UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

Departamento de Ingeniería Informática y Ciencias de la Computación

VISUALIZACIÓN DE DATOS PARA MONITOREO ESTRUCTURAL DE PUENTES MEDIANTE DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS

Tesis presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al grado de Magíster en Ciencias de la Computación

Por: Braulio Quiero Hernández
Profesor Guía: Gonzalo Rojas Durán

Concepción, Chile 2021

Tabla de Contenidos

Índice de Tablas

Índice de Ilustraciones

Resumen

El monitoreo de salud estructural (SHM) se define como el proceso de implementación de estrategias en la identificación de daño en estructuras en ingeniería civil, aeroespacial y mecánica [6]. Es un área multidisciplinaria dentro de la ingeniería, que involucra expertos en monitoreo estructural, análisis de datos, electrónica, computación, entre otros [7]. Los sistemas Software que apoyan el SHM poseen una arquitectura compleja que necesita dar soporte, entre otras prestaciones, a la visualización de los datos en tiempo real y datos históricos (para apoyar la toma de decisiones en relación al mantenimiento de la estructura) [4].

En este punto, existen plataformas software para SHM tanto privativas como libres, que centran sus esfuerzos en la visualización en tiempo real. Sin embargo, el análisis de datos históricos, que permite analizar patrones de medición y correlación entre variables, no cuenta con similar nivel de apoyo por parte de herramientas software. Así, la implementación de alternativas de visualización y exploración de los datos queda restringida a las capacidades de programación de los interesados.

En el caso de los expertos en monitoreo estructural, estos no necesariamente cuentan con las capacidades suficientes para, mediante el uso de lenguajes y herramientas de programación, implementar una nueva visualización.

En este documento se propone un enfoque dirigido por modelos en el desarrollo de alternativas de visualización en el Monitoreo de Salud Estructural de Puentes para reducir la complejidad en el desarrollo de alternativas de visualización. El enfoque considera la generación automática del código necesario, a partir de un Meta-modelo que definirá un lenguaje específico de dominio (DSL) que describa y asocie objetivos de exploración de datos con alternativas de visualización, en un alto nivel de abstracción. A partir de este Meta-modelo y mediante el uso de una herramienta de modelado gráfico, los usuarios podrán generar automáticamente el código de la visualización modelada, pudiendo utilizar directamente dicho código en plataformas destino. Así, el experto del dominio, prescindiendo de conocimientos avanzados de programación, puede generar rápidamente las visualizaciones que mejor respondan a sus propios requerimientos de análisis de los datos, para la toma de decisiones oportunas.

La propuesta fue evaluada en términos de cobertura de alternativas de visualización generadas a partir de una plataforma frente a un entorno de desarrollo representativo, usabilidad y tiempo de desarrollo en comparación a un enfoque tradicional.

Abstract

Structural Health Monitoring (SHM) is defined as the process of implementation of strategies in the identification of structural damage in civil, aerospace and mechanic engineering [6]. It is a multidisciplinary area inside engineering which involves experts on structural monitoring, data analysis, electronics, computing, among others [7]. Software systems that support SHM have a complex architecture that has to give support, among other benefits, towards visualization of data in real time and historical data (to support the decision making in relation to the maintenance of the structure) [4].

At this point, there are private as well as free software platforms for SHM which centers their efforts on real time visualization. However, the analysis of historical data that allows to analyze mediation patterns and correlation between variables does not count with a similar level of support from software tools. Therefore, the implementation of visualization of alternativates and the data exploration is restricted to the programming capabilities of the interested parties.

In the case of the experts of structural monitoring, they do not necessarily count with enough capabilities – through the use of programming language and tools – to implement a new visualization.

In this document it is proposed a model driven approach in the development of alternatives of visualization on the Structural Health Monitoring of Bridges to reduce the complexity in the development of visualization alternatives. The approach considers the automatic generation of the necessary code from a meta-model that will define a domain-specific language (DSL) that describes and associate objetives of data exploration with alternatives of visualization at a higher level of abstraction. From this metal-model and through the use of a graphic modeling tools, the ussers will be able to automatically generate the code of a modeled visualization, thus being able to directly use the aforementioned code on their destination platforms. Thus, domain's expert, regardless of his advanced knowledge in programming, can quickly generate visualizations that respond better to his own requirementes of data analysis for a timely decision-making.

The proposal was evaluated in terms of coverage on the visualization alternatives generated from a platform versus a representative development enviroment, ussability and development time in comparison with the traditional approach.

1. INTRODUCCIÓN

Los sistemas de Monitoreo de Salud Estructural (SHM, del inglés Structural Health Monitoring) permiten detectar daños de infraestructuras en ingeniería civil, mecánica y aeroespacial donde el daño es definido como cambios en el material, propiedades geométricas, entre otras. Para detectar este daño, los expertos en el dominio analizan datos provenientes de múltiples sensores instalados en la estructura [6].

El análisis de datos históricos permite a los expertos en el dominio identificar el daño y la prolongación de este mediante modelos analíticos [8]. Para realizar el análisis, los profesionales deben utilizar herramientas externas para la visualización de los datos y apoyarse en APIs disponibles en los distintos lenguajes de programación específicos para este dominio (R, Python, Matlab). Los expertos en SHM deben codificar cada gráfico que será generado. Si desean generar reportes o visualización web, deben recurrir a otros profesionales que manejen lenguajes de programación para este dominio (html, javascript, otros). Aunque existen actualmente bibliotecas específicas orientadas a la visualización y la generación de reportes web en algunos de los lenguajes de programación mencionados, el experto en el dominio necesita adquirir experiencia en el uso de estas bibliotecas.

El presente trabajo propone un enfoque de Desarrollo de Software Dirigido por Modelos para la generación automática de alternativas de visualización para el Monitoreo de Salud Estructural de Puentes. Usando este enfoque, los expertos en el dominio, prescindiendo de conocimientos avanzados de programación podrán generar sus propias alternativas de visualización y el desarrollo de las aplicaciones se reduce a la creación de ejemplares del modelo a partir de una herramienta gráfica de modelado.

1.1. Hipótesis

Se propone en este trabajo que la utilización de un enfoque dirigido por modelos en el desarrollo de alternativas de visualización para sistemas de monitoreo de salud estructural, permitirá simplificar y reducir el tiempo de implementación que enfrentan los expertos en monitoreo de puentes (u otras estructuras civiles) para construir estos sistemas, en comparación con el desarrollo basado en la codificación.

Para mostrar esto se ha creado, mediante el enfoque de desarrollo de software dirigido por modelos, una herramienta software capaz de generar visualizaciones

del dominio de manera automática mediante un Meta-modelo basado en las abstracciones obtenidas del análisis del dominio.

1.2. Objetivos

1.2.1. Objetivo General

Definir una propuesta de desarrollo de software dirigida por modelos para la generación automática de alternativas de visualización en sistemas de monitoreo de salud estructural, que permita simplificar y reducir el tiempo de desarrollo, por parte de expertos en el dominio, de alternativas de visualización para monitoreo estructural de puentes.

1.2.2. Objetivos Específicos

Para lograr dicho objetivo se han planteado los siguientes objetivos específicos

- Definir un Lenguaje Específico de Dominio para visualizaciones de datos almacenados, que permita modelar distintas alternativas de visualización.
- Definir las trasformaciones que permitirán, a partir del lenguaje específico de dominio, generar las alternativas de visualización.
- Evaluar la propuesta dirigida por modelos en términos de cobertura de alternativas de visualización en sistemas de monitoreo estructural, usabilidad de herramienta de edición de modelos de visualización y tiempo de implementación en comparación a un enfoque manual.

1.3. Metodología

Debido a la naturaleza de esta propuesta, de acuerdo con la hipótesis planteada, se ha adoptado una metodología de Investigación-Acción (Action-Research). Esta metodología, busca que los resultados de la investigación sean beneficiosos tanto para la comunidad científica, como para los participantes involucrados en el proceso de desarrollo [9].
En esta metodología, es necesario identificar los roles que desempeñan los participantes. Estos roles, se describen a continuación:

- El investigador: En este caso, quién presenta esta propuesta.
- El objeto investigado: El proceso de desarrollo de visualizaciones de datos para monitoreo estructural de puentes.

- El grupo crítico: Equipo del proyecto FONDEF Plataforma de Monitoreo Estructural de Puentes IT18I0112F.
- El beneficiario: Expertos en monitoreo estructural de puentes, expertos en visualización de datos, desarrolladores en el campo de la visualización de datos en plataformas web.

Esta metodología consiste en 4 etapas. Se describen a continuación, las tareas realizadas en estas etapas:

- Planificación: Se definieron en esta etapa las distintas metodologías utilizadas. Entre estas metodologías está la metodología para la investigación y la metodología para el desarrollo del producto software necesario para la implementación de esta propuesta. Además se definió la propuesta para la recolección de los resultados.
- Acción: En esta etapa, se llevó a cabo el proceso de investigación del estado del arte. En conjunto con este proceso, se realizó el desarrollo del software. Este desarrollo se ejecutó mediante el marco de trabajo adoptado en esta propuesta de investigación. Luego de cada iteración, se realizó una reunión con stakeholders para recibir retroalimentación de acuerdo a la etapa de Observación.
- Observación: Se recogió la retroalimentación tomada a partir de la etapa de acción. En la última iteración, se recibieron los resultados finales (cuestionarios, producto final, entre otros).
- Reflexión: Se analizaron los resultado obtenidos en las etapas anteriores.

1.3.1. Investigación del Estado del Arte:

Consistió en una búsqueda sistematizada [10], esta incluye algunos elementos de la búsqueda sistemática.
La búsqueda realizada incluyó los siguientes términos como palabras claves:

- User Interface
- System Health Monitoring
- Bridge
- Data Visualization
- Big Data
- Model Driven Software Development

Primero se realizó una búsqueda por separado de System Health Monitoring y Model Driven Software Development para obtener un marco teórico sobre los temas a abordar. Posteriormente, se realizó una combinación de los siguiente términos:

- Model Driven Software Development AND (System Health Monitoring OR Data Visualization OR User Interface OR Big Data) (616.000 resultados de búsqueda obtenidos).
- System Health Monitoring AND (Bridge OR Data Visualization OR Big Data) (3.010.000 resultados de búsqueda obtenidos).
- Model Driven Software Development AND System Health Monitoring AND Bridge (146.000 resultados de búsqueda obtenidos).

Se incluyeron artículos que incorporaban estos términos en el Título, Resumen o Introducción.

La búsqueda se realizó ordenando los artículos por relevancia, según el criterio de la base de datos consultada, luego se buscó ordenando los artículos por fecha (desde los más reciente a los más antiguos).
Se descartaron las publicaciones que:

- Contenían información de SHM y no estaban enfocadas a puentes.
- Trabajos similares de los mismos autores, seleccionando aquellos con mayor número de citas.
- Se descartaron trabajos genéricos de visualización.

La información recolectada sirvió para establecer un marco teórico y realizar una comparación con los trabajos relacionados.

1.3.2. Definición de un Lenguaje Específico de Dominio

Para la definición de un Lenguaje Específico de Dominio (DSL, del inglés Domain Specific Language) fue necesario:

- Generar un Meta-modelo que definió las reglas sintácticas y semánticas para los modelos generados a partir del DSL.
- Probar que el DSL es capaz de generar alternativas de visualización. Esto, mediante la construcción de una herramienta basada en el Meta-Modelo que genera estas visualizaciones.

Este proceso se llevó a cabo en base al proceso de ingeniería en Linea de Productos que consiste en 3 etapas, las que, por las necesidades de validación de interfaces se realizaron en forma iterativa.

- Análisis del Dominio: En esta etapa se establece el alcance del dominio, la variabilidad y la estructuración del dominio.
- Diseño del Dominio: Se define la arquitectura base y el plan de producción.
- Implementación del Dominio: Se generan los componentes, el DSL y el generador.

1.3.3. Definición de la Transformación de Modelo a Texto

Para cumplir con este objetivo se realizó una comparación de los lenguajes existentes que poseen bibliotecas para la visualización de datos.

- Se estudiaron los lenguajes existentes con bibliotecas de visualización para elegir el mejor candidato para la transformación del modelo basado en: facilidad de uso de biblioteca para visualización e integración con plataforma en la que se representa.
- Se identificaron los elementos del modelo mapeados al lenguaje elegido.
- Se llevó a cabo la transformación del modelo a los lenguaje elegidos mediante una herramienta adecuada para este fin.

1.3.4. Evaluación

- Para evaluar la propuesta en términos de cobertura de alternativas de visualización se generaron distintas alternativas de visualización a partir de una herramienta construida mediante desarrollo dirigido por modelos en este proyecto.
- Para evaluar la propuesta Dirigida por Modelos en términos de usabilidad se planteó un diseño de estudio de caso donde los stakeholders generaron distintas alternativas de visualización mediante una herramienta construida en base al DSL que se propone en este proyecto. Finalmente, se aplicó un cuestionario basado en una encuesta de usabilidad propuesta por IBM [11] utilizada en investigaciones con esta componente.
- Para evaluar la propuesta en términos de tiempo de implementación con respecto a un enfoque tradicional se comparó el tiempo de desarrollo usando un enfoque tradicional, con el tiempo de desarrollo utilizando una herramienta de generación de visualizaciones construida en este proyecto. Se convocó la participación de desarrolladores y expertos del dominio.

1.3.5. Alcance y limitaciones

Este trabajo presenta una propuesta metodológica para el desarrollo de las aplicaciones en el dominio del monitoreo de salud estructural de puentes, en particular, en la generación automática de visualizaciones. Aunque no se descarta que el enfoque aquí planteado pueda extenderse a otros dominios en el marco del monitoreo de salud estructural o visualizaciones automáticas.

Si bien existen desafíos en cuanto a la generación de visualizaciones de datos recientes en el monitoreo de salud estructural, como el manejo de grandes volúmenes de datos, estos desafíos están fuera del alcance de este trabajo y se deja en manos de bibliotecas especializadas.

El Meta-modelo generado en este proyecto fue construido con múltiples clases que dan la posibilidad de extender tanto las alternativas de visualización como la cantidad y tipo de sensores. Por ello, no todos los gráficos y sensores planteados en el modelo fueron implementados en la herramienta de modelado, pero existe la posibilidad de generarlos a futuro utilizando el mismo Meta-modelo o una actualización de este.

1.4. Estructura del informe

Este informe está organizado en 10 secciones.

La Sección 1 (pág. 1) muestra una descripción global del documento, contiene el planteamiento del problema y la solución propuesta, los objetivos para llevar a cabo este trabajo y la metodología utilizada. Además, presenta los alcances y limitaciones de la propuesta en la generación de alternativas de visualización de datos recientes, utilizando la metodología planteada. También presenta una breve descripción de cada capítulo presente en este informe.

La Sección 2 (pág. 9) muestra el trabajo relacionado. Está dividido en dos partes. Una parte, corresponde a los trabajos relacionados en el área de Sistemas de Monitoreo Estructural, enfocado en aquellos trabajos que han propuesto alternativas de visualización, pero también incorporando otros que proponen marcos de trabajo para el desarrollo de aplicaciones en este dominio. La segunda parte de esta sección se centra en aquellos trabajos que han utilizado el desarrollo de software dirigido por modelos como metodología para resolver problemas que plantean la visualización de datos. También, se han incluido trabajos que proponen esta metodología para resolver problemas que implican el uso de sensores.

La Sección 3 (pág. 20) presenta los conceptos fundamentales que son usados para la formulación de la propuesta planteada en el presente informe, se encuentra dividido en 3 partes. La primera parte, entrega los conceptos base sobre monitoreo de salud estructural de puentes que son necesarios para comprender este trabajo. Entre estos conceptos están: monitoreo de salud estructural, daño en estructuras, tipos de sensores utilizados para medir el daño en la estructura, entre otros. También, se muestran algunas de las características que deben tener los sistemas construidos para el monitoreo de salud estructural. La segunda parte, presenta los conceptos de Desarrollo de Software Dirigido por Modelos, dentro de estos conceptos están: Meta-modelo, Transformación de Modelos, Lenguaje Específico de Dominio y Generación Automática de Código. La última parte de esta sección trata sobre la visualización de datos en este dominio.

La Sección 4 (pág. 23) describe la propuesta planteada en el presente informe, mostrando cuáles son las razones que justifican su realización. También, se describe brevemente a los stakeholders que se benefician y cómo esta propuesta atiende a las necesidades específicas de ellos, en el contexto de la visualización de datos en el monitoreo estructural de puentes.

La Sección 5 (pág. 31) pone en contexto la necesidad de generar un lenguaje específico de dominio (DSL), que describe la relación entre las visualizaciones y los sensores que se encuentran en la estructura del puente. Se indica en esta sección qué aspectos fueron considerados en la creación del DSL y, finalmente, se muestran vistas del Meta-modelo utilizado para la creación del DSL.

La Sección 6 (pág. 43) muestra la Estrategia de Generación Automática de Código utilizada. Aquí se presentan las decisiones tomadas para la elección de los lenguajes de programación seleccionados para la transformación, la elección de las bibliotecas para la visualización, entre otros. También se describen las principales ventajas y desventajas de los lenguajes elegidos. Se abordará también la estrategia utilizada para la transformación de Modelo a Texto y algunas de las reglas implementadas.

La Sección 7 (pág. 54) muestra los resultados de la implementación de cada uno de los requerimientos indicados, mediante una aplicación realizada a partir del Meta-modelo propuesto para el DSL y la estrategia de generación automática de código presentada en este trabajo.

La Sección 8 (pág. 59) contiene la evaluación realizada en términos de cobertura de alternativas de visualización, usabilidad y comparativa con el enfoque tradicional.

