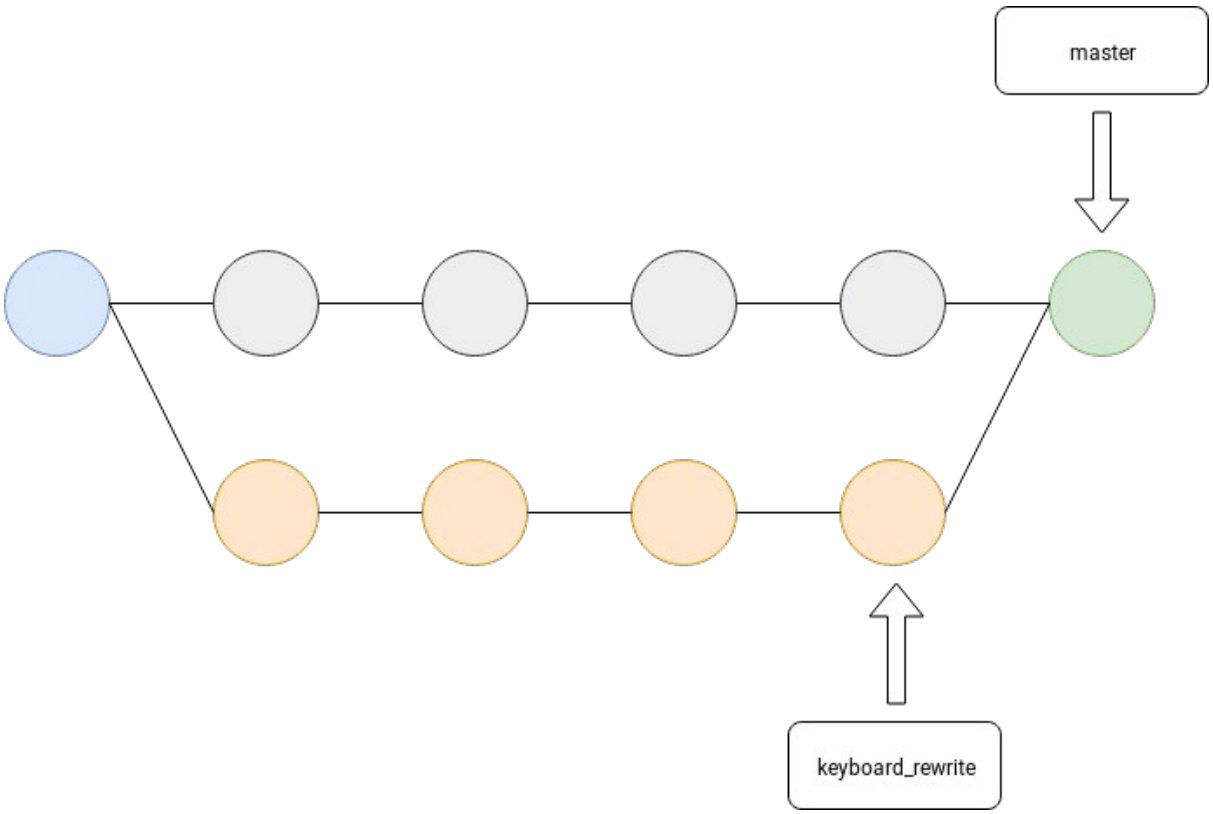


Figura 5: Diagrama ideal de flujo de las ramas.

La Figura 5 muestra el diagrama simplificado del flujo de las ramas, donde cada círculo representa un “commit” en el repositorio.

El *commit* de color azul es el último realizado en la rama *master* antes de comenzar el proyecto. Los de color naranja representan los distintos *commits* en la rama “*keyboard_rewrite*”. Los de color gris son los *commits* que se han realizado en la rama *master* paralelamente al desarrollo del proyecto. Por último el de color verde representa el *commit* en el cual se habrá realizado la fusión de las dos ramas, signo de que el proyecto ha finalizado.

4.3 Requisitos de los proyectos de YaST

Todo producto de YaST tiene como requisito disponer de una batería de tests que

avalen el correcto funcionamiento del sistema, cumplir las indicaciones de la guía de estilo de código de **Ruby** [7] y respetar la estructura organizativa del código [8] definidas por el equipo. Durante el desarrollo de este trabajo se ha tenido que tener en cuenta estos puntos.

4.3.1 Tests

Durante el desarrollo se ha hecho uso de la metodología llamada «*desarrollo dirigido por pruebas»,* más conocida por su nombre en inglés: *Test Driven Development* (TDD). TDD fue ideado por Kent Beck en los años noventa como parte de *Extreme Programming* [22].

Esta metodología se basa en desarrollar el test previamente a la implementación de la funcionalidad. Aparte de facilitar cumplir el requisito de tener las funcionalidades con sus correspondientes tests, nos ha permitido generar un diseño emergente, con tests que cubren las funcionalidades de la aplicación y que además sirven a modo de documentación para otros desarrolladores.

Para la implementación de los tests se ha hecho uso del framework RSpec [23].

Figura 6: Logo RSpec.

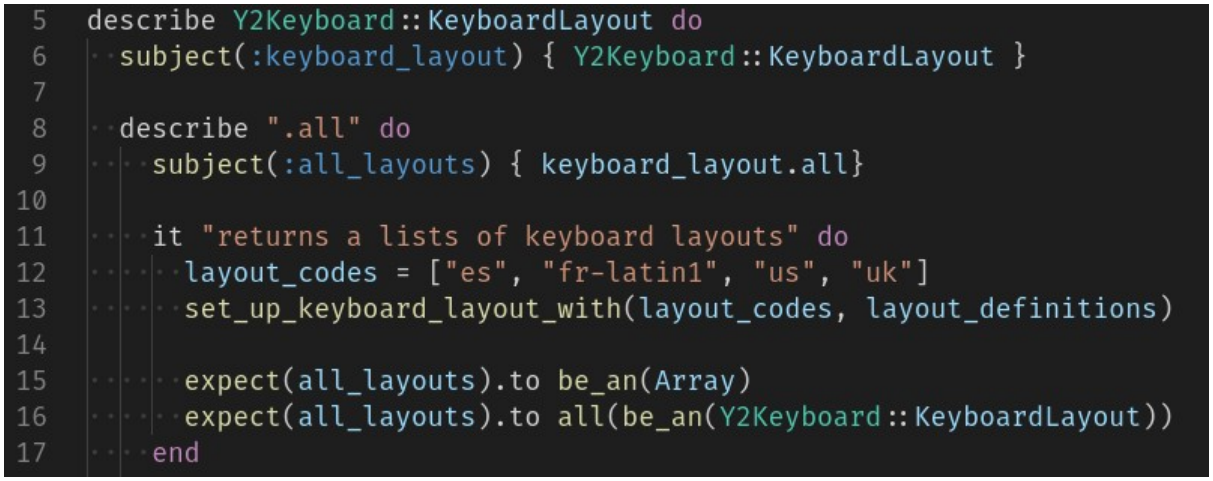
El uso de este framework viene dado por dos motivos:

- Se trata de una de los framework de tests más famosos para Ruby.
- Es usado por YaST en el resto de proyectos.

Con el objetivo de conseguir que los test sean fácilmente entendibles por otro desarrollador y facilitar su mantenimiento, se ha seguido la guía recomendada por el

equipo de YaST, denominada Better Specs [24].En esta guía se define un conjunto de buenas prácticas a seguir a la hora de escribir tests usando **RSpec**, con unos ejemplos bastante claros, así como prácticas comunes que no están consideradas como buenas prácticas, todo ello con su correspondiente explicación.

Como muestra del método, en la siguiente imagen se puede observar un test implementado en el proyecto. En él se está comprobando que al realizar una llamada al método *all* de la clase *KeyboardLayout* este retornará un *Array* que contiene objetos de tipo *KeyboardLayout.*



```
5  describe Y2Keyboard::KeyboardLayout do
6    subject(:keyboard_layout) { Y2Keyboard::KeyboardLayout }
7
8    describe ".all" do
9      subject(:all_layouts) { keyboard_layout.all}
10
11     it "returns a lists of keyboard layouts" do
12       layout_codes = ["es", "fr-latin1", "us", "uk"]
13       set_up_keyboard_layout_with(layout_codes, layout_definitions)
14
15       expect(all_layouts).to be_an(Array)
16       expect(all_layouts).to all(be_an(Y2Keyboard::KeyboardLayout))
17     end
```

Figura 7: Implementación de un test haciendo uso de RSpec.

En caso de que las comprobaciones que se ejecutan en el test no se cumplan tendremos una salida como la siguiente:

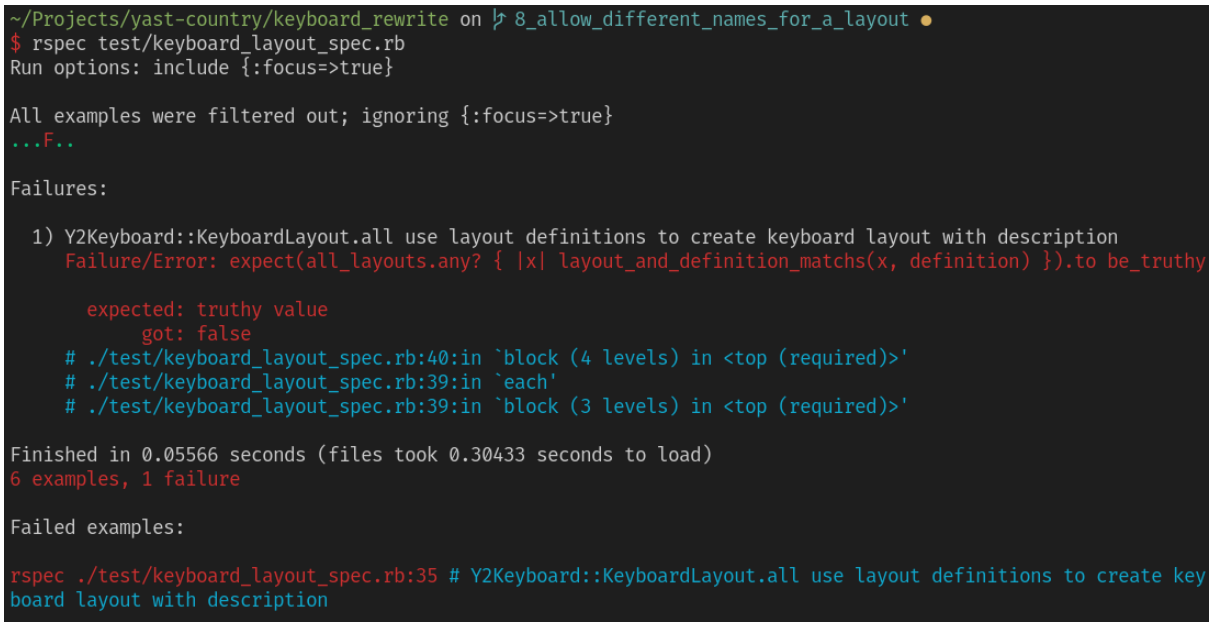


Figura 8: Resultado de ejecutar un test que no tiene una correcta implementación.

En el caso en que sí se cumplan dichas condiciones tendremos una salida similar a la que se puede observar en la Figura 9:

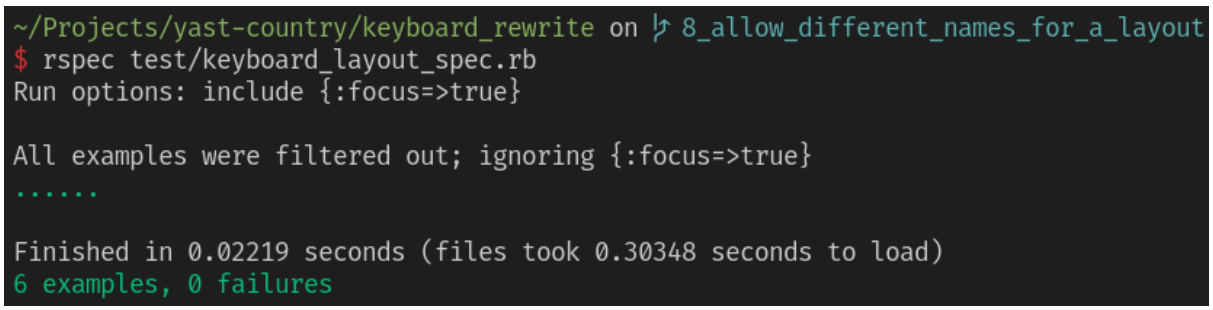


Figura 9: Resultado de ejecutar un test con una correcta implementación.

Los tests tienen otra gran ventaja, y es que sirven como documentación dirigida a futuros colaboradores del proyecto. Esto es algo muy importante en un proyecto de software libre, ya que en este tipo de proyectos una buena documentación facilita que otros desarrolladores puedan aportar al proyecto. En la Figura 10 se puede observar un ejemplo de cómo he utilizado los nombres de los tests a modo de documentación.

```
$ rspec --format doc test/systemd_strategy_spec.rb
Run options: include {:focus=>true}

All examples were filtered out; ignoring {:focus=>true}

Y2Keyboard::Strategies::SystemdStrategy
  #all
    returns a lists of keyboard layouts
    initialize the layout description
    does not return layouts that does not have description
  #codes
    returns a lists of available layout codes
  #apply_layout
    changes the keyboard layout
  #load_layout
    in X server
      changes the current keyboard layout used in xorg
      do not try to change the current keyboard layout in console
    in text mode
      do not try to change the current keyboard layout in xorg
      changes the current keyboard layout in console
    using ncurses inside X server
      when setting current keyboard layout in console
        do not raise error
        log error information
  #current_layout
    returns the current used keyboard layout

Finished in 0.0282 seconds (files took 0.30781 seconds to load)
12 examples, 0 failures
```

Figura 10: Resultado obtenido al ejecutar los tests con el formato de documentación.

En este proyecto no ha sido necesario realizar otro tipo de test más allá de los unitarios, como pueden ser tests de integración o tests punto a punto (“end to end”) debido a que la naturaleza del problema y la solución escogida no lo ha requerido.

4.3.2 Convenciones de código

Durante el desarrollo del módulo ha sido necesario cumplir con las convenciones establecidas por el equipo de YaST, en concreto con la convención para la organización de los ficheros [8] de los módulos de YaST así como con la guía de

estilo de Ruby [7] definida por SUSE.

La convención de organización del código ha afectado al proyecto en los siguientes puntos:

- Nombre de los directorios
- Organización de los ficheros en los distintos directorios
- Nombre de los módulos de Ruby definidos
- Creación de un cliente para que el módulo pueda ser ejecutado por YaST
- Ubicación de los ficheros de tests

En la guía de estilo de **Ruby** [7] se define una larga lista de convenciones que se deben respetar a la hora de desarrollar un proyecto **openSUSE** haciendo uso del lenguaje Ruby. Para facilitar esta tarea y automatizarla en la medida de lo posible, se hecho uso de la herramienta RuboCop [25].

Figura 11: Logotipo de RuboCop.

RuboCop se trata de una herramienta de software libre distribuido bajo la licencia MIT. En el fichero README.md [26] del repositorio público de **RuboCop** podemos leer:

«RuboCop es una analizador de código estático Ruby y formateador de código.»

Fuente: [26]

RuboCop se trata de una herramienta famosa en proyectos que hacen uso del lenguaje Ruby, que se trata de un lenguaje interpretado que proporciona al

desarrollador diversas formas de escribir el código. Debido a ello comenzaron a surgir las convenciones de código de Ruby, y **RuboCop** es la herramienta que se utiliza habitualmente para comprobar que se cumplen dichas convenciones. Podemos ver un ejemplo en la Figura 12.

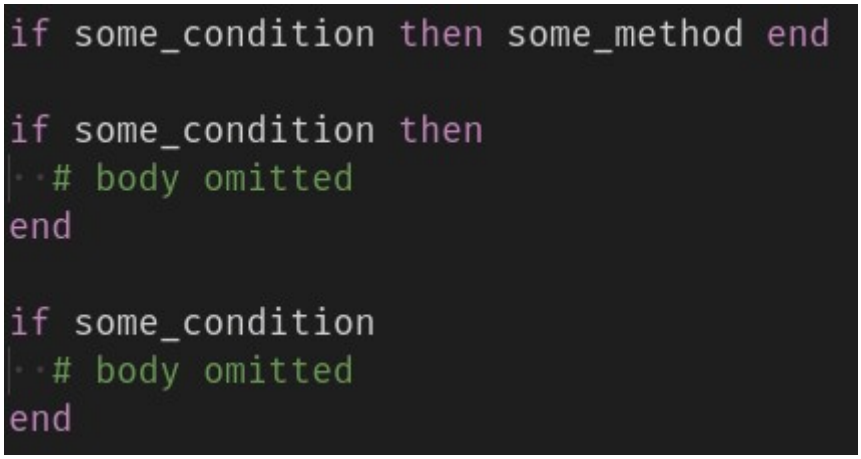


Figura 12: Diversos ejemplos de escribir un if en Ruby.

En la Figura 12 se muestran tres de las muchas formas en las que se puede escribir una condición "**if**" en Ruby. Esta flexibilidad del lenguaje permite al desarrollador buscar la mejor forma de expresar el código con la posibilidad de mejorar la legibilidad del código.

Pero seguir fielmente una convención de código en un lenguaje como Ruby sin ninguna herramienta que ayude al desarrollador a cumplir las diversas reglas es un trabajo arduo. Es ahí donde **RuboCop** aporta valor, ya que definiendo las reglas de la convención de código en un fichero **YAML** y posteriormente ejecutando **RuboCop** mediante la consola, podremos ver en qué parte del código se ha infringido alguna regla de la convención. Incluso proporciona la funcionalidad de arreglar automáticamente algunos de los tipos de las reglas que se hayan infringido.

En la Figura 13 podemos ver un ejemplo de la salida que produce **RuboCop** al ejecutarlo en el proyecto.

Figura 13: Ejemplo salida de RuboCop.

Como se puede observar en este caso la herramienta no ha encontrado ningún punto en el código que no esté cumpliendo con la convención de código.

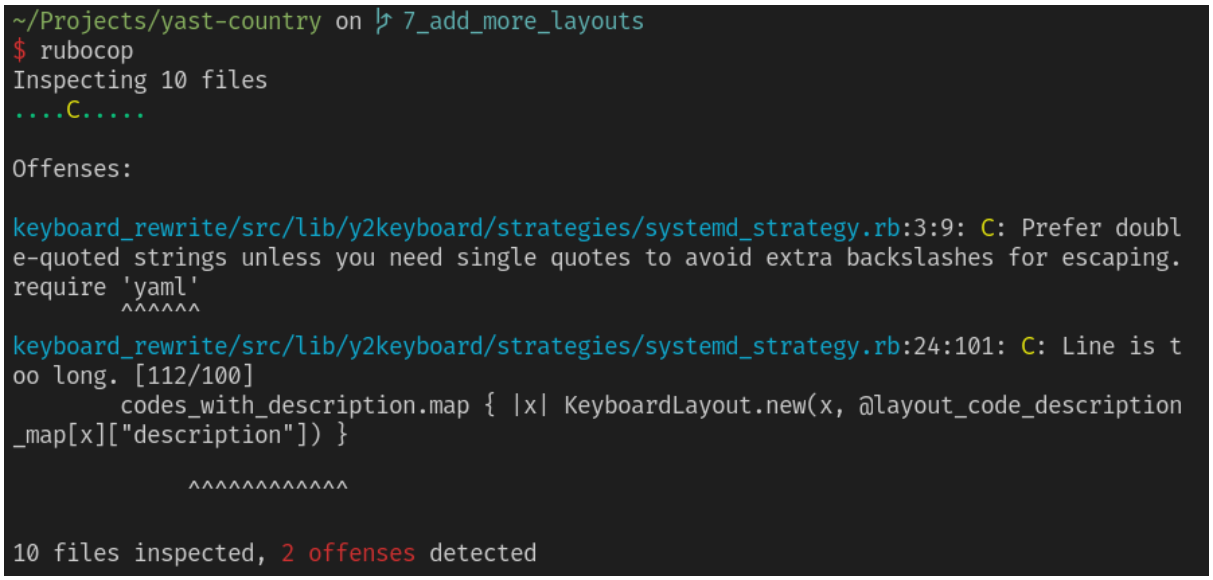


Figura 14: Ejemplo salida de RuboCop con "offenses".