La Sección 9 (pág. 80) concluye el presente trabajo mostrando el cumplimiento de los objetivos planteados en la Sección 1. La sección también trata sobre las ventajas de utilizar esta aproximación, las dificultades encontradas y eventuales mejoras que pueden ser realizadas en este trabajo que podrían llevar a trabajos futuros en base a esta propuesta.

Finalmente la Sección 11 (pág. 84) contiene las referencias bibliográficas utilizadas en este trabajo.

Este proyecto ha recibido financiamiento y se enmarca dentro del proyecto FONDEF *Plataforma de Monitoreo Estructural de Puentes IT18I0112F*.

2. TRABAJO RELACIONADO

Para esta propuesta, el análisis del trabajo relacionado se ha dividido en dos secciones. La primera sección considera aquellos trabajos que describen sistemas de Monitoreo de Salud Estructural. La segunda sección describe aquellos trabajos que han utilizado desarrollo de software dirigido por modelos para generar alternativas de visualización.

Se han analizado los aspectos más relevantes para la presente propuesta en términos de visualización/interfaz de usuario, usabilidad y desarrollo.

En cada sección, los trabajos relacionados están organizados por año en orden ascendente.

2.1. Sistemas de Monitoreo Estructural

En el artículo de H. Chung, T. Enomoto, M. Shinozuka et al [2] se muestra un sistema de monitoreo de salud estructural mediante el uso de sensores inalámbricos que incluye una alternativa gráfica en tiempo real. Este sistema recibe los datos desde una unidad de adquisición, para luego enviarlos a un panel de visualización personalizado construido en C++. Este trabajo centra sus esfuerzos de visualización en alternativas gráficas en tiempo real. El análisis de datos históricos es realizado con la ayuda de un software externo. La Figura 1 muestra la gráfica en tiempo real que genera el sistema, con las opciones de Iniciar, Detener y Data Log. El gráfico muestra un alto contraste que facilita la visualización, aunque no se aprecia ninguna opción de filtro en el gráfico, comparación con otras gráficas, u otro control dinámico.

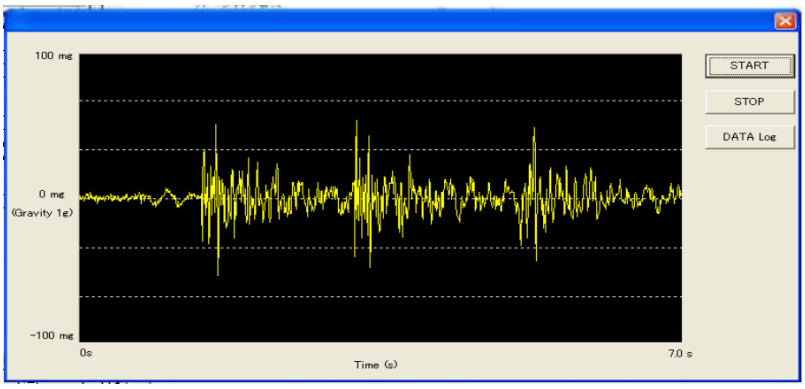


Figura 1: Muestra la medición en tiempo real para un sensor de acceleración. *Fuente: H. Chung, T. Enomoto, M. Shinozuka et al [2]*.

El trabajo de S. Masri, L. Sheng, J. Caffrey et al. [3] muestra un sistema de monitoreo estructural en tiempo real para sistemas de infraestructura civil. Este sistema posee una arquitectura que permite adquirir datos desde múltiples canales, monitoreo y acondicionamiento de los datos, para finalmente distribuirlos en tiempo real sobre Internet. Está diseñado alrededor de una interfaz publicación-suscriptor, donde el sistema envía datos de diferentes tipos (publicaciones) y los clientes (subscriptores) solicitan un tipo específico de datos. Entre los aspectos destacables en la visualización, se encuentra el uso de distintos colores para indicar las coordenadas xyz del eje de medición, como muestra la Figura 2a. Aunque existe preocupación en aspectos de usabilidad, no existen etiquetas que indiquen qué significa cada color, con lo que el usuario debe conocer de antemano la nomenclatura. Además, el sistema permite al usuario de la aplicación utilizar distintos tipos de filtros (integración, global y disparo), como muestra la Figura 2a sobre el gráfico, y permite correlación cruzada entre canales. Como desventaja, se menciona que este sistema presta atención solo a aspectos de visualización en tiempo real y no a datos históricos. El sistema propone una interfaz dinámica para el monitoreo con un filtro aplicado para escalar los datos. Luego, es posible añadir a los datos brutos distintos filtros, por ejemplo, filtros para eliminar frecuencias altas a cada canal o sensor específico. Asimismo, la interfaz permite retener a los clientes en caso de desconexión y posee un sistema de notificación por correo electrónico para dar aviso a los usuarios de algún evento.

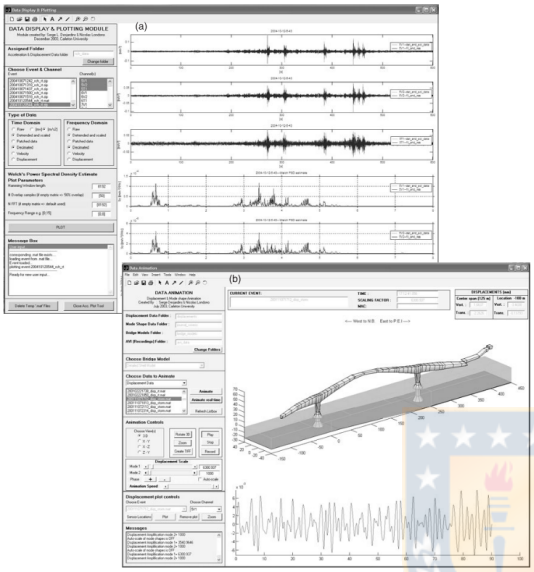
(a) Visualización de los datos brutos para los sensores.

(b) Visualización de los distintos filtros aplicables.

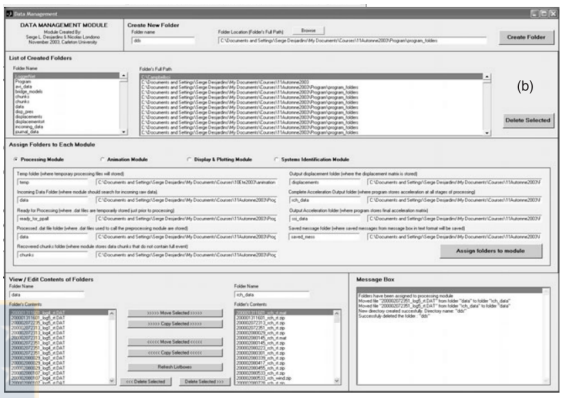
Figura 2: Capturas de dos partes que componen el sistema de visualización. *Fuente: S. Masri, L. Sheng, J. Caffrey et al. [3]*.

Se propone, en el artículo de S. Desjardins, N. Londoño, D. Lau et al. [4], un sistema en tiempo real de procesamiento de datos, análisis y visualización para monitoreo estructural. En dicha investigación, se expone un sistema integrado mediante la realización de una interfaz gráfica de usuario que soporta la visualización a través de módulos extensibles. Dentro de los módulos mencionados en la Figura 3a se encuentra una herramienta útil para el análisis que permite graficar datos; por ejemplo, posibilita graficar distintos canales en una misma ventana y, además, añade un módulo para la visualización de un modelo tridimensional de la estructura, y otro para la visualización de los datos que permitan el análisis de grandes conjuntos de estos. La arquitectura propuesta soporta la visualización de datos en tiempo real sujeta a la velocidad de la red. Se destaca que, además de los aspectos de visualización en tiempo real, posea soporte para el análisis de datos. Otro aspecto notable es el diseño modular presentado en este artículo y el soporte para datos históricos. Sin embargo, la rigidez en la interfaz presentada al usuario dificulta su uso. Los módulos presentados muestran demasiada

información en una misma ventana (por ejemplo, el modulo de manejo de archivo, en la Figura 3b, muestra 5 listas en una misma ventana), esto es contrario a una de las heurísticas de usabilidad de Jakob Nielsen [12], la de mantener un diseño estético y minimalista.



(a) Módulo de visualización de datos y Módulo de Animación.



(b) Módulo de Administración de Archivos.

Figura 3: Capturas de 3 módulos del sistema. *Fuente: S. Desjardins, N. Londoño, D. Lau et al. [4]*.

En el trabajo de S. Beskhyroun, L. D. Wegner, and B. F. Sparling [13], se propone un marco de trabajo de código abierto para SHM basado en una arquitectura orientada a servicios, donde la implementación de servicios es separada en tres módulos: Foundation Services (recolección de datos de sensores, comunicación, marcas de tiempo, otros), Application Services (Algoritmos para SHM) y Tools and Utilites (reinicios, cambios de canal, otros). Esta arquitectura se orienta a reducir la complejidad de las redes de sensores inalámbricos. Sin embargo, solo considera aspectos de la lógica en su arquitectura y no menciona o hace referencia a la visualización, es decir, no hay referencia a la forma en que los modelos planteados pueden ser representados o como estos interactúan con usuarios del sistema.

En el artículo de X. Hu, B. Wang, and H. Ji [14], se propone una arquitectura de software para SHM en Redes de Sensores inalámbricos personalizables y de bajo consumo. Como alternativa de visualización, se utiliza software propietario

ANSYS (para la estructura 3D). Una de las principales desventajas de este sistema es la utilización de un software externo para la visualización. Esto restringe su utilización, debido a que las interacciones deben ajustarse a interfaces predefinidas, cuya implementación es cerrada y difícilmente adaptable a necesidades particulares de análisis.

2.1.1. Comparativa con Software de Monitoreo Estructural

Existen algunos software disponibles para el monitoreo estructural, estas soluciones son privativas y ofrecen alternativas de visualización.

Uno de los software utilizados para comparar con este proycto es CATMAN [1], este software es parte del sistema HBM que ofrece una solución privativa para todas las etapas del monitoreo estructural. Dentro de las alternativas de visualización observadas en este software está la visualización de gráficos de línea, gráficos de barra , espectrogramas. Esto sujeto a la utilización de algunos DAC específicos. También cuenta con una separación en módulos de sus datos, lo que se traduce en distintas ventanas para la visualización.

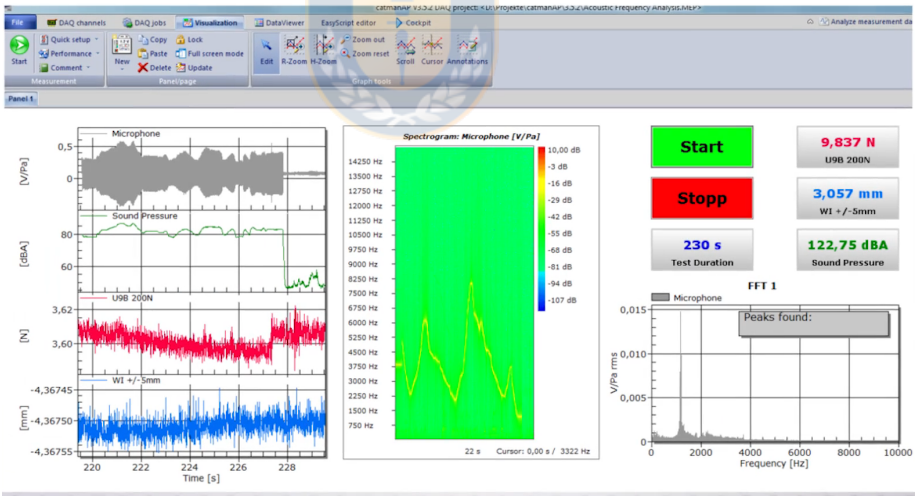


Figura 4: Captura de Software CATMAN. *Fuente:Sitio web CATMAN* .

ARTeMIS [2] es otro software dedicado al Análisis de Monitoreo estructural, esta herramienta privativa ofrece variadas alternativas de visualización y análisis de datos. Entre las alternativas observadas cuenta con gráficos de línea, gráficos de barra 2d y 3d, gráficos de Radar entre otros.

[1] https://www.hbm.com/en/2290/catman-data-acquisition-software/
[2] https://svibs.com/

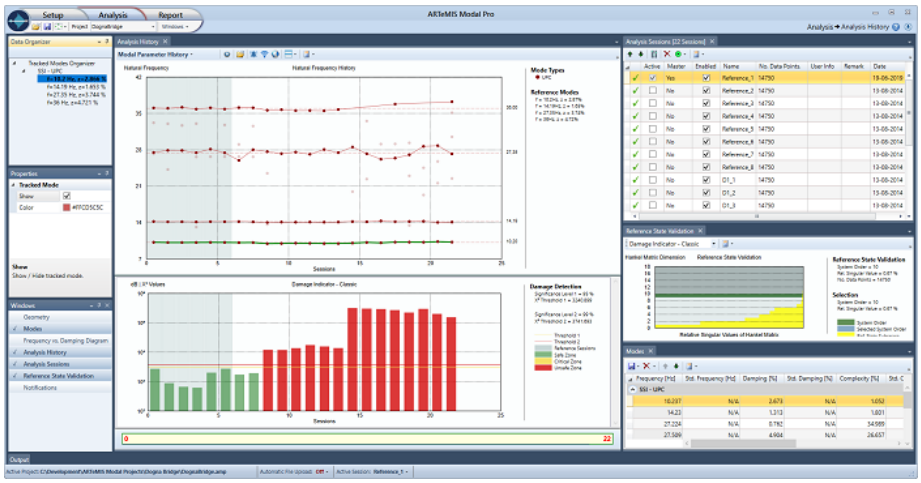


Figura 5: Captura de Software ARTeMIS. *Fuente: Sitio web ARTeMIS*[3].

Pulse [4] es un software que también se especializa en el análisis de datos para SHM. Entre las alternativas de visualización observadas está gráficos de línea, gráficos de barra, gráficos de calor 3d. Ofrece una vista modular para las visualizaciones.

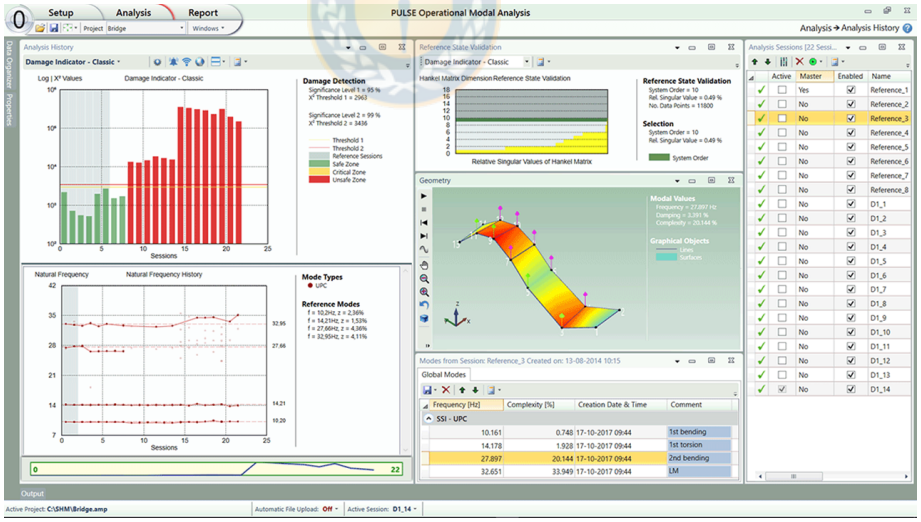


Figura 6: Captura de Software Pulse. *Fuente: Sitio web Pulse*.

Los tres sistemas antes mencionados son soluciones privativas para el Monitoreo Estructural, en comparación con la solución propuesta en este proyecto, las alternativas de visualización de estos software ofrecen una mayor variedad de

[3] https://svibs.com/applications/structural-health-monitoring/

[4] https://www.bksv.com/en/products/Analysis-software/structural-dynamics-software/modal-measurements-and-analysis/Structural-health-monitoring-BZ-8550-BZ-8554

servicios (aparte de las visualizaciones) que no están contempladas en este proyecto. Las herramientas presentadas requieren estar conectadas directamente al DAC para recibir los datos del puente u otro sistema de monitoreo. La solución aquí planteada esta construida para tomar datos históricos que no necesitan estar en un DAC, pueden ser obtenidas directamente de una base de datos o de un archivo (Se han probado los formatos CSV, MAT y base de datos timescaledb). Las aplicaciones mostradas, deben ser instaladas en un equipo para poder acceder a todas sus funcionalidades. La aplicación construida en este proyecto genera visualizaciones que son accesibles desde un navegador web, es decir, estas pueden ser ejecutadas en cualquier sistema como Windows Linux o Mac. Además de ser capaz de generar las visualizaciones, el sistema desarrollado es este proyecto también puede ser construido para funcionar en los sistemas recién mencionados. Adicionalmente la construcción mediante el Meta-modelo permite extender tanto las visualizaciones como los sensores desde los cuales se puede obtener la información.

Gempa [5] también ofrece un software para el Monitoreo Estructural de Puentes. La diferencia con las aplicaciones presentadas anteriormente es que esta aplicación esta basada en la web. Esta aplicación permite gráficos de líneas, gráficos de barra y separa cada visualización en módulos distintos.

Figura 7: Captura de Software Gempa. *Fuente: Sitio web Geampa*.

Esta aplicación, centrada en análisis en tiempo real, está construida bajo tecnología web pero no es posible obtener visualizaciones bajo demanda, como la solución presentada en este proyecto.