En la Figura 14 podemos ver un ejemplo de la salida por consola al ejecutar **RuboCop** cuando ha encontrado que alguna parte del código no cumple con la convención. En este caso se trata de el uso de comillas simples en vez de dobles, y de una línea que supera el límite de caracteres.

Para poder usar **RuboCop** ha sido necesario establecer una configuración para que las comprobaciones que realice se ajusten a la convención. Para ello se ha creado el fichero ".rubocop.yml". Podemos observar el contenido de este fichero en la Figura 15.

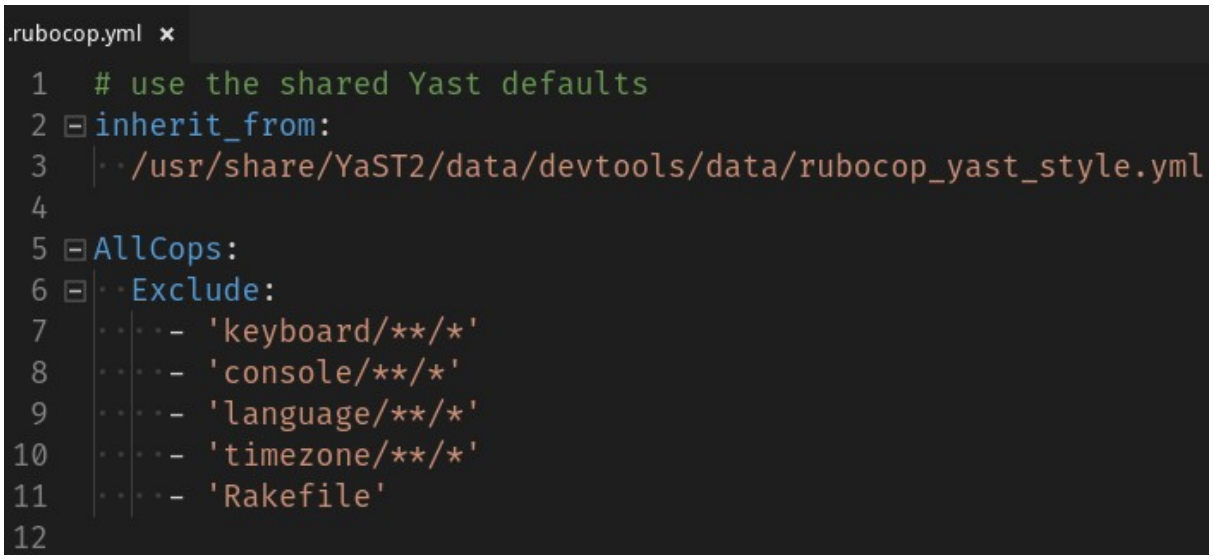


Figura 15: Fichero .rubocop.yml, configuración de RuboCop.

En la Figura 15 vemos que la configuración se trata de un fichero de tipo **YAML**.

En la segunda y tercera línea indicamos que herede la misma configuración que se encuentra en la ruta indicada en la tercera línea. Esa configuración se obtiene al instalar el paquete "*yast2-devtools*", un paquete que proporciona herramientas básicas para desarrollar módulos de YaST. El contenido de dicho fichero se puede ver en el repositorio del paquete [27].

Posteriormente se hace uso de la sección "Exclude" para indicarle a RuboCop que se abstenga de inspeccionar todos los ficheros que no estén incluidos en la carpeta de nuestro proyecto. Esto se ha hecho ya que el código existente en el resto de proyectos de este repositorio no cumple con la actual guía de estilos definida, y por tanto esta guía solo aplica al nuevo módulo desarrollado.

5 Diseño e implementación

En este apartado se exponen los aspectos más relevantes de la implementación realizada para cumplir con los requisitos del proyecto. Estos son el uso de principios y patrones de diseño, la implementación de la interfáz gráfica de usuario y la implementación del cliente de YaST. Por último se describen tambien los pasos que se han seguido para llegar a la implementación actual.

5.1 Uso del patrón Strategy para disminuir el acoplamiento

Con el objetivo de disminuir el acoplamiento del código a la implementación de **systemd** se ha utilizado el patrón de diseño **strategy**. En el libro “Design Patterns Elements of Reusable Object-Oriented Software” de los autores Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides este patrón está clasificado como un patrón de comportamiento. En dicho libro se enuncia además el propósito de este patrón:

«Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.» [14]

Design Patterns Elements of Reusable Object-Oriented Software

En otras palabras, la idea de este patrón de diseño es tener distintas implementaciones que cumplan un mismo contrato, de esta forma una clase que puede usar indistintamente cualquiera de las implementaciones siempre que cumpla con el con el contrato definido. (Nota: entiéndase por contrato a la interfaz pública de las implementaciones).

El diagrama **UML** de este patrón en la implementación actual queda tal y como se muestra en la Figura 16.

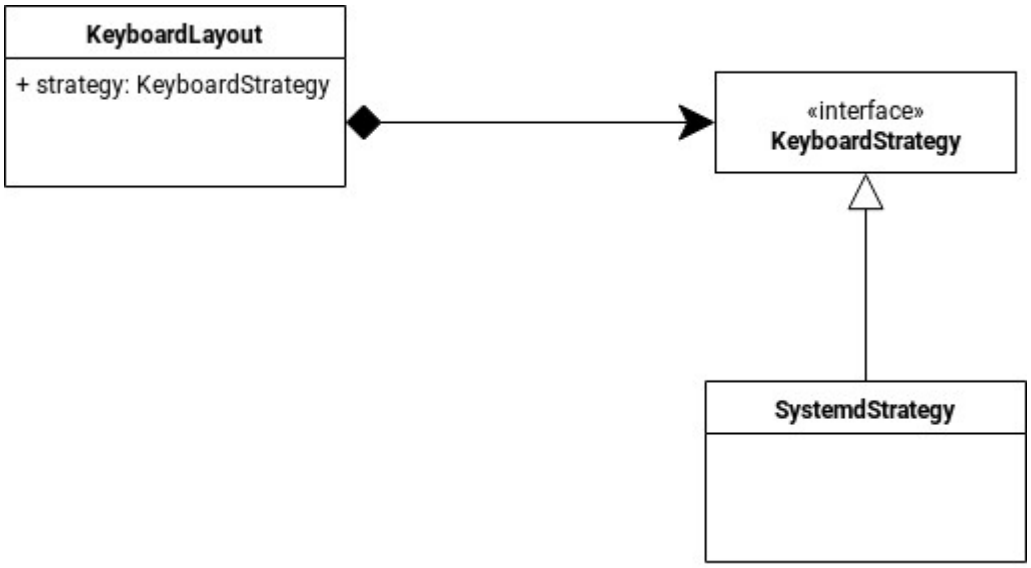


Figura 16: Diagrama UML de la implementación actual.

El concepto “interfaz” como contrato se ha tenido en cuenta a la hora de desarrollar la aplicación, pero al contrario de otros lenguajes como Java o C#, Ruby no proporciona una ninguna forma de explicitar dicha interfaz. Esto es debido a la naturaleza del lenguaje Ruby, que se trata de un lenguaje interpretado y de tipado dinámico en el que el uso del denominado “*duck typing*” [28] hace innecesario la explicitación formal de la interfaz. Para entender qué es “duck typing” está bastante extendida la siguiente frase:

> *“Cuando veo un ave que camina como un pato, nada como un pato y suena como un pato, a esa ave yo la llamo un pato”.*

Con “*duck typing*” se entiende que las variables en Ruby no tienen tipo, simplemente son objetos. Esto es así en Ruby porque incluso es posible añadir un nuevo método a una sola instancia de un objeto, entonces esa instancia del objeto no cumplirá con los métodos definidos en la clase. Lo que la frase anteriormente citada trata de explicar es que lo realmente importante es el comportamiento que tenga el objeto y no el tipo de objeto, dicho de otra forma, lo importante son los métodos que se pueden invocar.

Debido a esto, el contrato que han de cumplir las implementaciones concretas como la de **systemd** está definido implícitamente con los métodos públicos de las implementaciones.

Aplicar este patrón aunque en la actualidad solo se dispone de una implementación aporta una ventaja: en caso de querer usar otra implementación que no sea de **systemd,** solamente hay que crear una nueva clase con la nueva implementación que cumpla la misma interfaz que la implementación actual. Todo esto es posible hacerlo sin afectar la implementación actual de **systemd**. En la Figura 17 se muestra el diagrama UML en el caso de añadir una nueva implementación.

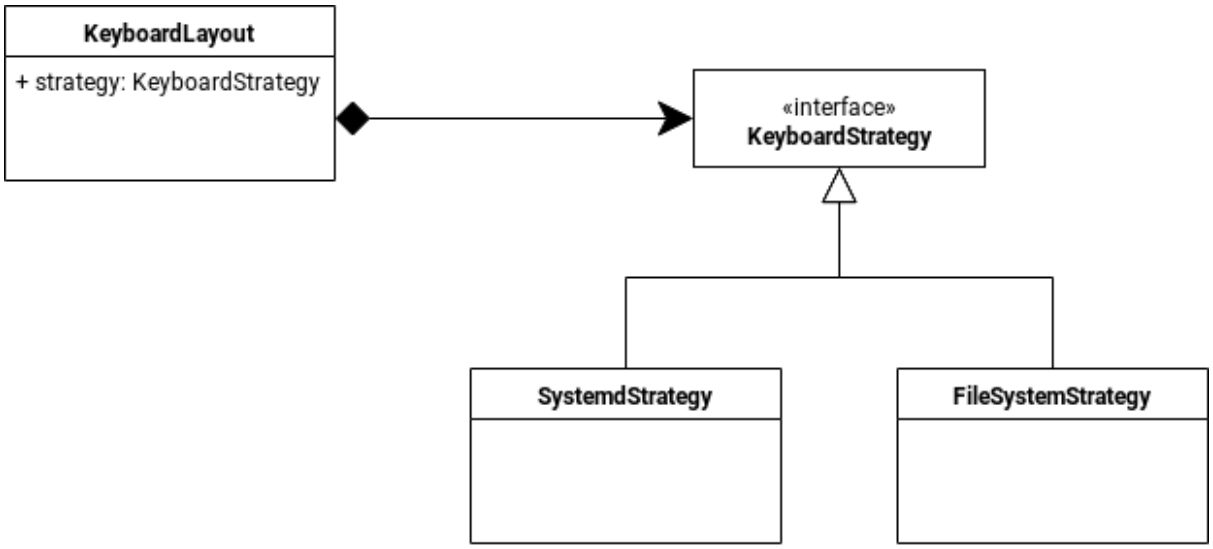


Figura 17: Diagrama UML con más de una implementación de la interfaz KeyboardStrategy.

Haber hecho uso de este patrón ha facilitado la aplicación del principio de diseño Open Closed Principle [29], que se basa en que la implementación debería estar abierta a extensión pero cerrada a modificación. Quiere decir que debe de ser posible añadir una nueva funcionalidad sin que ello conlleve realizar cambios en las implementaciones existentes. Esto es posible como se ha explicado al inicio de este apartado, creando una nueva implementación que pueda sustituir a la implementación de **systemd**, para ello ha de cumplir con el mismo contrato.

Esta ventaja que se ha obtenido se había expresado como requisito en la propuesta inicial del proyecto [1], como se puede leer en la siguiente cita:

«the interaction with systemd should be just one of many possible backends, so the module is ready for the next big change in Linux keyboard management or to be ported to systems without systemd.»

Extracto de la propuesta de proyecto inicial [1]

Las clases implicadas en la implementación de este patrón han sido dos principalmente, “KeyboardLayout” y “SystemdStrategy”. La clase “SystemdStrategy” es la implementación concreta de **systemd**. En la Figura 18 podemos ver esta clase.

```ruby
class SystemdStrategy
  # @return [Array<String>] an array with all available systemd keyboard layouts codes.
  def codes
    raw_layouts = Cheetah.run("localectl", "list-keymaps", stdout: :capture)
    raw_layouts.lines.map(&:strip)
  end

  # Use systemd-localed to apply a new keyboard layout in the system.
  # @param keyboard_layout [KeyboardLayout] the keyboard layout to apply in the system.
  def apply_layout(keyboard_layout)
    Cheetah.run("localectl", "set-keymap", keyboard_layout.code)
  end

  # @return [KeyboardLayout] the current keyboard layout in the system.
  def current_layout
    output = Cheetah.run("localectl", "status", stdout: :capture)
    get_value_from_output(output, "VC Keymap:").strip
  end

  def get_value_from_output(output, property_name)
    output.lines.map(&:strip).find { |x| x.start_with?(property_name) }.split(":", 2).last
  end

  private :get_value_from_output
end
```

Figura 18: Clase SystemdStrategy.

En la clase con la implementación para **systemd** se pueden observar los métodos públicos. Estos métodos son la interfaz del sistema desarrollado, el contrato que ha de cumplir cualquier implementación, de esta forma cualquier implementación será intercambiable.

Estos métodos son usados por la clase “KeyboardLayout”. Para que la clase “KeyboardLayout” y evitar así el acoplamiento a esta implementación se ha aplicado Dependency Inversion Principle, este principio se basa en no instanciar directamente una clase de una implementación concreta en la capa de negocio, sino abstraer una interfaz de dicha implementación a modo de contrato y recibir la instancia de la implementación concreta por lo que se denomina inyección de la dependencia, comúnmente se usa el constructor de la clase para este fin.

Con este objetivo se hace uso del método “*use*” para inyectar una instancia de la clase “SystemdStrategy”, consiguiendo con ello que la clase “KeyboardLayout” no

conozca la implementación de systemd, solo el contrato que ha de cumplir cualquier implementación, los métodos públicos. En este método "*use*" se almacena la instancia de esta implementación concreta en la variable de clase "*strategy*", esta variable de clase es usada en otros métodos de la misma clase. En la Figura 19 se puede observar la implementación de la clase nombrada.

```ruby
class KeyboardLayout
  # @return [String] code of the keyboard layout, for example 'us'.
  attr_reader :code
  # @return [String] description of the keyboard layout, for example 'English (US)'.
  attr_reader :description

  def initialize(code, description)
    @code = code
    @description = description
  end

  # Define the strategy and layout definitions to use.
  # @param strategy [Object] the strategy to apply the changes in the system.
  # @param strategy [Array<Object>] codes availables to use in the application
  # with the appropriate description
  def self.use(strategy, layout_definitions)
    @@strategy = strategy
    @@layout_definitions = layout_definitions
  end

  # @return [Array<KeyboardLayout>] an array with all available keyboard layouts.
  def self.all
    codes = @@strategy.codes
    layouts = @@layout_definitions.select { |x| codes.include?(x["code"]) }
    layouts.map { |x| KeyboardLayout.new(x["code"], x["description"]) }
  end

  def self.current_layout
    current_layout_code = @@strategy.current_layout
    layout_definition = @@layout_definitions.detect { |x| x["code"] == current_layout_code }
    KeyboardLayout.new(layout_definition["code"], layout_definition["description"])
  end

  # Apply a new keyboard layout in the system.
  def apply_layout
    @@strategy.apply_layout(self)
  end
end
```

Figura 19: Clase KeyboardLayout.

El método de clase "*use*" es llamado por el cliente. Este último es la clase "*SystemdKeyboard*" que se encuentra en el módulo (entiendase módulo como espacio de nombre) "*Clients*". Es en este nivel en donde se instancia la implementación de la estrategia a utilizar. Podemos ver la implementación de este cliente en la Figura 20.

```
25  module Y2Keyboard
26    module Clients
27      # Client with systemd implementation.
28      class SystemdKeyboard
29        def self.run
30          path = File.join(__dir__, "../data/keyboards.yml")
31          layout_definitions = YAML.load_file(path)
32          systemd_strategy = Y2Keyboard::Strategies::SystemdStrategy.new
33          Y2Keyboard::KeyboardLayout.use(systemd_strategy, layout_definitions)
34          Y2Keyboard::Dialogs::LayoutSelector.new.run
35        end
36      end
37    end
38  end
```

Figura 20: Clase SystemdKeyboard.

En la Figura 20 se observa cómo se crea una nueva instancia de la clase "*SystemdStrategy*" en la línea 32 y se usa como parámetro en la llamada al método "*use*" en la siguiente línea.

5.2 Interfaz gráfica de usuario

El desarrollo de la interfaz de usuario se ha realizado de igual forma que el resto de proyectos de YaST, es decir, haciendo uso de la librería **libYUI**. Esta librería proporciona una capa de abstracción sobre las librerías de interfaz gráfica más comunes en entornos GNU/Linux, que son **Qt**, **Gtk** y **ncurses**. Al desarrollar una única vez nuestra interfaz de usuario con **libYUI**, podremos ejecutarla en entornos con **Qt**, **Gtk** o **ncurses**. Con esta librería se simplifica en gran medida la creación de interfaces gráficas de usuario, limitando alguna de las funciones pero abstrayendo al desarrollador de la preocupación de tener que controlar complejas funciones. Otra ventaja es que de esta forma nos aseguramos en gran medida que se aplican los estilos definidos por YaST, en cuanto a la interfaz de usuario se refiere.

Como este proyecto se basa en reescribir un módulo ya existente de YaST, se ha tratado que la apariencia de cara al usuario sea lo más similar posible. En la Figura 21 y Figura 22 se puede observar el resultado obtenido en un entorno gráfico y en ncurses para entornos de consola virtual.

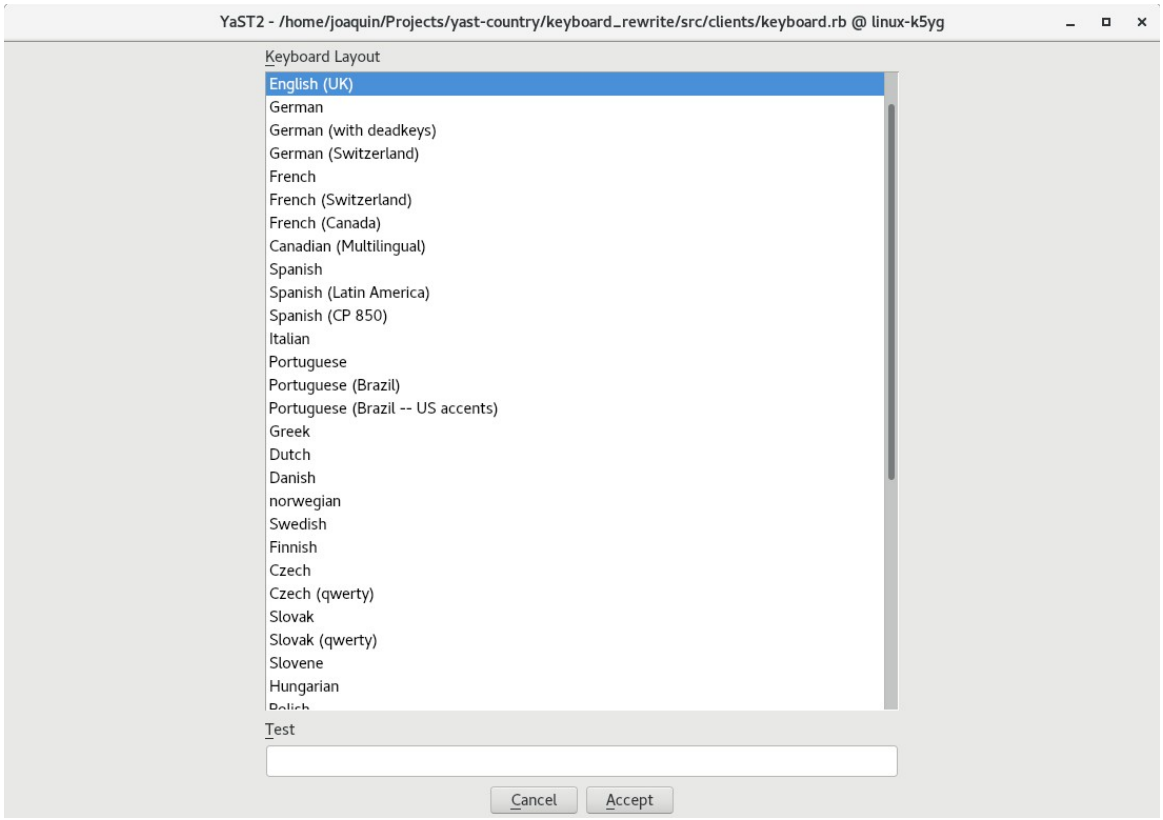


Figura 21: Diálogo principal de la aplicación.