[5] `https://www.gempageoservices.com/solutions/structural-health-monitoring/`

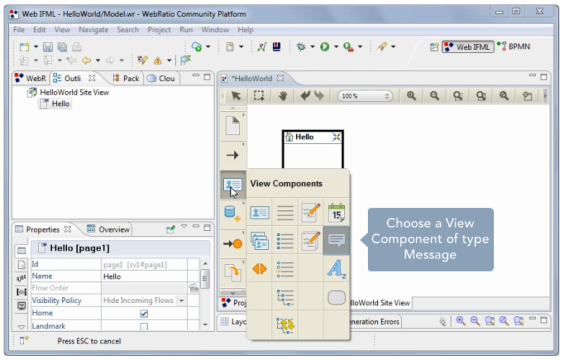
2.2. Visualización mediante Desarrollo Dirigido por Modelos

La idea de utilizar el Desarrollo de Software Dirigido por Modelos (MDSD, del inglés Model Driven Software Development) para generar alternativas de visualización no es nueva. Bull et al. [15] plantearon el uso de MDSD en la creación de visualizaciones, para así mejorar la eficiencia en la creación de vistas de un sistema. Los autores proponen una arquitectura de referencia para la Visualización Dirigida por Modelos, donde exista un Meta-modelo para representar el dominio de la aplicación y un Meta-modelo para cada uno de los visualizadores. Estos visualizadores serán los encargados de conformar cada una de las vistas del sistema software y, de esta manera, generar herramientas de visualización personalizables.

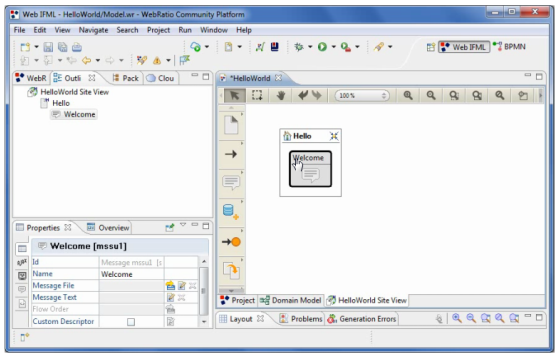
También se ha utilizado MDSD para manejar aspectos de interacción con el usuario. El artículo presentado por Marco Brambilla y Piero Fraternali [16] muestra como un enfoque dirigido por modelos puede ayudar a crear la lógica y el Front-End de una aplicación web. Esto mediante el uso de webRatio, una aplicación privativa que se basa en IFML (Interation Flow Modeling Language)[6], que es uno de los lenguajes estándar de modelados definidos por el OMG[7].

La Figura 8a muestra un ejemplo de uso de la herramienta webRatio. Se muestra el Web Model donde se pueden seleccionar componentes para ser agregados al contenedor Hello. En la Figura 8b, se ha agregado un mensaje al contenedor y finalmente en la Figura 8c se muestra la página web generada con la herramienta.

[6] https://www.ifml.org/
[7] https://www.omg.org/



(a) Se muestra la selección de componentes.



(b) Componente de mensaje en el contenedor.

(c) Página generada con el modelo.

Figura 8: Secuencia para la creación de una página web con un mensaje mediante Web Ratio. *Fuente: https://my.webratio.com/learn/learningobject/your-first-web-application-v-72*.

En el trabajo de Brambilla et al. [5] se propone un enfoque basado en modelos para el diseño de interfaces gráficas de usuarios para IoT, mediante IFML. En dicho artículo, se identificaron los componentes de IoT y fueron modelados mediante UML. La propuesta fue evaluada mediante tres casos reales: un sistema de automatización del hogar, un sistema de hornos inteligentes para la panadería y un sistema de gestión de impresoras industriales. Para llevarlo a cabo, se modelaron patrones de diseño comunes para generar interfaces gráficas para IoT. En la Figura 9 se muestra parte de la interfaz de usuario de la aplicación para automatizar el hogar.

Figura 9: Parte de la interfaz de usuario de la aplicación Smart-house. *Fuente: Brambilla et al. [5]*.

Por otro lado, Dibia et al. [17] no utiliza un desarrollo de software dirigido por modelos para la generación de herramientas de visualización, pero sí utiliza un enfoque basado en la generación automática. Esto lo realiza para un conjunto de datos dado, formulando la generación de visualización como un problema de traducción de lenguajes, donde transforman la visualización a un lenguaje declarativo.

2.3. Resumen

Por un lado, se han observados aspectos a mejorar dentro de los artículos consultados en la investigación del estado del arte. Dentro de estos aspectos se destacan los siguientes:

- Existen esfuerzos de visualización en Sistemas de Monitoreo de Salud Estructural con un fuerte enfoque al análisis en tiempo real de los datos. Aun-

que existe algún soporte al análisis de datos históricos, la mayoría de los trabajos mencionados utilizaron, para ello, el uso de herramientas complementarias (por ejemplo, Matlab).

- Los sistemas de visualización para Monitoreo de Salud estructural de puentes, utilizan como principal herramienta de análisis, gráficos de línea, área, dispersión e histogramas (esto se puede ver en [7]). En algunos casos específicos, se utilizó mapas de calor (Ejemplos en [18]).

- En la propuesta de webRatio se observa que aunque el DSL viene definido en este software, el usuario final de la herramienta también debe definir el modelo del dominio (que incluye, el manejo de base de datos). Esto se traduce en que el usuario debe tener un conocimiento de modelado, no bastando el DSL.

Por otro lado, se han observado algunas buenas prácticas y patrones de uso en la construcción de interfaces para la visualización de datos que han sido utilizadas en la realización de esta propuesta. Se destacan las siguientes:

- Los sistemas están compuestos por elementos que pueden ser caracterizados y definidos mediante un Meta-modelo, como los sensores instalados en la estructura o los gráficos generados para el análisis.

- Se observan elementos comunes que se visualizan en los sistemas como gráficos de línea por sensor o canal, filtros, correlación entre variables, entre otros (se puede observar esto en [3, 4, 19]). Estos elementos podrían ser generados de manera automática.

- Las propuestas orientadas al MDSD o la generación automática para la visualización a través de una Interfaz Gráfica de Usuario (GUI) apuntan al desarrollo con un enfoque basado en la web.

3. MARCO TEÓRICO

3.1. Monitoreo Estructural de Puentes

Como se indicó en la introducción, el monitoreo estructural es una disciplina que permite identificar el daño y prolongación de este en estructuras de ingeniería civil, aeroespacial y mecánica, donde el daño es definido como cambios en el material o estructuras geométricas [6]. Estos cambios en el material pueden ser detectados por diferentes métodos mencionados en la literatura como muestreo compresivo, enfoques de identificación de grietas [20], la inspección visual, cambios en la frecuencia u otros métodos donde se instalan múltiples sensores sobre la estructura. Usualmente se utilizan sensores piezo-eléctricos, fibra óptica u otros materiales que sirven para medir fenómenos físicos asociados al daño de la estructura [21]. Además, existen sensores inteligentes y más complejos para este propósito específico como los sensores *Fiber Bragg grating sensor* y *piezoelectric transducer [22]* usado por ejemplo, para medir tensión.

Los sistemas de monitoreo de salud estructural son sistemas complejos donde intervienen profesionales de distintas áreas [7]. En particular, en el monitoreo de salud estructural de puentes se instalan distintos sensores sobre la estructura, como acelerómetros, strain gauge, LDVT, entre otros. Estos sensores usualmente están conectados mediante una red inalámbrica (ejemplo de esto en [2,13,14,23]) que funciona mediante software libre (por ejemplo TinyOS), propietario (por ejemplo, National Instruments) o una combinación de ambos. Los datos adquiridos por los sensores son requeridos para ser procesados, realizar cálculos sobre estos y análisis de la estructura [22]. También se instalan estaciones meteorológicas en la estructura que permiten medir temperatura ambiental, velocidad del viento, entre otras variables [4, 24].

3.2. Desarrollo de Software Dirigido por Modelos

Tradicionalmente, el desarrollo de software se realiza considerando 4 etapas fundamentales de desarrollo, estas son especificación, diseño, construcción y validación [25]. Estas etapas son desarrolladas por diferentes metodologías de desarrollo tal como cascada, prototipo, espiral, RUP, entre otras. La etapa de construcción implica codificar la solución en algún lenguaje de programación y las etapas de especificación y diseño representan documentación importante para el mantenimiento del software. En el Desarrollo de Software Dirigido por Modelos, los modelos del sistema se establecen con suficiente detalle para que la implementación completa del sistema pueda generarse a partir de los modelos [26].

La aplicación de este enfoque se caracteriza por las etapas de Meta-modelado, Transformación del Modelo y una etapa de Generación Automática de Código.

En la etapa de Meta-modelado, el modelador se encarga de definir un Meta-modelo. El Meta-modelo, es un conjunto de elementos que son usados por modeladores para construir sus modelos [1], en otras palabras, un Meta-modelo es el modelo de un modelo. El Object Management Group (OMG) propone una jerarquía Meta-modelo de cuatro niveles que se explica en la Tabla 1, donde el Meta-modelo se encuentra en el nivel M2 y es el que se encarga de definir un lenguaje para especificar modelos. En el Meta-modelo se representan las reglas que definirán los modelos que serán generados para el dominio.

M3	Define un lenguaje para especificar un Meta-modelo
M2	Define un lenguaje para especificar modelos
M1	
M0	Contiene ejemplares de los elementos del modelo.

Tabla 1: Jerarquía de Modelos. *Fuente: Resumen de la tabla obtenida de [1].*

En la siguiente etapa se define un Lenguaje Específico de Dominio (DSL, del inglés Domain Specific Language), este lenguaje expresa la solución en el idioma y nivel de abstracción del experto en el dominio, esto permite que los expertos puedan entender por ellos mismos los programas basados en un DSL [27]. En la etapa de transformación del modelo, se toman uno o más modelos de entradas para producir uno o más modelos de salida [28]. Cuando se encuentra definido el Meta-modelo, es posible generar ejemplares del modelo que, mediante una herramienta de generación automática de código, puede ser transformado a código ejecutable. Una de las técnicas para realizar esta transformación es el uso de plantillas que permiten mediante el Meta-modelo (Template+Meta-model) generar código ejecutable a partir de los modelos generados en base al Meta-modelo. Un ejemplo de esto es Acceleo [8] que es una tecnología que permite generar plantillas de texto (o código fuente) mediante un modelo EMF [9] para, posteriormente, generar de forma automática texto o código en un lenguaje de programación.

3.3. Visualización de Datos

Los sistemas de monitoreo de salud estructural de puentes necesitan una arquitectura software capaz de soportar la adquisición, fusión y limpieza de datos,

[8] https://www.eclipse.org/acceleo/
[9] https://www.eclipse.org/modeling/emf/

características de extracción y condensación de la información, además de modelos estadísticos para el análisis de datos [29].

Dentro de los elementos que componen una arquitectura de software se encuentran las estructuras, comportamiento de funciones, interacciones, visualización, entre otros. La visualización define cómo los modelos son representados y cómo todos aquellos interesados en el sistema tendrán interacción con él [30].

El interés de este trabajo es, en particular, la visualización de datos. Para representar la información obtenida de sensores usualmente se utilizan distintos tipos de gráficos como gráficos de línea, área, histogramas, entre otros. Estos, permiten analizar la información obtenida desde los sensores. Como plantean Wang, L., Wang, G., & Alexander [31], la visualización de datos no solo es estática. Una visualización de datos también puede ser interactiva añadiendo elementos como zoom, foco, panorámica, entre otros.

Hasta hace poco tiempo, la mayoría de las herramientas disponibles limitan la visualización de datos a entornos científicos [31]. Dificultando la incorporación de estas visualizaciones a un sistema de monitoreo estructural o a un entorno web. Hoy existen distintas herramientas de visualización, aunque conllevan el aprendizaje de sus respectivas bibliotecas en conjunto con otras APIs para la manipulación de datos.

4. PRESENTACIÓN DE LA PROPUESTA

4.1. Justificación

Tradicionalmente, el monitoreo de puentes en Chile se realiza basado en las experiencias particulares de los especialistas, de una forma más reactiva que preventiva. El monitoreo se realiza frecuentemente con guías de **inspección visual** y con antecedentes de los estudios técnicos disponibles [32]. En Chile existen 14 puentes que se encuentran bajo un monitoreo especial debido al daño estructural. De estos puentes, 7 se encuentran bajo monitoreo permanente, mientras que los demás se encuentran sin uso, en reparación u otro [33]. De estos puentes, un caso emblemático es la caída del puente Cancura el 2018 que dejó una persona fallecida y 6 heridos [34]. Esto permite ejemplificar la importancia del monitoreo de salud estructural aplicado a los puentes por los daños ocasionados durante el colapso. Además, los puentes representan la continuidad espacial de un camino: al fallar un puente también se reduce la conectividad de la ciudad donde este puente está instalado.

4.1.1. Plataformas de Monitoreo Estructural

En la búsqueda de un modo de realizar el monitoreo de salud estructural de puentes de manera preventiva, surge la necesidad del desarrollo de una Plataforma de Monitoreo Estructural de Puentes. Estas plataformas permiten detectar el daño mediante sensores que son instalados en la estructuras. Para dar soporte a los requerimientos del monitoreo de salud estructural de puentes, es necesario definir una arquitectura de software para el sistema. Esta debe representar la estructura general del sistema y sus interacciones [35]. Dentro de los elementos que componen una arquitectura se encuentran las estructuras, comportamiento de funciones, interacciones, visualización, entre otros. De los elementos mencionados, la visualización es la que genera un mayor interés en esta propuesta, porque, como en todo sistema Big Data, la visualización es efectiva para presentar información esencial en grandes cantidades de datos y también puede conducir a análisis más complejos [36]. Desde el punto de vista de la arquitectura de software, la visualización define cómo los modelos son representados y cómo todos aquellos interesados en el sistema tendrán interacción con él [30].

4.1.2. Captura y visualización de Datos

La gran cantidad de datos que son capturados desde una infraestructura de sensores lleva a basar el análisis y visualización de datos en SHM en tecnologías

Big Data (grandes volúmenes de datos). Hasta hace poco tiempo, la mayoría de las herramientas Big Data disponibles limitan la visualización de datos a entornos científicos [31]. Esto dificulta incorporarlos a otras plataformas (de monitoreo estructural o plataformas web) para mayor accesibilidad.

Usualmente las herramientas de visualización utilizadas son gráficos de línea, área, histogramas, dispersión entre otros. Donde es deseable una visualización dinámica que permita acciones como zoom, foco, ajustar intervalos de datos en tiempo de ejecución, entre otros.

Muchas de las herramientas Big Data disponibles para científicos no permiten trabajar con tecnologías basadas en la web. En los sistemas de monitoreo estructural estos desafíos se observan en que los profesionales deben trabajar con herramientas externas para la visualización de datos (Matlab/Octave, Python, R) en las que se debe codificar cada gráfico para ser visualizado. Algunas de estas herramientas poseen poco soporte para generar informes o visualización web. Esto implica recurrir a otros profesionales expertos en visualización que manejen lenguajes para este fin (html, javascript u otro), lo que aumenta la complejidad del sistema.

Con el aumento reciente en el interés y demanda por servicios orientados al análisis de grandes volúmenes de datos, han surgido entornos de desarrollo y ejecución de visualizaciones de datos orientados a una audiencia más amplia. Aun así, el nivel requerido de experiencia en lenguajes de programación y bibliotecas de manipulación y visualización de datos es aún alto.

4.2. Propuesta

Esta propuesta, presenta un enfoque dirigido por modelos para reducir la complejidad en el desarrollo de alternativas de visualización para el Monitoreo Estructural de Puentes. La propuesta se implementa en una herramienta llamada *vis4bridge* que se describirá en los siguientes capítulos. El enfoque considera la generación automática del código necesario, a partir de un Meta-modelo. El Meta-modelo, es el encargado de fijar las reglas que definen un lenguaje específico de dominio (DSL) que describe y asocia objetivos de exploración de datos con alternativas de visualización, en un alto nivel de abstracción. A partir de este Meta-modelo y mediante el uso de una herramienta de modelado gráfico, los usuarios podrán generar automáticamente el código de la visualización modelada, pudiendo utilizar directamente dicho código en plataformas destino. Así, el experto del dominio, prescindiendo de conocimientos avanzados de programación, puede generar rápidamente las visualizaciones que mejor respondan a sus propios requerimien-

tos de análisis de los datos, para la toma de decisiones oportunas. Además, el Meta-modelo separa de forma modular sensores y visualizaciones, lo que permite, reutilizar el Meta-modelo y todos sus componentes en visualizaciones pertenecientes a otros dominios.

4.3. Principales Stakeholders

Existen distintos actores con el potencial de beneficiarse con la realización de esta propuesta. Dentro de estos actores están:

Expertos en Monitoreo Estructural:

Se refiere a todos aquellos profesionales del área de Ingeniería Civil u otras áreas relacionadas dedicados al monitoreo de salud estructural de puentes. Con la realización de este proyecto, estos profesionales, prescindiendo de conocimientos avanzados de programación, podrán generar sus propias visualizaciones de datos históricos recientes, a partir de ejemplares del Meta-modelo.

Autoridades:

Se refiere a todas aquellas autoridades que, en base a la información suministrada por los expertos en monitoreo estructural, tienen la potestad de tomar decisiones referentes al uso y reparación de puentes. Con la realización de este proyecto, podrán obtener de manera oportuna la información suministrada por los expertos de monitoreo de salud estructural para una mejor toma de decisiones.