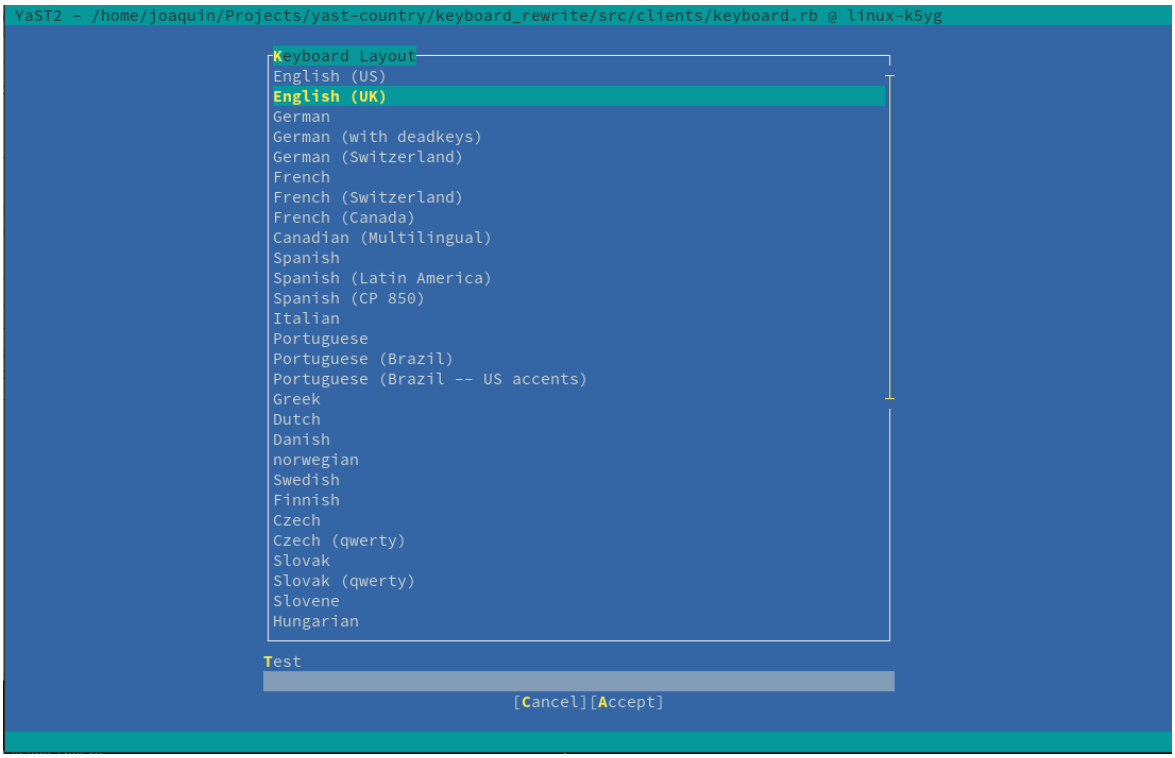


Figura 22: Diálogo principal de la aplicación en ncurses.

En la Figura 22 se muestra la aplicación en un entorno de consola virtual. En este tipo de entornos no podemos controlar la aplicación haciendo uso del ratón ya que se trata de una consola virtual y no de un entorno gráfico. Por esto se han tenido en cuenta los atajos de teclado para que el usuario pueda moverse por la aplicación, de esta forma con el uso de la tecla "*Alt*" seguida de la letra destacada en negrita y de color amarillo es posible moverse entre los distintos elementos de la interfaz. Por ejemplo si presionamos "*Alt + C*" ejecutará la acción del botón cancelar, o "*Alt + T*" pondrá el cursor en la caja de texto para probar el teclado. Estos atajos también se pueden aprovechar en entornos gráficos, pero son especialmente útiles en la consola virtual.

La clase "*LayoutSelector*" tiene la responsabilidad de mostrar el contenido del diálogo y gestionar los eventos que ocurren cuando el usuario interacciona con la interfaz de usuario. Para ello es necesario que esta clase herede de "*Dialog*" de **libYUI**, esto se puede ver en la Figura 23.

```ruby
# Main dialog where the layouts are listed and can be selected.
class LayoutSelector < UI::Dialog
  def initialize
    textdomain "country"
    @keyboard_layouts = KeyboardLayout.all
    @previous_selected_layout = KeyboardLayout.current_layout
  end
```

Figura 23: Definición de la clase LayoutSelector.

Para que la librería **libYUI** pueda iniciar el diálogo es importante que esta clase implemente el método "*dialog_content*". Dicho método devuelve la definición del diálogo que queremos mostrar al usuario.

```
38      def dialog_options
39        Opt(:decorated, :defaultsize)
40      end
41
42      def dialog_content
43        VBox(
44          HBox(
45            HWeight(20, HStretch()),
46            HWeight(50, layout_selection_box),
47            HWeight(20, HStretch())
48          ),
49          footer
50        )
51      end
```

Figura 24: Métodos dialog_content y dialog_options de la clase LayoutSelector.

En la Figura 24 vemos dos métodos, el primero, "*dialog_options*" simplemente establece unas opciones que van a ser tenidas en cuenta por la librería. El segundo, "*dialog_content*" es el método donde se define el contenido del diálogo. En él se hace uso de métodos de **libYUI** como VBox y HBox para definir el contenido. Nótese que se están ejecutando los métodos "*layout_selection_box*" y "*footer*". Estos métodos son propios de la clase. Vamos a ver el primero de ellos:

```
53      def layout_selection_box
54        VBox(
55          SelectionBox(
56            Id(:layout_list),
57            Opt(:notify),
58            _("&Keyboard Layout"),
59            map_layout_items
60          ),
61          InputField(Opt(:hstretch), _("&Test"))
62        )
63      end
64
65      def map_layout_items
66        @keyboard_layouts.map do |layout|
67          Item(
68            Id(layout.code),
69            layout.description,
70            layout.code == @previous_selected_layout.code
71          )
72        end
73      end
```

Figura 25: Métodos layout_selection_box y map_layout_items de la clase LayoutSelector.

Como se puede ver en la Figura 25, en el método "*layout_selection_box*" también se hace uso de métodos de la librería para definir el la caja de selección donde se muestran las distribuciones de teclado. Por otro lado el método "*map_layout_items*" devuelve una lista de items a mostrar en la caja de selección que se obtiene a partir de las distribuciones de teclado disponibles.

```
94      def footer
95        HBox(
96          HSpacing(),
97          PushButton(Id(:cancel), Yast::Label.CancelButton),
98          PushButton(Id(:accept), Yast::Label.AcceptButton),
99          HSpacing()
100       )
101     end
```

Figura 26: Método footer de la clase LayoutSelector.

El otro método usado por "*dialog_content*" es "*footer*" en el cual se definen los botones y el espaciado que aparecen en la parte inferior del diálogo.

Por último es necesario definir qué acciones se deben ejecutar cuando el usuario interactúa con los distintos elementos de la interfaz. Para ello se hace uso de los métodos que se observan en la Figura 27.

```
75      def accept_handler
76        selected_layout.apply_layout
77        finish_dialog
78      end
79
80      def cancel_handler
81        KeyboardLayoutLoader.load_layout(@previous_selected_layout)
82        finish_dialog
83      end
84
85      def layout_list_handler
86        KeyboardLayoutLoader.load_layout(selected_layout)
87      end
88
89      def selected_layout
90        selected = Yast::UI.QueryWidget(:layout_list, :CurrentItem)
91        @keyboard_layouts.find { |x| x.code == selected }
92      end
```

Figura 27: Métodos que realizan las acciones al interactuar con la interfaz de usuario de la clase LayoutSelector.

Para que un método sea ejecutado cuando el usuario interactúa con un determinado elemento de la interfaz es necesario que el nombre del método tenga la siguiente estructura “[símbolo]_handler”, dónde *símbolo* es el nombre del símbolo que se usó para llamar al método “*Id*”. Por ejemplo el método “*accept_handler*” se ejecutará al hacer click en el botón aceptar ya que cuando se define dicho botón se le pasó un identificador usando el símbolo “*:accept*”. Se puede ver de manera clara la definición del botón en la Figura 28.

```
PushButton(Id(:accept), Yast::Label.AcceptButton),
```

Figura 28: Definición del botón aceptar en la clase LayoutSelector.

Siguiendo esta norma, el método “*cancel_handler*” será ejecutado cuando el botón cancelar es accionado y “*layout_list_handler*” es ejecutado cuando se selecciona una opción en la caja de selección.