Desarrolladores:

Se refiere a aquellas personas que desarrollan software de visualización de datos para monitoreo estructural de puentes. Mediante ejemplares del modelo, los desarrolladores obtendrán código generado de forma automática, evitando de esta manera, la introducción de errores al programar. Además, podrán manipular el código obtenido por defecto y utilizarlo en la misma plataforma de monitoreo o ser adaptado según sus necesidades. Por otro lado, la independencia del Meta-modelo con el lenguaje de salida permitirá soporte a varios lenguajes.

4.4. Propuesta arquitectónica

La propuesta arquitectónica será presentada mediante los tres primeros niveles de un modelo C4 [10]. El cuarto nivel es el código y no será presentado en esta sección, en su lugar, se mostrará el Meta-modelo y la organización de paquetes en la Sección 5.

En el primer nivel, se muestra un diagrama de contexto(ver Figura 10). Aquí cada uno de los stakeholders interactúa con el sistema. La autoridad (Authority) usa el sistema para la toma de decisiones, el experto en monitoreo (Structural health monitoring expert) obtiene las visualizaciones del sistema haciendo uso de la herramienta gráfica *vis4bridge*, mientras que el desarrollador (Developer) utiliza la herramienta gráfica para obtener el código fuente y adaptarlo a sus necesidades. Por otro lado la plataforma de monitoreo (Structural Monitoring Platform) provee la información de sensores del puente.

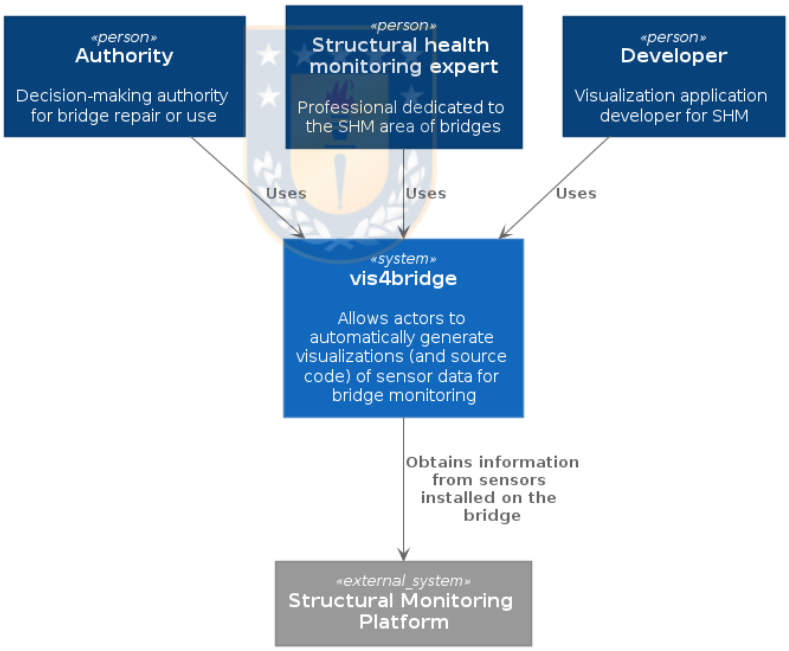


Figura 10: Diagrama de Contexto del Modelo C4 para la aplicación vis4bridge. *Fuente: Creación propia*.

En el segundo nivel, se muestra un diagrama de Contenedores (ver Figura 11). Este nivel, es una vista ampliada del sistema *vis4bridge* mostrado en el diagrama de contexto (ver Figura 10).

[10] `https://c4model.com/`

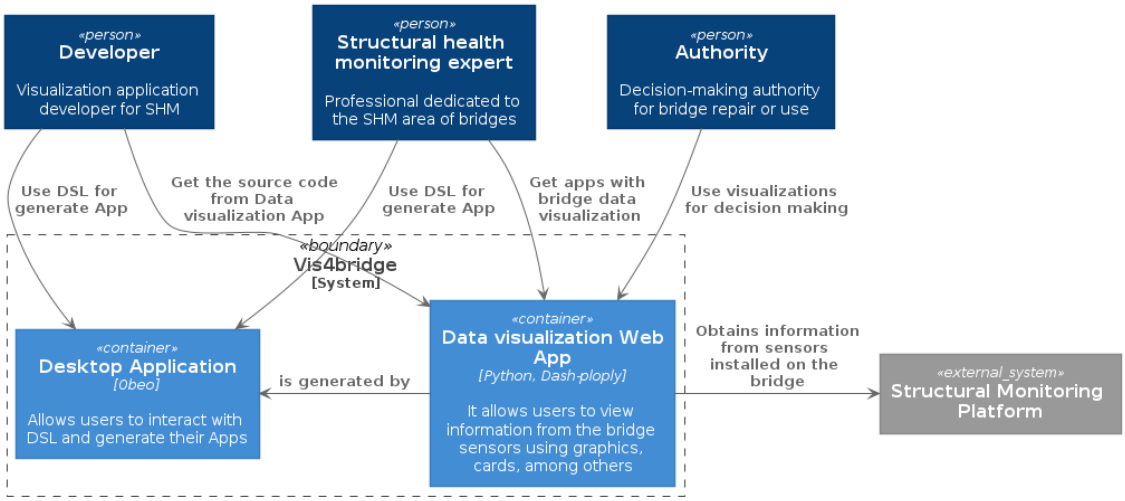


Figura 11: Diagrama de Contenedores del Modelo C4 para la aplicación *vis4bridge* (Utilizando Python). *Fuente: Creación propia*.

El Experto en Monitoreo de Salud Estructural (Structural health monitoring expert) usa la aplicación de escritorio *vis4bridge* (Desktop Application) para modelar una visualización. Una vez realizado el modelo, el Experto en Monitoreo de Salud Estructural genera una aplicación web (Data visualization Web App) con las visualizaciones modeladas.

Las visualizaciones generadas por el Experto en Monitoreo de Salud Estructural son usadas por la Autoridad (Authority) para la toma de decisiones.

El Desarrollador (Developer) usa la aplicación de escritorio *vis4bridge* (Desktop Application) para modelar una visualización. Una vez realizado el modelo, el Desarrollador genera una aplicación web (Data visualization Web App). El Desarrollador tiene acceso al código fuente de la aplicación para utilizarla o modificarla.

En todos los casos, la información de los sensores, es obtenida desde un sistema externo (Structural Monitoring Platform).

La aplicación de escritorio (Desktop Application) puede generar visualizaciones independiente del lenguaje de salida utilizado. Para demostrar esto, se ha construido una segunda transformación (que se explica con más detalle en la Sección 6) en el desarrollo de la aplicación. Esto solo cambia la tecnología utilizada para construir el contenedor Data visualization Web App como lo muestra la Figura 12.

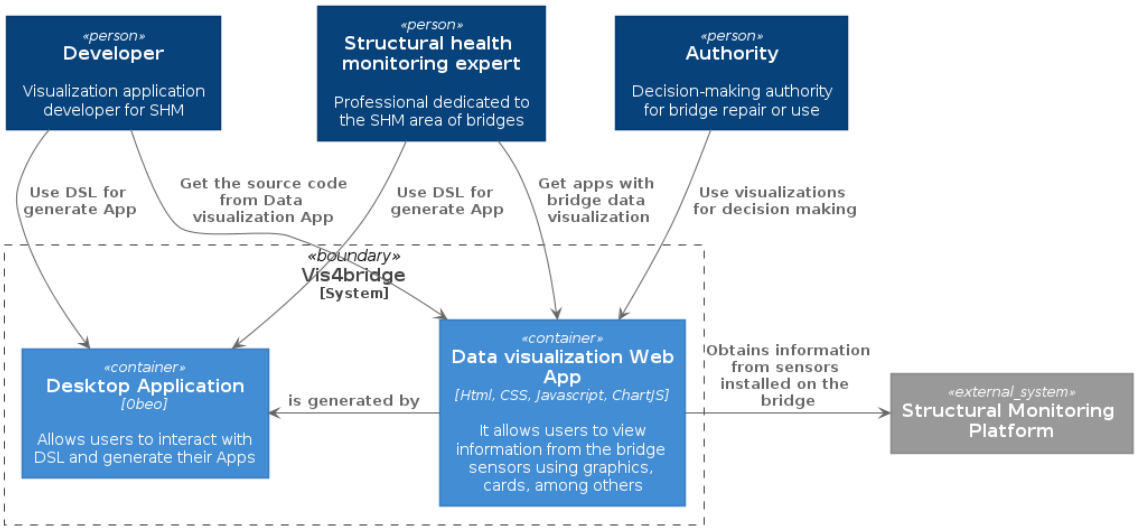


Figura 12: Diagrama de Contenedores del Modelo C4 para la aplicación *vis4bridge* (Utilizando Javascript). *Fuente: Creación propia*.

En el tercer nivel, se muestra un diagrama de Componentes (ver Figura 13 y Figura 14), este diagrama muestra los mismos componentes que el segundo nivel, pero ampliando el nivel de detalle.

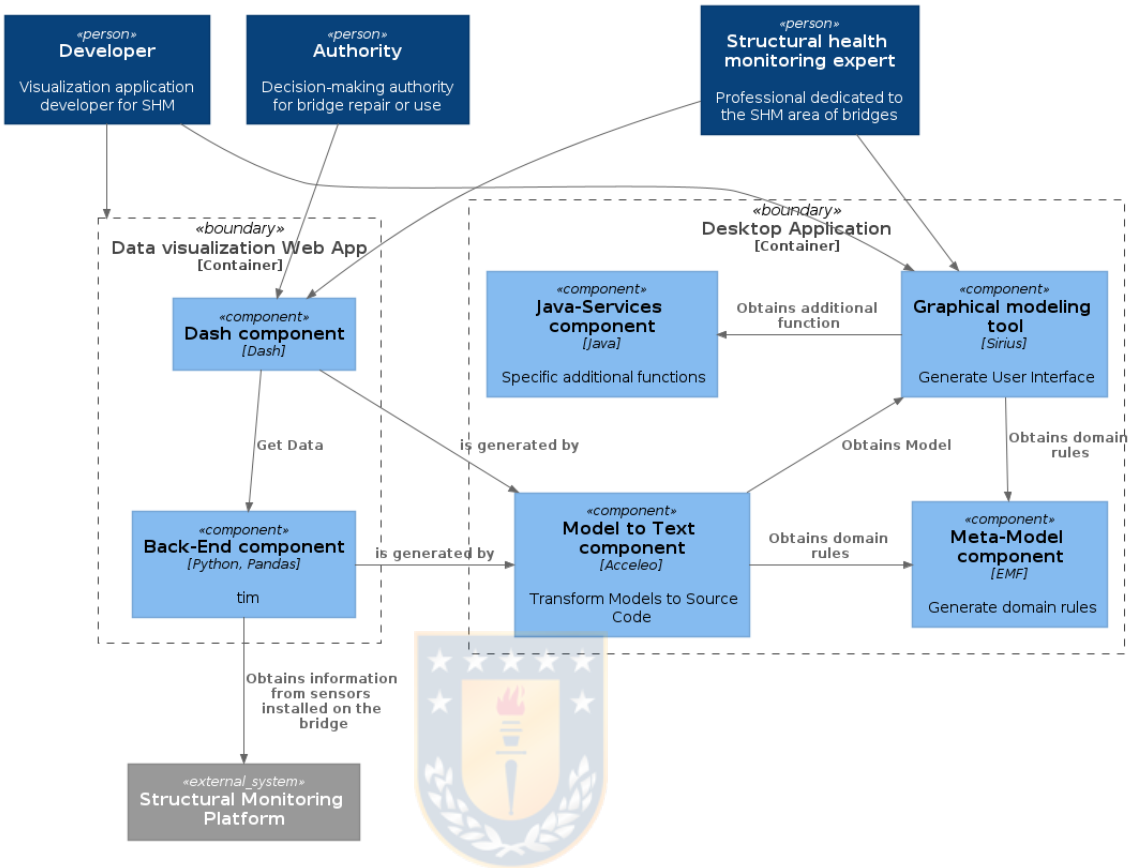


Figura 13: Diagrama de Componentes del Modelo C4 para la aplicación *vis4bridge*. *Fuente: Creación propia*.

Las interacciones del Experto en Monitoreo de Salud Estructural, la Autoridad y el Desarrollador son las mismas que en el diagrama anterior. Para explicar los flujos internos, se detallan las interacciones realizadas por el Desarrollador.

El Desarrollador se conecta a la aplicación de escritorio mediante el componente Graphical modeling tool creado con Sirius. Esta interfaz, permite al Desarrollador crear ejemplares del Meta-modelo. Para esto, obtiene las reglas del dominio desde el componente Meta-modelo (Meta-Model component). El componente servicios de Java (Java-Service component), proporciona funciones para establecer atributos del modelo de forma automática. Una vez generado el modelo, el Desarrollador puede transformar el Modelo al lenguaje de programación deseado mediante plantillas escritas con Acceleo a través del componente Modelo a Texto (Model to Text component).

Cuando la transformación se encuentre finalizada, el Desarrollador podrá interactuar con la visualización creada o directamente con el código fuente. Esta inter-

acción dependerá de la tecnología utilizada para generar las visualizaciones.

En el caso de generar la visualización en Python, el Desarrollador podrá interactuar con los gráficos creados mediante el componente Dash component que genera los gráficos y elementos para interactuar con ellos (barras de desplazamiento, selector de sensores, entre otros). Este componente, obtiene los datos procesados desde el componente Back-End component que se conecta al sistema externo Structural Monitoring Platform para obtener los datos. Adicionalmente el desarrollador podrá modificar estos componentes directamente desde el código fuente.

En el caso de generar la visualización en Javascript, el Desarrollador podrá interactuar con los gráficos creados mediante el componente Visualization que genera los gráficos, elementos para interactuar con ellos (barras de desplazamiento, selector de sensores, entre otros) y define los estilos aplicados a los elementos de la visualización. Este componente, se conecta a Main component, que obtiene los datos desde el sistema externo Structural Monitoring Platform. El procesamiento de los datos y las funciones que generan los gráficos se obtienen desde el componente Chart lib. Nuevamente, en este caso, el desarrollador podrá modificar estos componentes directamente desde el código fuente.

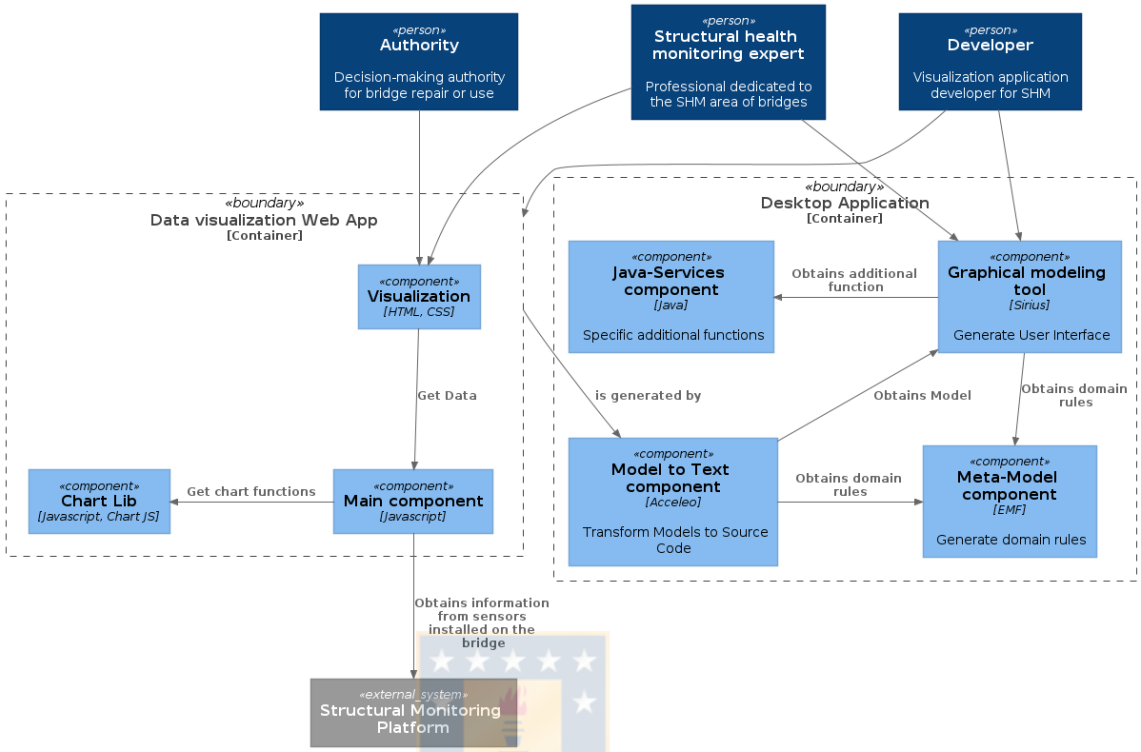


Figura 14: Diagrama de Componentes del Modelo C4 para la aplicación *vis4bridge*. *Fuente: Creación propia*.

5. DEFINICIÓN DEL LENGUAJE ESPECÍFICO DE DOMINIO

5.1. Descripción del Lenguaje de Modelado

Un lenguaje específico de dominio se puede definir como un lenguaje que permite, mediante notaciones y abstracciones apropiadas, expresar y generalmente restringir el problema a un dominio particular [27]. En el caso de los lenguajes de modelados, estos están definidos mediante una sintaxis abstracta que representa las abstracciones y relaciones del dominio, una sintaxis concreta que representa la forma en que los usuarios aprenderán y usarán el lenguaje y una semántica que revela el significado de las expresiones [37].