5.3 Cliente de YaST

La aplicación se trata de un módulo de YaST que ha de poder ser ejecutado por la aplicación principal de YaST. Para ello se ha creado lo que se denomina un cliente, que no es más que un fichero Ruby que inicia la aplicación. Podemos ver el contenido de este fichero en la Figura 29.

keyboard.rb ×

```
1  require_relative "../lib/y2keyboard/clients/systemd_keyboard"
2
3  Y2Keyboard::Clients::SystemdKeyboard.run
4
```

Figura 29: Fichero Keyboard.rb.

Como se puede ver simplemente se importa un cliente con una implementación específica, en este caso "*SystemdKeyboard*" y se ejecuta. A su vez el cliente "*SystemdKeyboard*" que se encuentra dentro del módulo de ruby "*Y2Keyboard::Clients*" y realiza las acciones necesarias para iniciar la aplicación y mostrar el diálogo.

En caso de querer usar una nueva implementación simplemente sería necesario crear el cliente específico para dicha implementación en el módulo de Ruby "*Y2Keyboard::Clients*" y usarlo en el cliente ejecutado por YaST.

5.4 Pasos seguidos para llegar a la implementación actual

El desarrollo se ha realizado en pequeños pasos. A continuación se describen los pasos que se han seguido con una breve descripción de las acciones más importantes realizadas en cada uno. Cada uno de estos pasos tiene asociado un *pull request* que son los cambios realizados durante dicha fase, en la descripción de cada uno de los pasos se encuentra la referencia al *pull request* asociado en el que se encuentran los cambios realizados.

5.4.1 Cargar distribuciones de teclado disponibles

Pull request ***#144*** [30].

Para arrancar el proyecto se ha decidido comenzar por mostrar al usuario la lista de distribuciones de teclado disponibles. Como primera aproximación solo se habilitaron sólo unas pocas distribuciones, ya que en esta temprana fase de desarrollo no era necesario disponer de todas las posibles opciones.

Esta funcionalidad implicó a desarrollar una primera versión de la interfaz gráfica de usuario, para poder mostrar dicho listado al usuario. A su vez también implicó crear el cliente para poder arrancar la aplicación. Además acarreó el desarrollo de la primera llamada al servicio **systemd-localed** para obtener las distribuciones disponibles en el sistema operativo.

5.4.2 Cambiar distribución de teclado del sistema

Pull request ***#168*** [31].

Una vez el usuario tiene la posibilidad de ver las distribuciones de teclado disponibles, el siguiente paso ha sido permitir al usuario cambiar la distribución de teclado usada por el sistema. Por lo que durante esta fase se siguió trabajando en la integración del módulo con **systemd**, ejecutando el comando "*localectl set-keymap [código de la distribución]*" cuando el usuario selecciona una distribución de teclado.

5.4.3 Probar distribución de teclado antes de aplicar los cambios en el sistema

Pull request ***#173*** [32].

En esta fase se siguió implementando funcionalidades existentes en el antiguo módulo. En este caso se trata de permitir al usuario probar una distribución de teclado antes de aplicarla en el sistema. Se trata de una funcionalidad importante ya que aparte de mejorar la experiencia de usuario, minimiza el riesgo de que el usuario se confunda al elegir una distribución y acabe aplicando en el sistema una distribución de teclado no deseada.

Algo a tener en cuenta en este punto es que esta funcionalidad ha de tener en cuenta si la aplicación se está ejecutando en un entorno con interfaz gráfica o si por el contrario se trata de un entorno de consola, ya que la forma de actualizar la distribución de teclado no es la misma en ambos entornos.

En este paso se quería cambiar la distribución de teclado en caliente, sin aplicarla en el sistema por lo que las herramientas proporcionadas por **systemd** no son las adecuadas para esta funcionalidad. Por lo que los comandos usados para conseguirlo han sido "*loadkeys*" para el entorno de consola virtual y "setxkbmap" para entornos gráficos.

5.4.4 Añadir configuración de RuboCop al proyecto

Pull request ***#174*** [33].

En este caso no se añadió ninguna funcionalidad nueva a la aplicación ya que se quería cubrir una necesidad de la fase desarrollo, el uso de **RuboCop** para asegurar el cumplimiento de la convención de código, ya que este ha sido el primer módulo que se encuentra en el repositorio "*yast-country"* que hace uso de esta herramienta. Es por ello que se ha configurado **RuboCop** de tal forma que solo compruebe los ficheros de este módulo.

5.4.5 Extraer una abstracción del repositorio de systemd.

Pull request ***#195*** [34].

Como se ha comentado este proyecto tiene como requisito importante evitar el acoplamiento con la implementación subyacente en el sistema, en este caso **systemd**. Debido a esto ha sido necesario esta fase en la que se ha trabajado para minimizar el acoplamiento de la aplicación a la implementación concreta de systemd. Con la idea de conseguir disminuir este acoplamiento durante esta fase se han aplicado los conceptos de **SOLID,** Single Responsibility Principle, Open Close Principle y Dependency Inversion Principle.

5.4.6 Seleccionar distribución de teclado usada por el sistema al iniciar la aplicación

Pull request ***#196*** [35].

Con esta fase se quiere continuar añadiendo funcionalidad a la aplicación, en este caso una sencilla y es que al arrancar el módulo aparezca seleccionada en la lista

de distribuciones de teclado aquella distribución que está siendo usada actualmente por el sistema. Por lo que durante este paso se siguió trabajando en la integración con **systemd**.

5.4.7 Añadir distribuciones de teclado disponibles a seleccionar

Pull request ***#197*** [36].

Una vez desarrolladas las funcionalidades anteriores se ha procedido ampliar la cantidad de distribuciones de teclado disponibles en la aplicación. Como el objetivo es suplantar al antiguo módulo, se han añadido todas las distribuciones de teclado disponibles en el módulo que actualmente usa **openSUSE**.

El código del módulo antiguo posee dos ficheros donde se encuentran las definiciones de estas distribuciones de teclado, estos ficheros son "*keyboard_raw_opensuse.ycp*" y "*keyboard_raw.ycp*". Para aclarar la duda de que fichero se debería usar como referencia para la definición de las distribuciones de teclado en el nuevo módulo, planteé la duda mediante un correo electrónico en la lista de correo yast-devel. A continuación podemos ver un extracto de un correo donde el integrante del equipo de **YaST**, Ancor Gonzalez Sosa, responde a la pregunta.

> «I would go for keyboard_raw. And just in case you have not done it already, please migrate it to a sane format like Yaml containing only the information that is still relevant nowadays.»

Extracto del correo enviado por Ancor Gonzalez Sosa [37].

Como se puede leer, la duda fue resuelta aclarando que se debería hacer uso del fichero "*keyboard_raw.ycp*" para este fin. Además se propone el uso de un formato como por ejemplo **YAML**, para la definición de las distribuciones de teclado. Esta sugerencia se hace con el objetivo de conseguir un mantenimiento más sencillo respecto al fichero usado actualmente.

YAML (YAML Ain't Markup Language) se define en su página oficial [38] como un estándar de serialización de datos legible por humanos y para todos los lenguajes de programación. Por su parte los ficheros con extensión "*.ycp*" son ficheros

heredados de la tecnología usada por los desarrolladores de YaST antes de usar Ruby como lenguaje de programación.

keyboard_raw.ycp

```
{
textdomain "country";

// map <string, list>

return ($[
  "english-us":
    [
    // keyboard layout
    _("English (US)"),
    $[
        "pc104": $[ "ncurses": "us.map.gz" ],
        "macintosh" : $[ "ncurses": "mac-us.map.gz" ],
        "type4": $[ "ncurses": "sunkeymap.map.gz" ],
        "type5": $[ "ncurses": "sunkeymap.map.gz" ],
        "type5_euro": $[ "ncurses": "sunkeymap.map.gz" ],
    ]
    ],
  "english-uk":
    [
    // keyboard layout
    _("English (UK)"),
    $[
        "pc104": $[ "ncurses": "uk.map.gz"],
        "macintosh" : $[ "ncurses": "mac-uk.map.gz" ],
        "type4": $[ "ncurses": "sunkeymap.map.gz"],
        "type5": $[ "ncurses": "sunt5-uk.map.gz"],
        "type5_euro": $[ "ncurses": "sunt5-uk.map.gz"],
    ]
    ],
  "german":
    [
    // keyboard layout
    _("German"),
    $[
        "pc104": $[ "ncurses": "de-latin1-nodeadkeys.map.gz"],
        "macintosh" : $[ "ncurses": "mac-de-latin1-nodeadkeys.map.gz"],
        "type4": $[ "ncurses": "sunkeymap.map.gz" ],
        "type5": $[ "ncurses": "sunt5-de-latin1.map.gz" ],
        "type5_euro": $[ "ncurses": "sunt5-de-latin1.map.gz" ],
```

Figura 30: Fichero keyboard_raw.ycp, fichero de configuración del módulo antiguo.

Partiendo del contenido del fichero “*keyboard_raw.ycp*”, podemos ver un extracto en

la Figura 30. Se ha creado un documento con extensión **YAML** con los datos mínimos necesarios para la nueva aplicación. En la Figura 31 se observa el resultado.

keyboards.yml

```yaml
- description: "English (US)"
  code: "us"

- description: "English (UK)"
  code: "uk"

- description: "German"
  code: "de-latin1-nodeadkeys"

- description: "German (with deadkeys)"
  code: "de-latin1"

- description: "German (Switzerland)"
  code: "sg-latin1"

- description: "French"
  code: "fr-latin1"

- description: "French (Switzerland)"
  code: "fr_CH-latin1"

- description: "French (Canada)"
  code: "cf"

- description: "Canadian (Multilingual)"
  code: "cn-latin1"

- description: "Spanish"
  code: "es"

- description: "Spanish (Latin America)"
  code: "la-latin1"

- description: "Spanish (CP 850)"
  code: "es-cp850"
```

Figura 31: Fichero keyboards.yml, definición de las distribución de teclado del módulo.

Se ha tratado de simplificar al máximo la configuración necesaria de las distribuciones disponibles en el módulo. Como se puede observar para añadir una nueva distribución de teclado solo es necesaria una descripción que será visible al usuario final y un código.