Para el caso particular de esta propuesta, la sintaxis abstracta es implementada mediante el Meta-modelo que nace del análisis del dominio (sub-sección 5.2) y es mostrado en la sub-sección 5.4. La sintaxis concreta puede ser representada en forma textual o gráfica, para esta propuesta se utilizó una representación gráfica de los elementos que componen el dominio, esta representación es mostrada en la sub-sección 5.5. La semántica se ven representada mediante las transformaciones que se generan al lenguaje de destino, estas transformaciones se muestran en la sección 6. En la Figura 15 se describe, mediante un diagrama de clases, el lenguaje de modelado para esta propuesta.

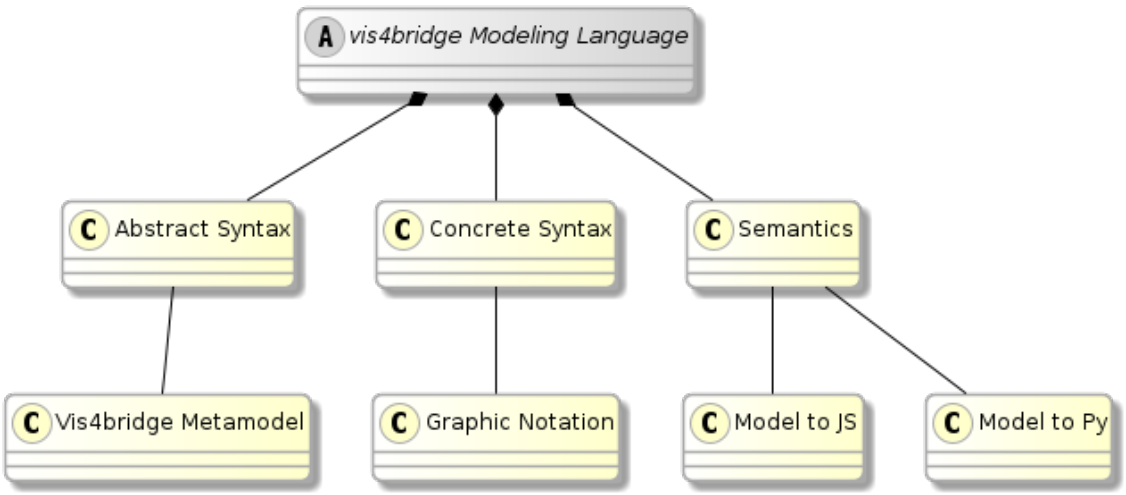


Figura 15: Descripción del Lenguaje de Modelado. *Fuente: creación propia*

5.2. Análisis del Dominio

Como se observó en la sección 4, uno de los elementos centrales que compone esta arquitectura (ver 13) es el Lenguaje Específico de Dominio (DSL, del inglés Domain Specific Language). Para describir este lenguaje se utiliza un Meta-modelo que representa las abstracciones del dominio.

Para encontrar las abstracciones es necesario realizar un análisis del dominio. En este caso, se realizó un análisis mediante revisión de literatura y con expertos en el dominio. A partir del análisis, se identificaron dos grandes grupos de abstracciones. Por un lado, los sensores, que son la fuente de información para la visualización de datos. Por otro lado, los gráficos, que permiten representar la información entregada por los sensores. Existen múltiples tipos de sensores, los cuales, entregan información sobre el estado del puente.

Luego de realizado el análisis del dominio, las relaciones encontradas entre los elementos se definen mediante un Meta-modelo. Este Meta-modelo, es la base para generar el Lenguaje Específico de Dominio. Luego de definir el DSL, mediante una técnica de generación automática de código, se podrán generar las visualizaciones para el monitoreo de salud estructural de puentes.

5.3. Modelo de Características

Inicialmente el Meta-modelo fue generado a partir de las abstracciones de sensor (sensor) y gráficos (graph). Adicionalmente, para la generación de cada visualización se agregó el elemento vista (view), donde las vistas están compuestas de toda la información visual que será mostrada al usuario final, es decir, cada vista estará compuesta por gráficos, tarjetas, filtros o cualquier otra información que sea de interés para los expertos en el dominio. La generación de vistas en el desarrollo de software dirigido por modelos no es nueva, R. Bull, M. Storey et al. [15] proponen una arquitectura que soporta visualización, en este trabajo se ha utilizado parte de esta metodología, pero con la diferencia que las vistas en este trabajo representan la agrupación de múltiples elementos de la visualización y cada ejemplar del meta-modelo puede contener una o más vistas. Además, aquí se han manejado los elementos que no pertenecen a la vista dentro del modelo en clases específicas para ellos.

Antes de generar el Meta-modelo, los componentes que constituyen el modelo fueron formulados de manera simplificada mediante un modelo de características (ver Figura 16).

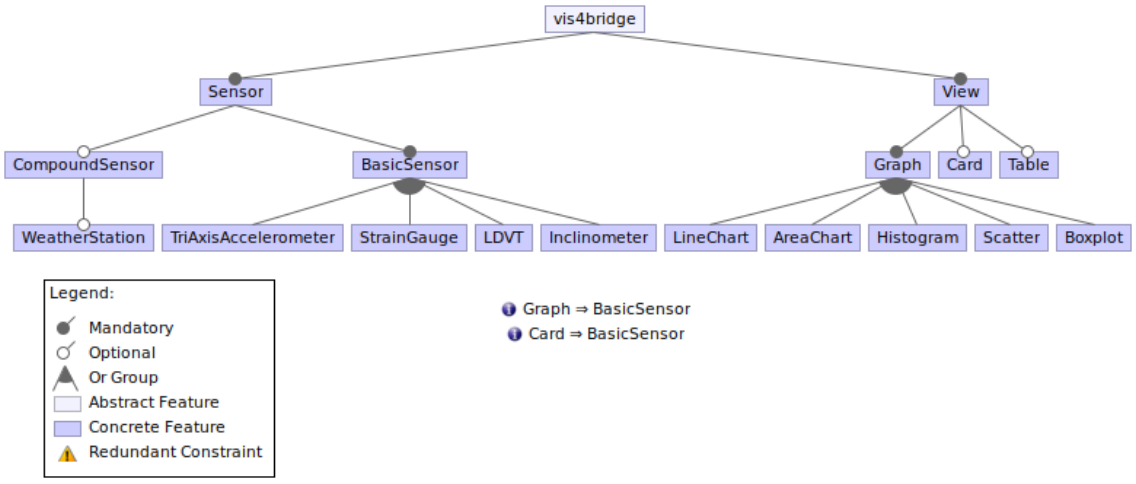


Figura 16: Modelo de características. *Fuente: creación propia*

A partir del modelo de características basado en el análisis del dominio, se obtuvieron las principales abstracciones que representan el problema. Utilizando la información del modelo de características fue posible generar un Meta-modelo que representará el problema de la generación de visualización de datos para el monitoreo estructural de puentes.

5.4. Presentación del Meta-modelo

El modelo de características (ver Figura 16), es usado como base para la construcción del Meta-modelo. En esta sección se presenta el Meta-modelo (ver Figura 17), el cual, describe el DSL para la visualización de datos en monitoreo estructural de puentes. El Meta-modelo fue construido utilizando un Diagrama de Clases de UML mediante la herramienta ecore[11].

En esta sección, se presenta la versión completa del Meta-modelo (que también encontrará en el Anexo A) y el detalle de sus componentes principales.

[11] `https://www.eclipse.org/ecoretools/`

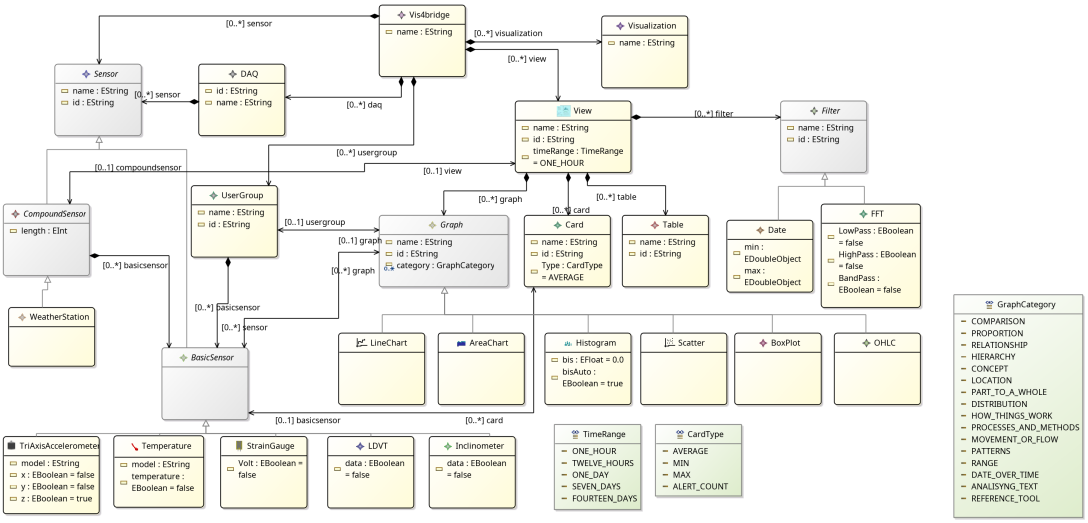


Figura 17: Meta-modelo generado con ecore tool. *Fuente: Creación propia*.

5.4.1. Clase principal

Cada una de las clases generadas en el Meta-modelo representa las características que se desean representar en el DSL. Inicialmente se encontraron dos grandes abstracciones que componen el diagrama, estas fueron representadas en la clase Sensor y la clase View. Adicionalmente se añadió la clase DAQ que representa un convertidor analógico digital al que se conectan los sensores en un puente y UserGroup que representa un grupo de sensores definido por el usuario (ver Figura 18).

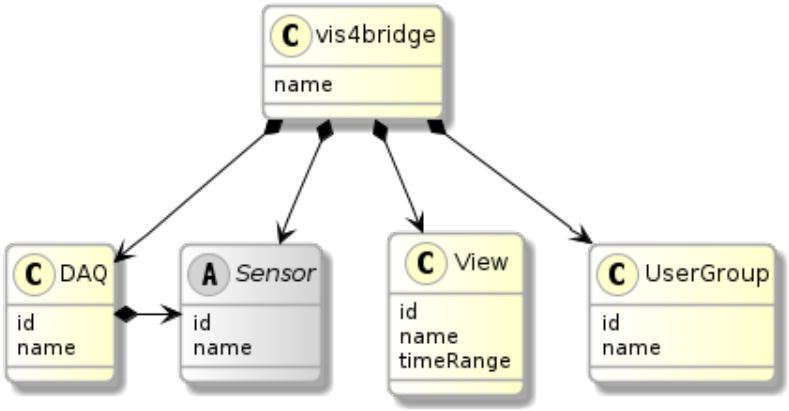


Figura 18: vista principal del Meta-modelo. *Fuente: Creación propia*

5.4.2. Sensores

Existen distintos tipos de sensores que se utilizan en puentes. Algunos sensores entregan información especializada como los acelerómetros o strain gauge, otros sensores están compuestos de sensores básicos y pueden entregar información de más de una variable del puente u otros parámetros ambientales. Para representar esto se generaron las clases BasicSensor y CompoundSensor(ver Figura 19). La clase BasicSensor representa los sensores básicos, CompoundSensor representa los sensores más complejos que pueden a su vez estar compuestos de otros sensores.

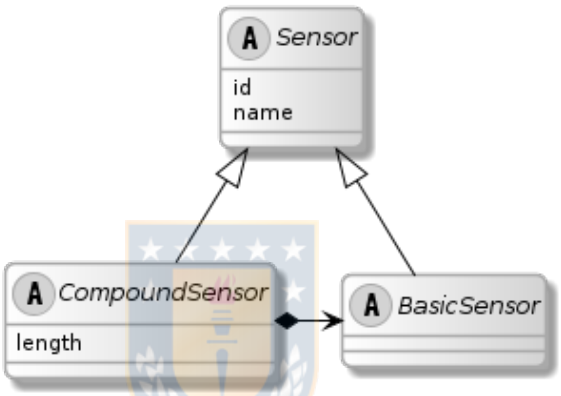
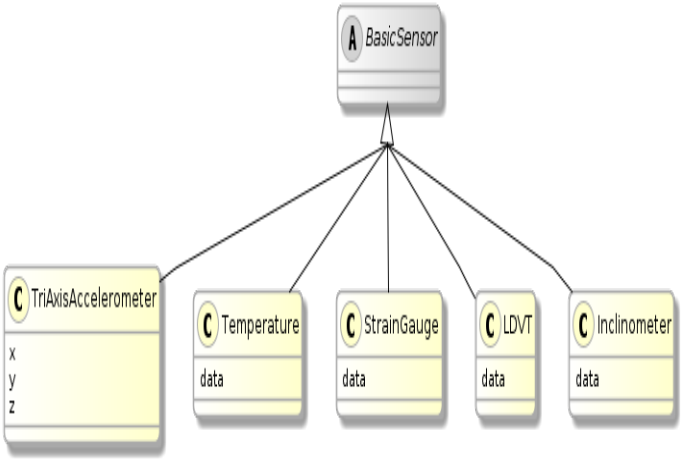


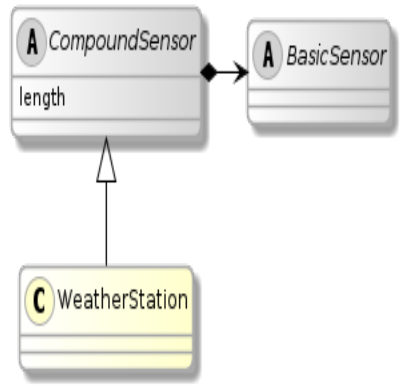
Figura 19: vista clase Sensor del Meta-modelo. *Fuente: Creación propia*.

Los sensores básicos y sensores compuestos heredan las características de BasicSensor y CompoundSensor respectivamente (ver Figura 20). Estos, al estar representados individualmente, facilitan la agregación de sensores en el futuro.

En las Figuras 20a y 20b se muestran las clases que representan los sensores.



(a) Vista de la clase BasicSensor del Meta-modelo.



(b) Vista de la clase CompoundSensor del Meta-modelo.

Figura 20: vista de sensores del Meta-modelo. *Fuente: Diseño Propio*.

5.4.3. Vistas

Las vistas representan la agrupación de elementos que serán mostrados en la aplicación (ver Figura 21). Cada vista puede estar compuesta por gráficos, tarjetas o tablas. Adicionalmente se pueden aplicar filtros sobre los elementos en la vista. Estos filtros se representaron mediante la clase Filter.

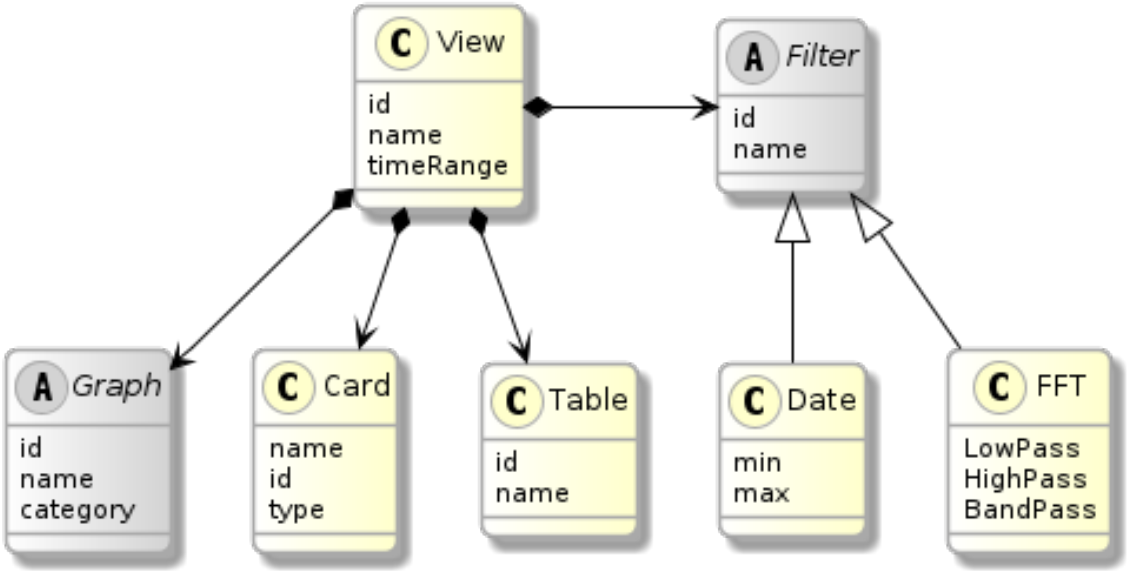


Figura 21: vista de la clase View del Meta-modelo. *Fuente: Creación propia*

Al igual que con los sensores, cada gráfico está representado individualmente. Así es posible añadir las características particulares de cada uno. Todos los gráficos heredan de la clase Graph lo que facilita añadir más gráficos en el futuro.

En la Figura 22 se muestran las clases que representan los gráficos. La clase Graph posee un atributo llamado category (del tipo GraphCategory) que representa la categoría a la cuál pertenece el gráfico[12]. Inicialmente se pensó utilizar este atributo para mostrar al usuario final la categoría del gráfico. Aunque no fue implementada en la herramienta presentada en este proyecto, es posible añadir esta información, ya sea para indicar la categoría del gráfico en la ventana de propiedades, en la etiqueta o para crear capas en la herramienta. Esta última opción, puede permitir mostrar solo gráficos que pertenezcan a una categoría específica y ocultar los otros.

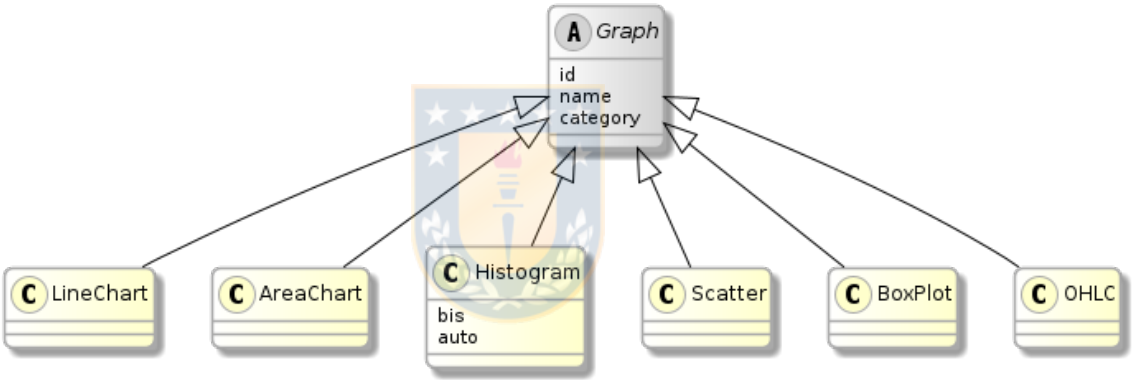


Figura 22: vista de la clase Graph del Meta-modelo. *Fuente: Creación propia*

Cada una de las clases del modelo tienen su representación en XMI. Esta representación es generada automáticamente por la herramienta ecore tool. Como se observa en la Figura 23 el archivo XMI guarda la clase, atributos, relaciones y la documentación que debe ser generada por quién genera el Meta-modelo.

[12]Las categorías fueron obtenidas de `https://datavizcatalogue.com/search.html`

```
  <eClassifiers xsi:type="ecore:EClass" name="Vis4bridge">
    <eAnnotations source="http://www.eclipse.org/emf/2002/GenModel">
      <details key="documentation" value="Vis4bridge is the root class of the metamodel, this class is made up of
the main elements of the model. The elements are divided into two large groups.&#xA;&#xA;On the one hand there
are the elements that deliver information, such as sensors, weather station or other groups of sensors. On the
other hand, there are the visual elements that will be generated from this information. Within the visual
elements are the views (see), graphs (aka chart) tables and cards.&#xA;Attributes:&#xA;name: Name of the
application. *&#xA;&#xA;* All classes have an attribute called name. this attribute is used by sirius to give a
default name to the class label."/>
    </eAnnotations>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType http://www.eclipse.org
/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="visualization" upperBound="-1"
        eType="#//Visualization" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="view" upperBound="-1" eType="#//View"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="sensor" upperBound="-1"
        eType="#//Sensor" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="daq" upperBound="-1" eType="#//DAQ"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="usergroup" upperBound="-1"
        eType="#//UserGroup" containment="true"/>
  </eClassifiers>
```

Figura 23: Extracto del archivo XMI generado a partir del Meta-modelo. Clase Vis4bridge. *Fuente: Creación propia*.

5.5. Representación Gráfica

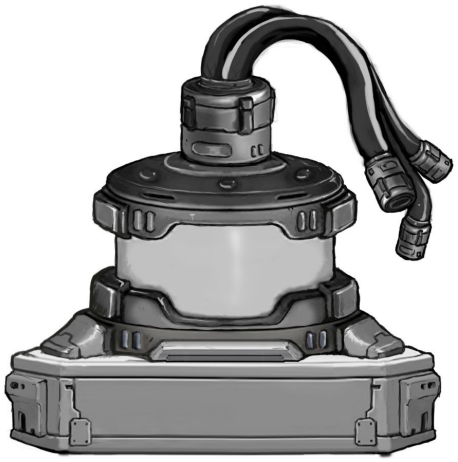
Teniendo las reglas que definen el lenguaje mediante el Meta-modelo, es posible generar una representación gráfica para los elementos que componen el Lenguaje Específico de Dominio (DSL). Una de las principales ventajas del uso de un DSL es que la solución es expresada en el idioma y nivel de abstracción del experto en el dominio, por lo que los expertos pueden entender por ellos mismos los programas basados en un DSL [27]. Además del uso de abstracciones del dominio, el presente trabajo hace uso de heurísticas de usabilidad para complementar esta relación entre el DSL y el experto en el dominio que usará este lenguaje. En particular, se utilizaron heurísticas de usabilidad de Nielsen [12] en la definición de la representación gráfica. Así los elementos del DSL y sus relaciones fueron creados teniendo en consideración estas heurísticas. Los elementos representados fueron sensores, gráficos, tarjetas, relaciones y contenedores.

5.5.1. Sensores, Gráficos y Tarjetas

El primer paso fue representar las abstracciones presentes en el Meta-modelo. Para los sensores se utilizó su representación física para generar una relación entre el sistema y el mundo real. Se puede observar un ejemplo de esto la Figura 24.

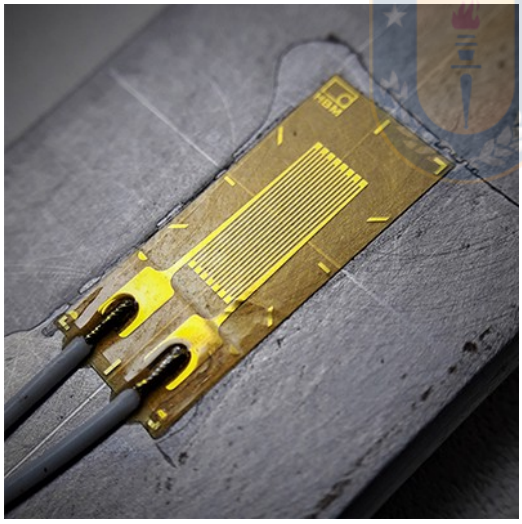
(a) Acelerómetro Real.

Fuente: fotografía propia



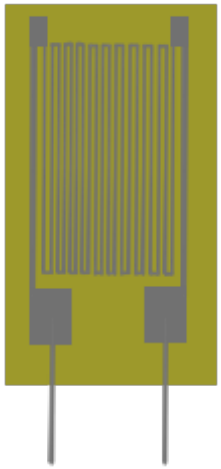
(b) Representación gráfica Acelerómetro.

Fuente: G. Jara



(c) Strain Gauge Real.

Fuente:Encardio Rite

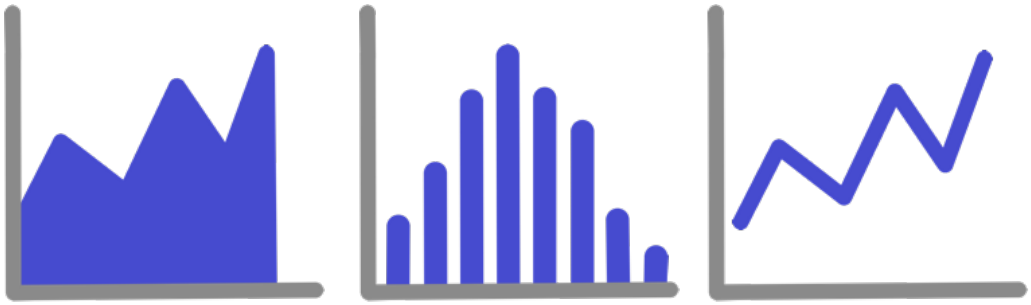


(d) Representación gráfica Strain Gauge.

Fuente: Creación Propia

Figura 24: Representación de Sensores

La representación de los gráficos se hizo en forma de iconos mostrando solo los ejes y la forma (curva, barras, puntos) que caracterizan al gráfico para así mantener un diseño estético y minimalista.



(a) Representación Gráfico de área. (b) Representación Histograma. (c) Representación Gráfico de línea

Figura 25: Representación de gráficos en la Herramienta *vis4bridge*. *Fuente: Creación Propia*.

El diseño de tarjetas se hizo de manera similar a las tarjetas mostradas en herramientas de visualización de datos como dash[13], kibana[14] o Power BI[15]. Sobre la tarjeta se representa el tipo de información que mostrará y el nombre de la tarjeta.

(a) Tarjeta Promedio. (b) Tarjeta Máximo.

Figura 26: Representación de Tarjetas. *Fuente:Diseño Propio*.

5.5.2. Vistas y Grupos

Cada vista (view) representa una agrupación de elementos que serán visualizados como gráficos y tarjetas. Las vistas son independientes y pueden tener propiedades que afectan a los elementos que están dentro de la vista como filtros, rango de fechas u otros. Por este motivo, la elección más indicada para representar la vista es un contenedor que agrupe a estos elementos al cuál asociar las características comunes.

[13]https://plotly.com/dash/
[14]https://www.elastic.co/es/kibana
[15]https://powerbi.microsoft.com/es-es/

En el diseño inicial, cada sensor estaba conectado mediante una flecha directamente a un gráfico. Esto para indicar que se ha agregado un nuevo sensor al gráfico que será generado. Esto causaba confusión al usuario y convertía el proceso de asociación de elementos en algo molesto. Esto llevó a añadir al lenguaje contenedores de sensores. Entre estos contenedores está el contenedor userGroup que permite al usuario agrupar sensores para luego asociar un grupo específico de sensores a un gráfico. De esta manera se reduce significativamente el número de elementos mejorando la sintaxis, pero manteniendo la interpretación.

5.5.3. Relaciones e interacción entre elementos

Otra característica esperada en este DSL es que los usuarios puedan mediante el mismo lenguaje tener claridad de los elementos que serán generados (mantener la visibilidad y estado del sistema). Para esto, se añadió información visual que se presenta de forma dinámica a través de los iconos generados.

Al asociar un gráfico a un grupo de sensores, el icono del gráfico cambia dinámicamente según el número de sensores que se encuentren en el grupo. Por ejemplo, al asociar un gráfico de líneas a un grupo de 3 sensores, el icono del gráfico de línea ahora presentará 3 líneas el lugar de una. De manera similar, si se asocia un grupo de 2 sensores a un histograma entonces el icono del histograma pasará a tener dos grupos de barras. Con esto, el usuario que está construyendo el modelo a través del DSL tendrá más claridad de los elementos que serán generados en la visualización.

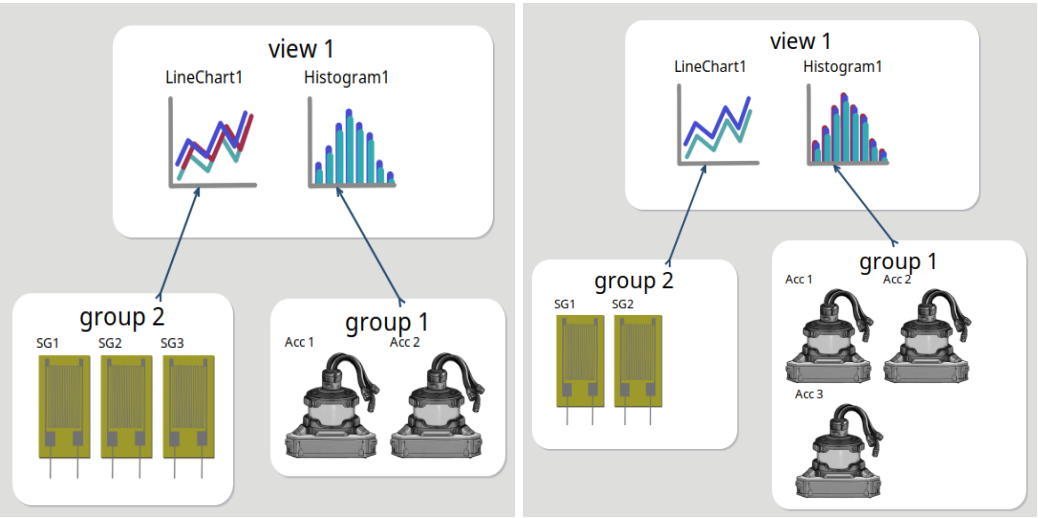


Figura 27: En las imágenes se puede observar como los gráficos cambian dinámicamente en función del número de elementos asociados. *Fuente: Creación Propia*.

Cada tarjeta también cambia su etiqueta dependiendo de su tipo, que puede ser máximo, mínimo, promedio u otra.

Existen ciertas consideraciones generales que fueron incluidas en el DSL para todos los elementos que lo componen. Al elegir el color de los elementos en la aplicación se utilizaron dos colores como base, a partir de estos, fue generada la paleta mediante la herramienta online colourco[16] que se basa en el modelo HSL que permite obtener los colores complementarios y analógicos en base a un color específico.

[16]https://colourco.de/

6. ESTRATEGIA DE GENERACIÓN AUTOMÁTICA DE CÓDIGO

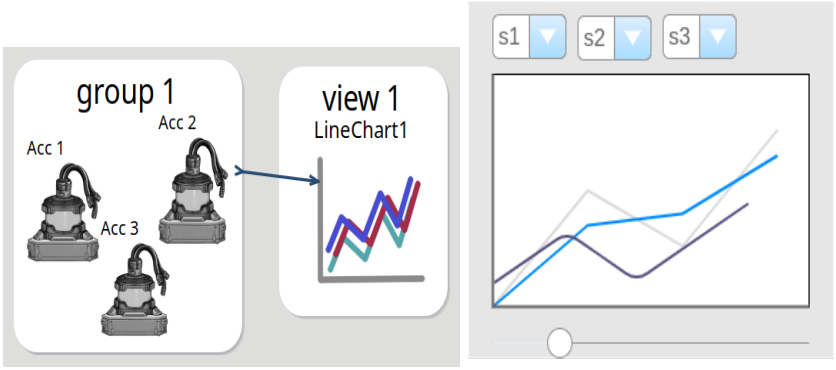
6.1. Salidas Esperadas por Expertos en el Dominio

Con el DSL, los expertos en el dominio pueden representar las visualizaciones que esperan obtener en el Monitoreo Estructural de Puentes.

Antes de comenzar a definir salidas generales, se deben describir las salidas esperadas por cada componente. Para esto, hay que recordar que los elementos visibles por el usuario serán aquellos que están en cada vista (view) del DSL. Los sensores, que se encuentran en los grupos definidos por el usuario, serán representados en los gráficos o tarjetas de cada vista.

Siguiendo esta idea, se mostrarán algunas salidas esperadas de gráficos. Primero, de cada gráfico, y posteriormente algunas salidas compuestas por más de un gráfico.

En el caso de un gráfico de línea asociado a dos sensores, este debe mostrar una línea por cada sensor. El DSL representa de manera genérica cada sensor, si se desea configurar una fuente de datos específica, esta se debe indicar en el archivo de salida. También son posibles más opciones como el rango de fechas asociados a cada sensor.

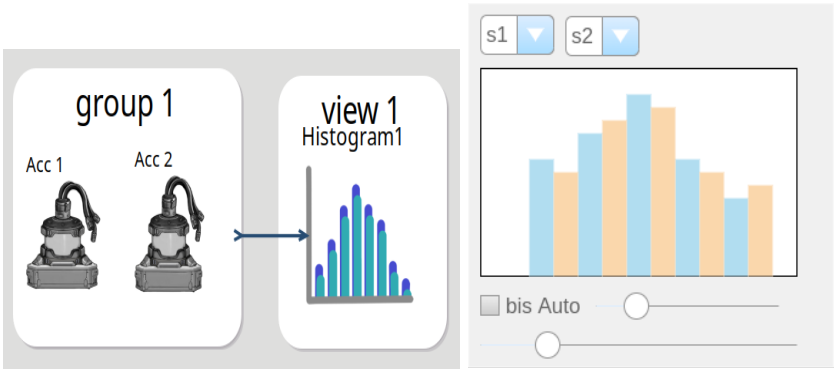


(a) Linechart. (b) Salida esperada.

Figura 28: Salida esperada para un gráfico de linea. *Fuente:Diseño Propio*.

Los histogramas son gráficos que permiten ver la tendencia de los datos, en este casos los datos de cada sensor serán representados mediante barras. Dentro de las características del histograma está el intervalo de las clases. Al igual que en

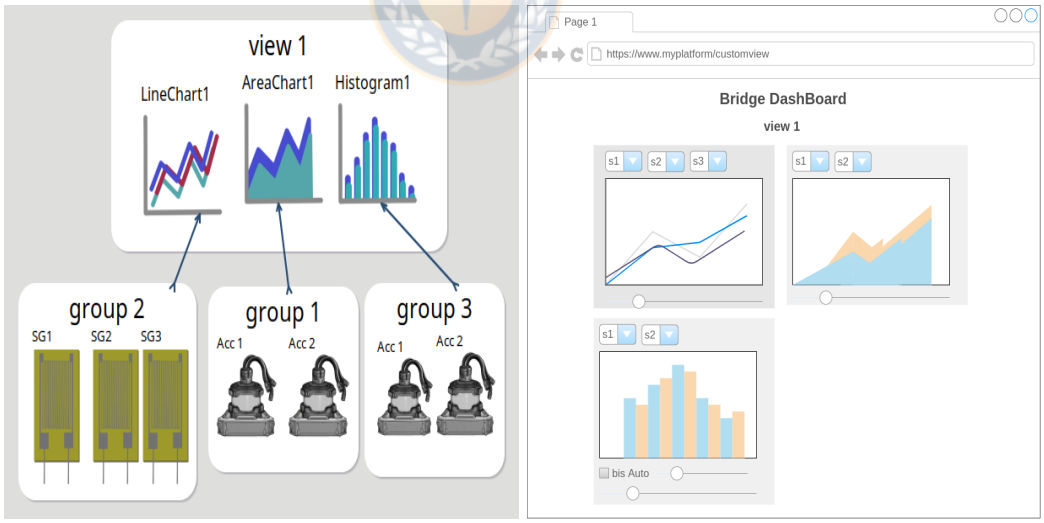
el gráfico de línea es posible seleccionar el rango de fechas para cada sensor.