5.4.8 Arreglar un error al cambiar el la distribución de teclado en entorno gráfico

Pull request ***#198*** [39].

Con los cambios realizados en el paso anterior se encontraron casos en los que la aplicación fallaba y se cerrabaautomáticamente.

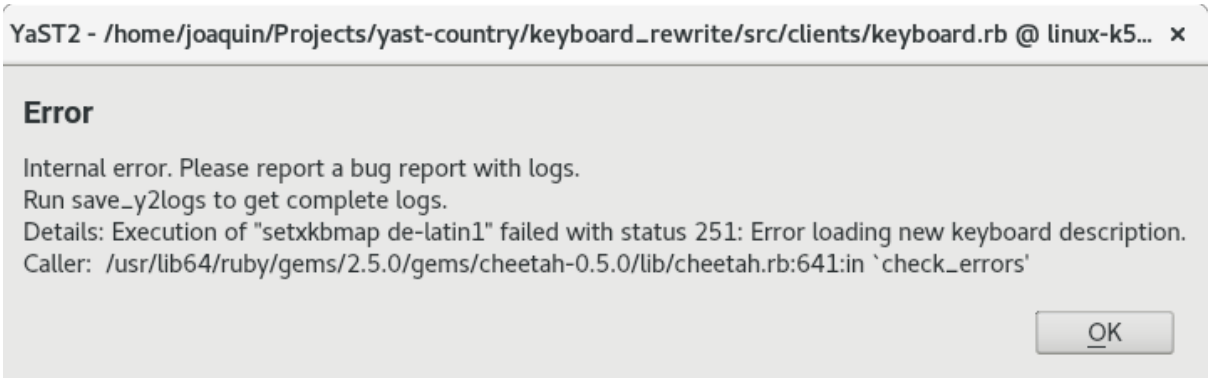


Figura 32: Mensaje de error al cerrarse la aplicación.

El problema se trataba exactamente al cambiar la distribución de teclado en caliente en el entorno gráfico. Esto solo ocurría con algunas distribuciones de teclado. Concretamente el problema se encontraba en los parámetros que se estaba pasando al comando "*setxkbmap*", ya que en determinados casos no eran los adecuados.

La solución fue hacer uso de la herramienta "**xkbctrl**" que posee **openSUSE**. Está facilita en gran medida la obtención de los parámetros necesarios para ejecutar "*setxkbmap*". En la siguiente Figura 33 se puede ver que el valor de la propiedad "*Apply*" son los parámetros necesarios para establecer la distribución inglesa de gran bretaña del teclado en el entorno gráfico.

```
$ /usr/sbin/xkbctrl uk
$[
   "XkbLayout"     : "gb",
   "XkbModel"      : "pc105",
   "XkbOptions"    : "terminate:ctrl_alt_bksp",
   "Apply"         : "-layout gb -model pc105 -option terminate:ctrl_alt_bksp"
]
```

Figura 33: Ejecución del comando xkbctrl.

5.4.9 Añadir documentación a clases y métodos

Pull request ***#199*** [40].

Aunque el proyecto posee sus propios tests que sirven de documentación del módulo, es buena práctica en proyectos de código abierto escribir un pequeño comentario explicativo en cada uno de los métodos más importantes del proyecto, así como en las clases y sus propiedades públicas.

```ruby
# This class represents a keyboard layout.
class KeyboardLayout
  # @return [String] code of the keyboard layout, for example 'us'.
  attr_reader :code
  # @return [String] description of the keyboard layout, for example 'English (US)'.
  attr_reader :description

  def initialize(code, description)
    @code = code
    @description = description
  end

  # Define the strategy and layout definitions to use.
  # @param strategy [Object] the strategy to apply the changes in the system.
  # @param strategy [Array<Object>] codes availables to use in the application
  # with the appropriate description
  def self.use(strategy, layout_definitions)
    @@strategy = strategy
    @@layout_definitions = layout_definitions
  end

  # @return [Array<KeyboardLayout>] an array with all available keyboard layouts.
  def self.all
    codes = @@strategy.codes
    layouts = @@layout_definitions.select { |x| codes.include?(x["code"]) }
    layouts.map { |x| KeyboardLayout.new(x["code"], x["description"]) }
  end
```

Figura 34: Fichero keyboard_layout.rb.

En la Figura 34 se observan comentarios a modo de documentación a nivel de clase, métodos y atributos. Estos comentarios son utilizados por la herramienta que es utilizada para generar la documentación de proyectos Ruby de YaST, esta herramienta se trata de **YARD** [41].

5.4.10 Dividir responsabilidades entre systemd y otras acciones sobre el sistema

Pull request ***#200*** [42].

Durante el desarrollo del módulo han utilizado distintas utilidades para interactuar con el sistema operativo, por lo que ha llegado un punto en que se ha visto la necesidad de aislar el uso de dichas utilidades en diferentes clases. En este caso se han separado las acciones que hacen uso de comandos de **systemd** de acciones que hacen uso de otras utilidades como "*loadkeys*" y "*setxkbmap*".

Para ello se ha creado la clase "*KeyboardLayoutLoader*" que será la encargada de hacer el cambio "en caliente" y no persistente de la distribución de teclado. Esto se logra haciendo uso de los comandos del sistema "*xkbctrl*", "*loadkeys*" y "*setxkbmap*" como se puede ver en la Figura 35.

```ruby
class KeyboardLayoutLoader
  include Yast::Logger

  # Load x11 or virtual console keys on the fly.
  # @param keyboard_layout [KeyboardLayout] the keyboard layout to load.
  def self.load_layout(keyboard_layout)
    load_x11_layout(keyboard_layout) if !Yast::UI.TextMode
    begin
      Cheetah.run("loadkeys", keyboard_layout.code) if Yast::UI.TextMode
    rescue Cheetah::ExecutionFailed => e
      log.info(e.message)
      log.info("Error output:    #{e.stderr}")
    end
  end

  # Load x11 keys on the fly.
  # @param keyboard_layout [KeyboardLayout] the keyboard layout to load.
  def self.load_x11_layout(keyboard_layout)
    output = Cheetah.run("/usr/sbin/xkbctrl", keyboard_layout.code, stdout: :capture)
    arguments = get_value_from_output(output, "\"Apply\"").tr("\"", "")
    setxkbmap_array_arguments = arguments.split.unshift("setxkbmap")
    Cheetah.run(setxkbmap_array_arguments)
  end

  def self.get_value_from_output(output, property_name)
    output.lines.map(&:strip).find { |x| x.start_with?(property_name) }.split(":", 2).last
  end

  private_class_method :get_value_from_output, :load_x11_layout
end
```

Figura 35: Clase KeyboardLayoutLoader.

6 Resultado y trabajos futuros

6.1 Resultado actual

Durante el desarrollo de este proyecto se ha logrado construir las funcionalidades principales de este módulo, que son: listar las distribuciones de teclado disponibles, poder probar una distribución de teclado sin aplicarla en el sistema y cambiar distribución de teclado del sistema. Nótese que estas son todas las funcionalidades relacionadas con systemd. El resto de funcionalidades no están relacionadas con esta tecnología.

6.2 Próximos pasos: paso a producción del módulo

En los inicios de este proyecto se había planteado el deseo de llevar el módulo desarrollado a producción, más allá del objetivo del trabajo de desarrollar el módulo. Usando para ello la versión rolling release de **openSUSE**, denominada Tumbleweed. Este objetivo extra no ha podido ser cumplido aún. Por un lado aún no ha sido posible revisar la totalidad de los *pull requests* creados hasta el momento y por otro lado no se han implementado todas las funcionalidades que ofrece el módulo que actualmente usan los usuarios. Así pues, llevar este módulo a producción en su estado actual conlleva una pérdida de funcionalidad para los usuarios finales.

Por ello el siguiente paso inmediato es terminar de revisar con la ayuda de el equipo de YaST los *pull requests* pendientes. La partiticipación de los integrantes de YaST se hace imprescindible para contar con su validación técnica, que permita dar el siguiente paso hacia su integración en openSUSE.

Una vez hecho esto, el siguiente paso sería desarrollar las funcionalidades que han quedado pendientes, que son las denominadas como “opciones avanzadas” en el antiguo módulo. Estas opciones son mostradas en un nuevo diálogo e incluyen las funcionalidades siguientes: establecer el “repeat rate” y “delay before repetition starts”, habilitar o deshabilitar el “num lock” al iniciar el sistema y por último deshabilitar o habilitar “caps lock”.

Una vez desarrolladas dichas funcionalidades se habrá completado el desarrollo del

módulo al reimplementar la totalidad de las funcionalidades ya existentes. Así se podría continuar con los procedimientos necesarios para llevar el módulo desarrollado a producción. Para ello se podrá contar con la ayuda de el equipo de YaST y la comunidad de **openSUSE**.

6.3 Trabajos futuros

Más allá de sustituir las funcionalidades del módulo actual, existen diversas posibilidades a explorar, por ejemplo se podría integrar con otros módulos de YaST como puede ser el módulo de idioma y localización.

Tambien podría ser una posible mejora incluir la funcionalidad de que el usuario añadir nuevas distribuciones de teclado a la lista de posibles opciones siempre que esten disponibles en su sistema.

Centrandonos aún más en la experiencia de usuario, se podría investigar la posibilidad de crear un flujo de uso en el que el usuario introduzca una serie de caracteres en su teclado y el sistema sea capaz de detectar que distribución de teclado está usando el usuario, con la idea de facilitar al usuario la configuración de su sistema.

7 Conclusiones

7.1 Aportaciones a la comunidad

Tras la realización de este trabajo se ha obtenido un módulo de YaST para la gestión del teclado del sistema operativo. La implementación de este módulo está preparada para interactuar con sistemas que hagan uso de systemd.