(a) Histogram. (b) Salida esperada.

Figura 29: Salida esperada para un histograma. *Fuente: Diseño Propio*.

Teniendo una representación de los elementos individuales se puede tener una salida esperada de los requerimientos de visualización, en la Figura 30 se muestra una de las salidas esperadas.



(a) Ejemplo del Modelo. (b) Salida esperada.

Figura 30: Ejemplo del modelo con su salida esperada. *Fuente:Diseño Propio*.

6.2. Lenguaje de Destino

6.2.1. Elección del lenguaje para la transformación

Para la elección del lenguaje de la transformación fueron considerados aspectos como: Soporte de bibliotecas de visualización, soporte de bibliotecas para manejo de datos, generación de visualizaciones web y cercanía del lenguaje al usuario.

Existen distintos criterios utilizados para seleccionar el lenguaje de destino para la transformación. Entre estos criterios se encuentra el soporte de bibliotecas para la visualización, generación de visualizaciones web y lenguaje de programación adecuado para los expertos en el dominio.

Lenguaje	Bibliotecas de Visualización	Soporte Vis. Web	Para E. del Dominio
Matlab	Sí, posee bibliotecas nativas	No	Sí
Python	Matplotlib, Plotly, Leather, Seaborn, otras	Sí	Sí
Java Script	D3js, ChartJS, ReChart, otras	Sí	No
R	ggplot2, Plotly, otras	Sí	Sí

Tabla 2: Comparativa de Bibliotecas de Visualización, creación propia.

De los lenguajes consultados se eligió Python como lenguaje de destino para la transformación por ser uno de los más completos en cuanto a requisitos.

Como biblioteca de visualización se utilizó Plotly, esto mediante la versión Open Source de Dash que es un framework de desarrollo que permite generar aplicaciones de visualización web utilizando solo Python. El manejo de datos se hizo mediante Pandas.

6.3. Esquema de la transformación

La transformación se realizó mediante Acceleo, esta tecnología permite la transformación de un modelo EMF a Texto. Acceleo esta basado en el estándar MTL (Model to Text Language) del Object Management Group. Al realizar la transformación fue necesario considerar aspectos específicos del lenguaje de destino.

Aunque la implementación fue generada de acuerdo a los criterios mencionados en la Sub-sección 6.2, el DSL está construido para ser independiente del lenguaje de destino seleccionado. Para demostrar adicionalmente la transformación generada con Python y la biblioteca Dash se generó una transformación en Javascript con la biblioteca ChartJS. La elección de este segundo lenguaje se basó en la posibilidad de ejecutar las visualizaciones directamente desde Front-End, dando la posibilidad de ejecutar visualizaciones prescindiendo de un servidor web.

El esquema de ambas transformaciones se mostrará en este capítulo.

6.3.1. Esquema de Transformación del Modelo a Python

En la Figura 31 el árbol de directorios corresponde a los paquetes que componen la transformación. Para un mayor detalle, puede ver el diagrama de paquetes mostrado en el Anexo B.1.

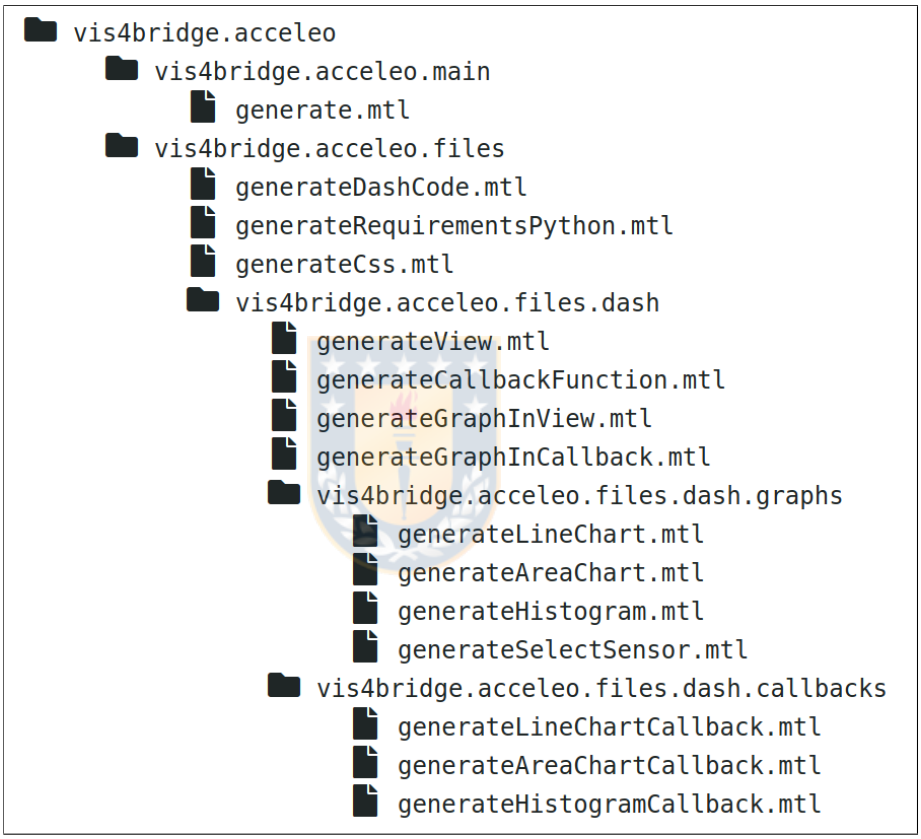


Figura 31: Paquetes de la transformación de modelo a texto. *Fuente: Creación propia*.

El paquete raíz está compuesto de dos paquetes. Dentro de los paquetes se encuetran archivos con extensión mtl. Cada archivo está compuesto por un módulo de Acceleo, el nombre de cada módulo es el mismo del archivo pero omitiendo la extensión.

Paquete vis4bridge.acceleo.main:

Este paquete contiene el módulo principal llamado `generate`. Este módulo es el encargado de generar la transformación. Para esto, llama a los módulos ubicados

en el paquete `vis4bridge.acceleo.files`.

Paquete vis4bridge.acceleo.files:

Cada módulo en este paquete genera un archivo de salida. Los módulos se detallan a continuación:

- El módulo `generateDashCode` genera un archivo llamado app.py que contiene la aplicación. Para esto llama a módulos contenidos en el paquete `vis4bridge.acceleo.files.dash`. El archivo se genera en una carpeta llamada output ubicada en el directorio raíz del proyecto.
- El módulo `generateRequirementsPython` genera un archivo llamado requirements.txt que proporciona la lista de requerimientos (bibliotecas necesarias) de la aplicación. El archivo se genera en una carpeta llamada output ubicada en el directorio raíz del proyecto.
- El módulo `generateCSS` genera un archivo llamado custom.css que contiene la hoja de estilos de la aplicación. El archivo se genera en una carpeta llamada assets dentro de la carpeta output del proyecto.

Paquete vis4bridge.acceleo.files.dash:

Contiene módulos que importa `generateDashCode` para generar el código de la aplicación. Los módulos se detallan a continuación:

- El módulo `generateView` genera el código en Python encargado de la disposición de las vistas (agrupación de gráficos y tarjetas) en la aplicación. Este módulo importa `generateGraphInView` para generar el código que distribuye los gráficos en cada vista.
- El módulo `generateCallbackFunction` genera el código en Python de las funciones que generan y actualizan los gráficos, tarjetas y otros elementos dinámicos de la aplicación. Este módulo importa `generateGraphInCallback` que genera el código que actualiza los gráficos.

Paquete vis4bridge.acceleo.files.dash.graphs:

Contiene módulos que importa `generateGraphInView`. Los módulos en este paquete generan el código específico para distribuir los gráficos en cada vista. Los módulos se detallan a continuación:

- Los módulos `generateLineChart`, `generateAreaChart` y `generateHistogram` generan el código en Python encargado de la disposición del gráfico de línea, área e histograma respectivamente. Además generan el código de los componentes asociados a cada gráfico.

- El módulo `generateSelectSensor` es llamado por los otros módulos de este paquete para generar el componente Select que permite seleccionar los distintos sensores asociados al gráfico.

Paquete vis4bridge.acceleo.files.dash.callbacks:

Contiene módulos que importa `generateCallbackFunction`. Los módulos en este paquete generan el código específico para cargar los datos y actualizar la información de los gráficos. Los módulos se detallan a continuación:

- Los módulos `generateLineChartCallback`, `generateAreaChartCallback` y `generateHistogramCallback` generan las funciones que actualizan los respectivos gráficos.

6.3.2. Esquema de Transformación del Modelo a Javascript

En la Figura 32 el árbol de directorios corresponde a los paquetes que componen la transformación. Para un mayor detalle, puede ver el diagrama de paquetes mostrado en el Anexo B.2.

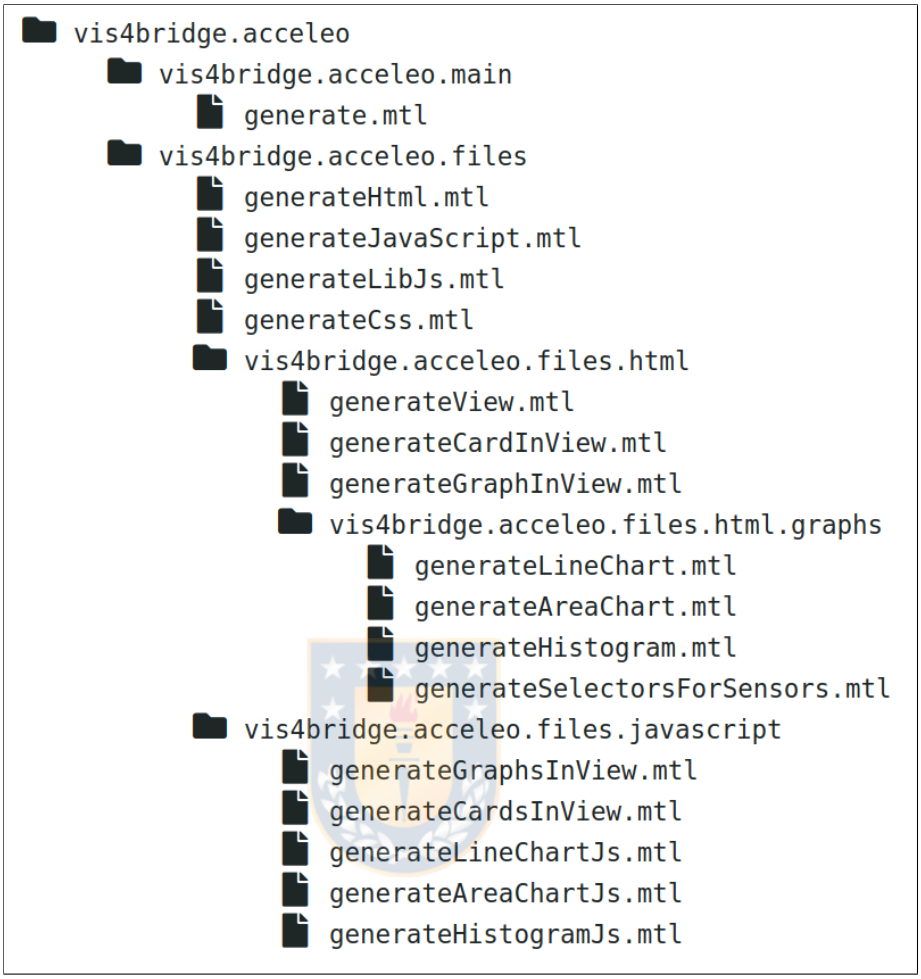


Figura 32: Paquetes de la transformación de modelo a texto. *Fuente: Creación propia*.

Como en el caso anterior, el paquete raíz está compuesto de dos paquetes. Dentro de los paquetes se encuetran archivos con extensión mtl. Cada archivo está compuesto por un módulo de Acceleo, el nombre de cada módulo es el mismo del archivo pero omitiendo la extensión.

Paquete vis4bridge.acceleo.main:

Este paquete contiene el módulo principal llamado `generate`. Este módulo es el encargado de generar la transformación. Para esto, llama a los módulos ubicados en el paquete `vis4bridge.acceleo.files`.

Paquete vis4bridge.acceleo.files:

Cada módulo en este paquete genera un archivo de salida. Los módulos se detallan a continuación:

- El módulo `generateHtml` genera un archivo llamado index.html que contiene el punto de entrada a la aplicación. Para esto llama a módulos contenidos en el paquete `vis4bridge.acceleo.files.html`. El archivo se genera en una carpeta llamada output ubicada en el directorio raíz del proyecto.
- El módulo `generateJavaScript` genera un archivo llamado scripts.js que proporciona la lista de requerimientos (bibliotecas necesarias) de la aplicación. El archivo se genera en una carpeta llamada js ubicada en el carpeta output del proyecto.
- El módulo `generateLibJs` genera un archivo llamado lib.js. Este archivo contiene todas las funciones encargadas de generar los gráficos y su actualización. Estas funciones son llamadas desde el archivo scripts.js. Así el código generado por el módulo `generateLibJs` siempre es el mismo en cada transformación.
- El módulo `generateCSS` genera un archivo llamado master.css que contiene la hoja de estilos de la aplicación.

El módulo `generateRequirementsPython` ya no existe (o uno similar) debido a que las bibliotecas requeridas para el funcionamiento se cargan mediante CDN (Content Delivery Network) en el modulo `generateHtml`.

Paquete vis4bridge.acceleo.files.html:

Contiene módulos que importa `generateHtml` para generar el código de la aplicación. Los módulos se detallan a continuación:

- El módulo `generateView` genera el código en Html encargado de la disposición de las vistas (agrupación de gráficos y tarjetas) en la aplicación. Este módulo importa `generateGraphInView` y `generateCardInView` para generar el código que distribuye los gráficos y tarjetas en cada vista.

Paquete vis4bridge.acceleo.files.html.graphs:

Contiene módulos que importa `generateGraphInView`. Los módulos en este paquete generan el código específico para distribuir los gráficos en cada vista. Los módulos se detallan a continuación:

- Los módulos `generateLineChart`, `generateAreaChart` y `generateHistogram` generan el código en Html encargado de la disposición del gráfico de línea, área e histograma respectivamente. Además generan el código de los componentes asociados a cada gráfico.

- El módulo `generateSelectorsForSensor` es llamado por los otros módulos de este paquete para generar el componente Select que permite seleccionar los distintos sensores asociados al gráfico.

Paquete vis4bridge.acceleo.files.javascript:

Contiene módulos que importa `generateJavaScript`. Los módulos en este paquete generan el código que llama a la biblioteca lib.js para inicializar y actualizar los gráficos y sus componentes asociados.

- Para cada vista (view), el módulo `generateGraphsInView` llama a los módulos `generateLineChartJs`, `generateAreaChartJs` y `generateHistogramJs` si estos son requeridos.
- Los módulos `generateLineChartJs`, `generateAreaChartJs` y `generateHistogramJs` se encargan de llamar a las funciones de la biblioteca lib.js que generan cada gráfico según el modelo de entrada.
- El módulo `generateCardsInView` carga y actualiza los datos a las tarjetas generadas por `generateCardInView`.

6.4. Consideraciones al implementar un lenguaje de destino

Es posible realizar la transformación del modelo a texto utilizando otro lenguaje de destino, pero se debe considerar que:

- El lenguaje de destino debe tener soporte para bibliotecas de visualización.
- El modelo no está diseñado para generar alternativas de visualización individualmente, para esto se deberá modificar el modelo.
- Los ajustes de tamaño y disposición de elementos no se encuentran como atributos de los elementos del modelo, esto se debe definir en la transformación de modelo a texto, si desea dar estas opciones en el DSL deberá agregar estos atributos al modelo.
- Si se desea una generación de código totalmente ejecutable, deberá ajustar la fuente de datos en la transformación de modelo a texto asegurándose que el lenguaje de destino soporte la fuente de datos.

6.5. Reglas implementadas

A continuación se mostrarán y explicarán algunas de las reglas implementadas en la transformación en Python. Si desea ver todas las reglas puede ver el Anexo C. También se encuentra disponible una versión en `https://gitlab.com/braulioqh/vis4bridgem2t`.

En la Figura 33 se muestra la implementación del módulo `generateDashCode` con Acceleo. En las primeras líneas de código se puede observar que el módulo es generado a partir del Meta-modelo vis4bridge, esto permitirá mapear los elementos del modelo al código.

```
1 [comment encoding = UTF-8 /]
2 [**
3  * The documentation of the module generateDashCode.
4  */]
5 [module generateDashCode('https://gitlab.com/braulioqh/vis4bridge.git')]
6 [import braulioqh::vis4bridge::acceleo::files::dash::generateView /]
7 [import braulioqh::vis4bridge::acceleo::files::dash::generateCallbackFunction /]
8 [**
9  * The documentation of the template generateDashCode.
10 * @param aVis4bridge
11 */]
12 [template public generateDashCode(aVis4bridge : Vis4bridge)]
13 [file ('app.py', false, 'UTF-8')]
14 [initialStaticContent(aVis4bridge) /]
15
16 # Component tree is defined using app.layout
17 app.layout = html.Div(children =[
18     html.H1(children='Bridge DashBoard'),
19     html.Div(children='''
20         Dash: A web application framework for Python.
21     '''),
22
23 [for (itView : View | self.view)]
24 [generateView(itView) /]
25 [/for]
26 ]/],
27 id='visualization')
28
29 [for (itView : View | self.view)]
30 [generateCallbackFunction(itView) /]
31 [/for]
32
33 # Startup configuration for dash
34 if __name__ == '__main__':
35     app.run_server(debug=True)
36 [/file]
37 [/template]
```

Figura 33: Módulo generateDashCode. *Fuente: Creación propia*.