Durante el desarrollo de este módulo se ha tenido en cuenta que debe de estar preparado para el siguiente gran cambio en cuanto a la gestión del teclado en sistema Linux, como ya se expresó como requisito en la propuesta inicial del proyecto. Es por ello que el desarrollo se ha realizado teniendo muy en cuenta los principios de diseño de orientación a objetos así como aquellos patrones de diseño que aplicaban dada la naturaleza del problema.

Gracias a esto se ha conseguido reimplementar las funcionalidades esenciales y un buen número de funcionalidades añadidas del módulo actual, con lo cual el nuevo desarrollo queda en un estado cercano para reemplazar al módulo actual. Por ello considero que el proyecto puede considerarse exitoso.

7.2 Aportaciones a mi desarrollo profesional

A nivel personal, haber desarrollado este módulo ha sido una gran experiencia, he podido aprender cómo funciona el desarrollo en el mundo de software libre. Por ejemplo, cómo comenzar a colaborar con un proyecto, qué flujo de desarrollo se sigue en la mayoría de proyectos de código abierto y cómo comunicarse en este tipo de comunidades. Que se trate de un trabajo en colaboración con un proyecto de software libre de escala internacional me ha hecho aprender lo que significa colaborar con un equipo descentralizado, donde los integrantes viven en diversas partes del planeta, y lo que ello implica como horarios completamente distintos.

A nivel tecnológico, ha sido todo un reto trabajar en un lenguaje como Ruby, que si bien se trata de un lenguaje orientado a objetos, presenta algunas características muy distintas a los lenguajes que acostumbro a usar o que se enseñan en el ámbito de nuestra universidad. Esto tiene la ventaja de que me ha aportado otro punto de vista a la hora de usar lenguajes orientados a objetos. Usar Ruby me ha hecho

también conocer diversas herramientas que giran entorno al lenguaje Ruby, como RuboCop, Rspec, etc.

El desarrollo de este trabajo de fin de grado ha significado para mí que he aportado un pequeño grano de arena en el gran proyecto que es **openSUSE**. Al colaborar con desarrolladores de diversos puntos del planeta, he sentido que he contribuido a mantener vivos al proyecto y a la comunidad.

8 Glosario de términos

- **Distribución de teclado**: Es la disposición de las teclas del teclado, puede variar debido al tipo de distribución, por ejemplo qwerty o azerty, y tambien por el idioma.
- **Distribución GNU/Linux**: Es un sistema operativo que hace uso de las herramientas de GNU/Linux.
- **Fork**: Bifurcación de un repositorio de código.
- **GitHub**: Portal web para el alojamiento de repositorios de código. En este servició se encuentran alojados los repositorios de openSUSE.
- **GNU/Linux**: Proyecto libre que tiene como objetivo crear un sistema operativo usando software libre, y Linux como núcleo del sistema.
- **Issue**: Propuesta de cambio, puede ser de varios tipos, un error, una mejora, una nueva funcionalidad.
- **libYUI**: librería usada para la creación de la interfaz gráfica de usuario.
- **Pull request**: Es una propuesta de cambio en un repositorio de código.
- **RuboCop**: Herramienta usada para comprobar el cumplimiento de las convenciones de código.
- **systemd**: conjunto de herramientas software que proporciona elementos fundamentales para un sistema operativo Linux. Es usado por openSUSE.
- **systemd-localed**: servicio de systemd que gestiona la localización y el teclado del sistema.

9 Índice de Figuras

Figura 1: Menú de YaST.....8
Figura 2: Parte del resultado obtenido al ejecutar el comando “localectl list-keymaps”.....21
Figura 3: Información obtenida al ejecutar el comando “localectl status”.....22
Figura 4: Mensaje enviado para comunicar el comienzo del proyecto.....23
Figura 5: Diagrama ideal de flujo de las ramas.....26
Figura 6: Logo RSpec.....27
Figura 7: Implementación de un test haciendo uso de RSpec.....28
Figura 8: Resultado de ejecutar un test que no tiene una correcta implementación. 29
Figura 9: Resultado de ejecutar un test con una correcta implementación.....29
Figura 10: Resultado obtenido al ejecutar los tests con el formato de documentación.....30
Figura 11: Logotipo de RuboCop.....31
Figura 12: Diversos ejemplos de escribir un if en Ruby.....32
Figura 13: Ejemplo salida de RuboCop.....33
Figura 14: Ejemplo salida de RuboCop con “offenses”.....33
Figura 15: Fichero .rubocop.yml, configuración de RuboCop.....34
Figura 16: Diagrama UML de la implementación actual.....36
Figura 17: Diagrama UML con más de una implementación de la interfaz KeyboardStrategy.....37
Figura 18: Clase SystemdStrategy.....38
Figura 19: Clase KeyboardLayout.....39
Figura 20: Clase SystemdKeyboard.....40
Figura 21: Diálogo principal de la aplicación.....41
Figura 22: Diálogo principal de la aplicación en ncurses.....41
Figura 23: Definición de la clase LayoutSelector.....42
Figura 24: Métodos dialog_content y dialog_options de la clase LayoutSelector….43
Figura 25: Métodos layout_selection_box y map_layout_items de la clase

LayoutSelector.....44
Figura 26: Método footer de la clase LayoutSelector.....44
Figura 27: Métodos que realizan las acciones al interactuar con la interfaz de usuario de la clase LayoutSelector.....45
Figura 28: Definición del botón aceptar en la clase LayoutSelector.....45
Figura 29: Fichero Keyboard.rb.....46
Figura 30: Fichero keyboard_raw.ycp, fichero de configuración del módulo antiguo. 50
Figura 31: Fichero keyboards.yml, definición de las distribución de teclado del módulo.....52
Figura 32: Mensaje de error al cerrarse la aplicación.....53
Figura 33: Ejecución del comando xkbctrl.....54
Figura 34: Fichero keyboard_layout.rb.....54
Figura 35: Clase KeyboardLayoutLoader.....56

10 Referencias

1: Propuesta de proyecto openSUSE mentoring, https://github.com/openSUSE/mentoring/issues/79#issue-204992792

2: What is free software?, https://www.gnu.org/philosophy/free-sw.html

3: GPL-2 License Text, https://www.gnu.org/licenses/gpl-2.0.html

4: systemd System and Service Manager, https://www.freedesktop.org/wiki/Software/systemd/

5: Página documentación de systemd-localed , https://www.freedesktop.org/software/systemd/man/systemd-localed.service.html

6: Documentación de SUSE para la gestión del teclado, https://www.suse.com/documentation/sled-12/book_sle_deployment/data/cha_y2_hw_keym.html

7: Ruby style guide, https://github.com/SUSE/style-guides/blob/master/Ruby.md#ruby-style-guide

8: openSUSE:YaST: Code Organization , https://en.opensuse.org/openSUSE:YaST:_Code_Organization

9: Repositorio de software YaST:Head, https://build.opensuse.org/project/show/YaST:Head

10: Repositorio de software openSUSE:Factory, https://build.opensuse.org/project/show/openSUSE:Factory

11: Principios SOLID, https://en.wikipedia.org/wiki/SOLID

12: Don't Repeat Yourself principle, https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

13: Law of Demeter principle, https://en.wikipedia.org/wiki/Law_of_Demeter

14: John Vlissides, Ralph Johnson, Richard Helm, Erich Gamma, Design Patterns: Elements of Reusable Object-Oriented Software, 1994

15: Repositorio mentoring de openSUSE, https://github.com/openSUSE/mentoring

16: Repositorio del proyecto yast-country, https://github.com/yast/yast-country

17: Página web del IRC freenode.net , http://freenode.net/

18: Lista de correos yast-devel, https://lists.opensuse.org/yast-devel/

19: Email en la lista de correos yast-devel para comunicar el inicio del proyecto, https://lists.opensuse.org/yast-devel/2017-09/msg00050.html

20: GitHub Workflow for Open Source Projects, https://github.com/shundhammer/qdirstat/blob/master/doc/GitHub-Workflow.md

21: YaST Contribution Guidelines, https://github.com/yast/yast-country/blob/master/CONTRIBUTING.md#yast-contribution-guidelines

22: Kent Beck, Extreme Programming Explained: Embrace Change, 1999

23: Página web oficial de RSpec, http://rspec.info/

24: Better Specs, http://www.betterspecs.org/

25: Rubocop repository, https://github.com/rubocop-hq/rubocop

26: Fichero README.md de RuboCop, https://github.com/rubocop-hq/rubocop/blob/master/README.md

27: Fichero rubocop_yast_style.yml, https://github.com/yast/yast-devtools/blob/62a49a654a4bd402c478d434808c7ea50024b610/ytools/y2tool/rubocop_yast_style.yml

28: Duck Typing, http://rubylearning.com/satishtalim/duck_typing.html

29: Open closed principle, https://en.wikipedia.org/wiki/Open%E2%80%93closed_principle

30: Pull request Keyboard rewrite: Load keyboard layouts , https://github.com/yast/yast-country/pull/144

31: Pull request Keyboard rewrite: Change keyboard layout, https://github.com/yast/yast-country/pull/168

32: Pull request Keyboard rewrite: Test keyboard layout before change it, https://github.com/yast/yast-country/pull/173

33: Pull request Keyboard rewrite: Add rubocop, https://github.com/yast/yast-country/pull/174

34: Pull request Keyboard rewrite: Use strategy pattern to avoid coupling, https://github.com/yast/yast-country/pull/195

35: Pull request Keyboard rewrite: select current layout on startup, https://github.com/yast/yast-country/pull/196

36: Pull request Keyboard rewrite: add more layouts to the module, https://github.com/yast/yast-country/pull/197

37: Email en respuesta a "Some questions about keyboard layouts", https://lists.opensuse.org/yast-devel/2018-05/msg00023.html

38: Página oficial YAML, https://yaml.org/

39: Pull request Keyboard rewrite: fix error setting x11 keymap,

https://github.com/yast/yast-country/pull/198

40: Pull request Keyboard rewrite: add documentation and license information, https://github.com/yast/yast-country/pull/199

41: Página web de YARD, https://yardoc.org/

42: Pull request Keyboard rewrite: encapsulate systemd actions, https://github.com/yast/yast-country/pull/200