El módulo `generateDashCode` recibe como entrada un ejemplar del Meta-modelo construido a partir del DSL (Línea 12 del código fuente). Dentro del módulo se indica el nombre del archivo generado y la codificación. Antes de comenzar a mapear cada elemento se llama al módulo `initialStaticContent` que carga todo el contenido estático como bibliotecas de Python y configuraciones.

Luego de cargar las bibliotecas correspondientes se inserta el código fuente en Python que generará las visualizaciones.

Se puede observar en el código que para generar las visualizaciones se crea un iterador `itView` encargado de recorrer las vistas contenidas en el modelo de entrada `aVis4bridge`, por cada vista se llama al módulo `generateView`. Luego, nuevamente se crea un iterador que para cada vista genera las funciones que actualizan los gráficos y elementos relacionados.

Finalmente se añade el código necesario para iniciar el servidor.

7. RESULTADOS DE LA IMPLEMENTACIÓN

La herramienta que genera las visualizaciones fue implementada mediante Obeo Designer [17]. Este, es un entorno de desarrollo basado en eclipse, que tiene preinstalado complementos que apoyan cada una de las etapas del Desarrollo Dirigido por Modelos.

A continuación, se muestra el resultado de los requerimientos implementados, indicados en la Sección 5 y la Sección 6. Se mostrará una figura con el modelo generado a partir del DSL y posteriormente el resultado generado a partir del modelo en Python y Javascript.

La Figura 34 muestra un modelo generado a partir del DSL. Aquí se pueden observar 2 vistas.

La primera vista llamada 'Accelerometer' posee un gráfico de línea, uno de área y un histograma. Cada gráfico está asociado a su respectivo grupo de sensores.

La segunda vista llamada 'Straingauge' posee 2 histogramas, cada uno de ellos asociados a sus respectivos grupos de sensores.

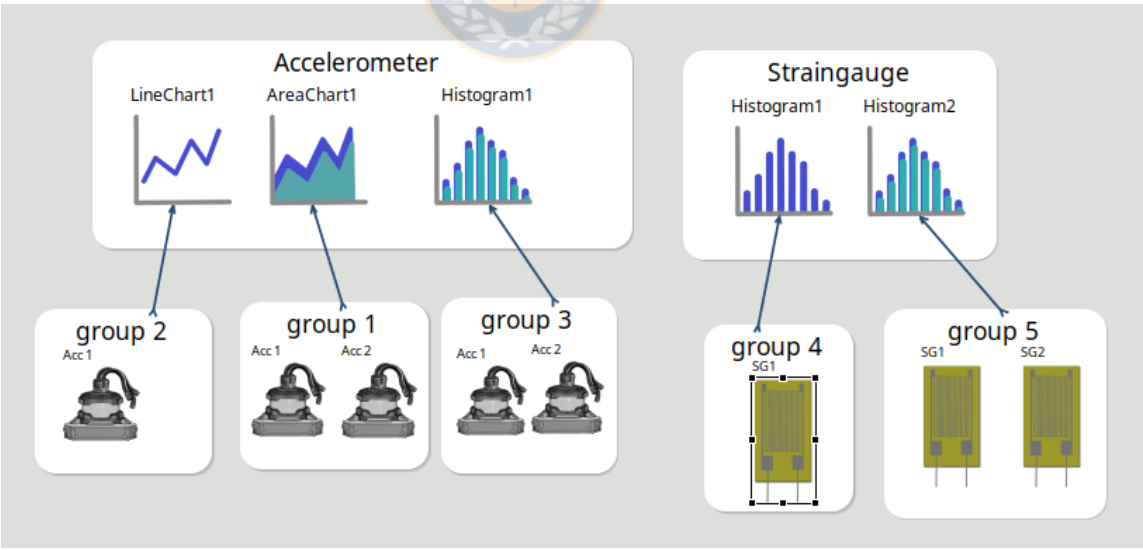


Figura 34: Modelo generado a partir del DSL. *Fuente: Creación propia*.

El resultado generado en base al modelo mostrado en la Figura 34 se muestra en la Figura 35 (salida utilizando Python) y la Figura 36 (Salida utilizando javascript), mostrando el resultado de la implementación generada en base al modelo

[17] https://www.obeodesigner.com

mostrado. Donde los requerimientos plasmados en el modelo se transforman a una visualización de datos en un Dashboard web que contiene cada uno de los gráficos solicitados separados en vistas.

El diseño de estos requerimientos concuerda con las salidas esperadas mostradas en la Sección 6. En la Figura 35 los datos utilizados para los acelerómetros son datos históricos reales del puente Tianjin Yonghe [38] que fueron facilitados para la realización de este proyecto. Los datos de los Straingauge son datos de prueba.

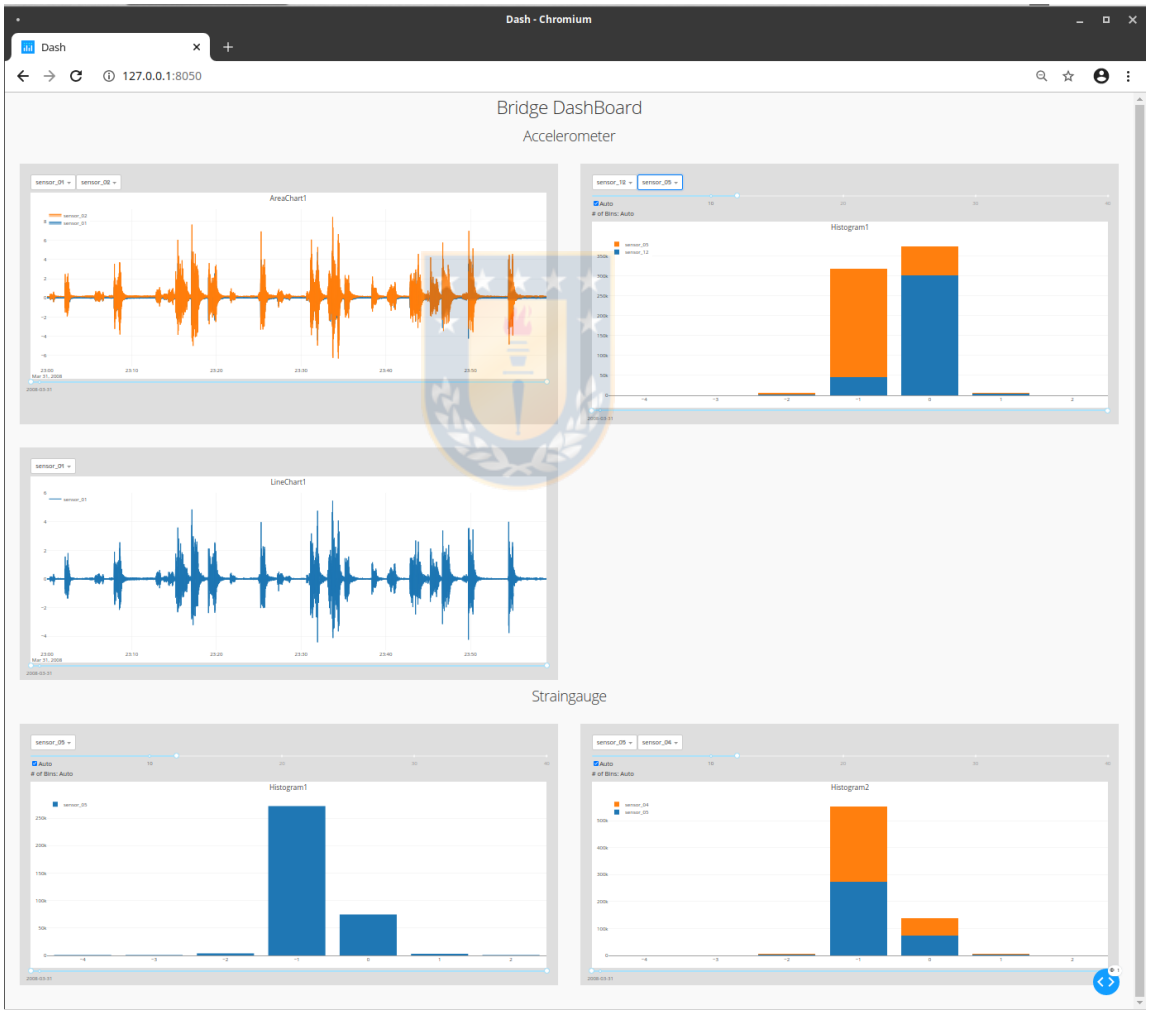


Figura 35: Aplicación generada a partir del DSL. *Fuente: Creación propia*.

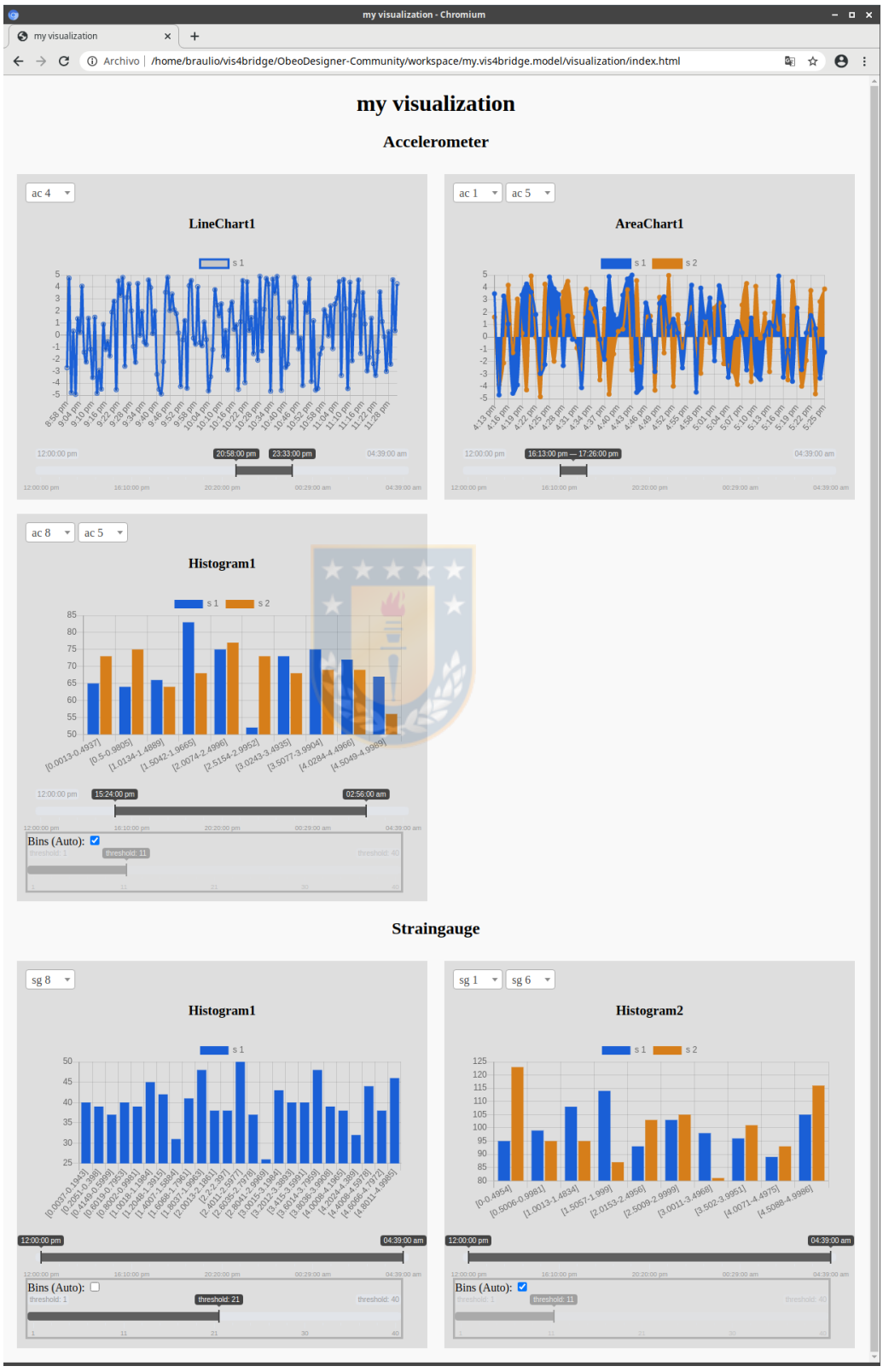


Figura 36: Aplicación generada a partir del DSL. *Fuente: Creación propia*.

La Figura 37 muestra otro modelo generado a partir del DSL. Aquí se puede observar 1 vista llamada 'Accelerometor', esta vista esta compuesta de 4 tarjetas: Una tarjeta que muestra el máximo, una que muestra el mínimo y 2 tarjetas que muestran el promedio de los datos.

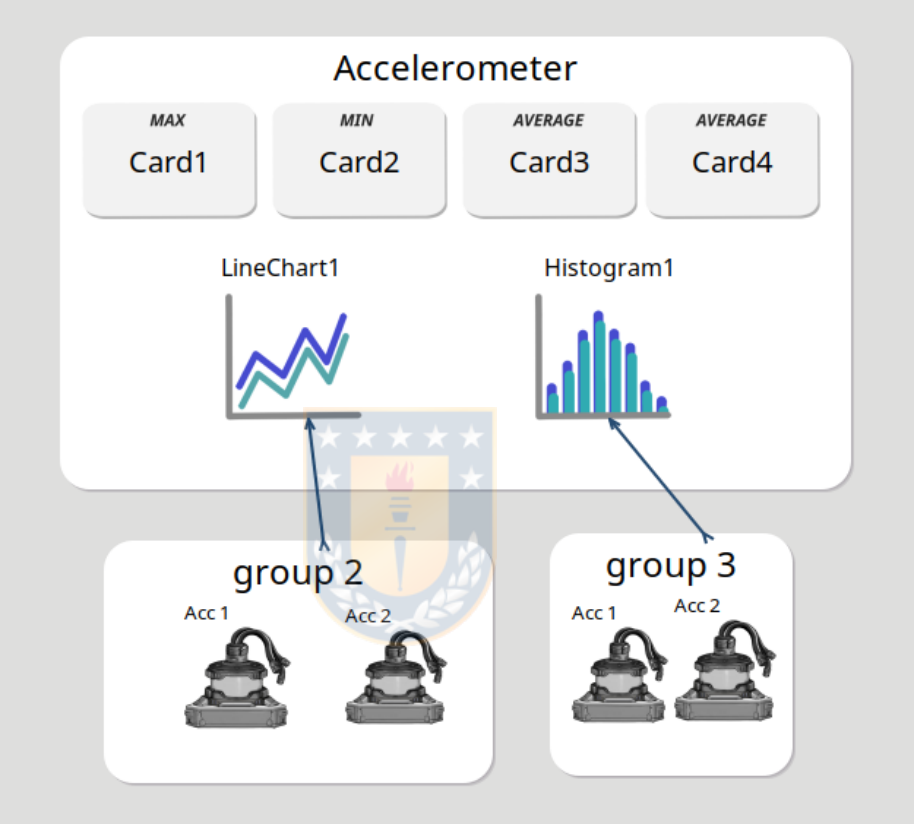


Figura 37: Modelo generado a partir del DSL. *Fuente: Creación propia*.

En la Figura 38 (salida en Python) y la Figura 39 (salida en javascript) se muestra el resultado de la generación automática de código generada a partir de la Figura 37.

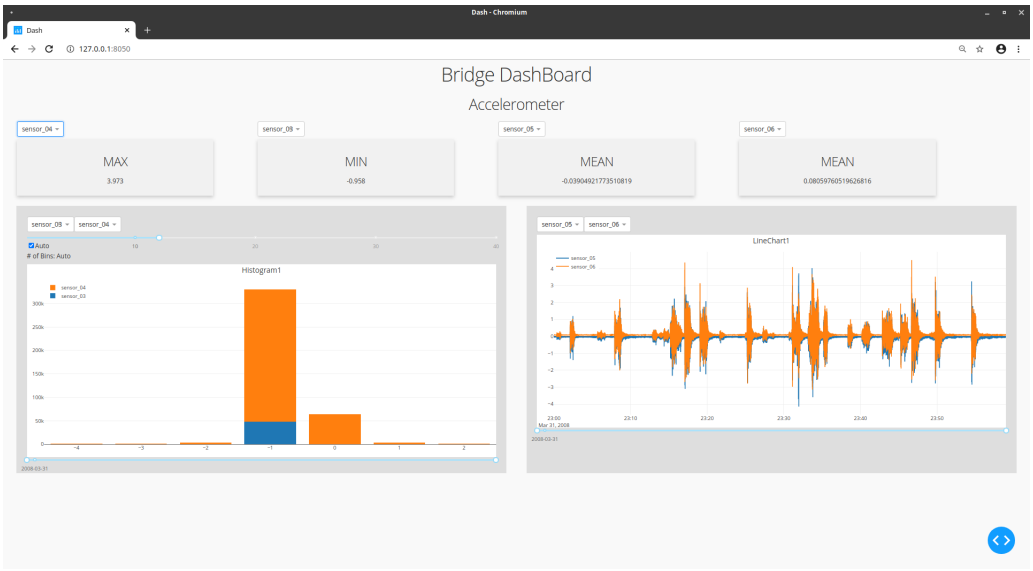


Figura 38: Aplicación generada a partir del DSL. *Fuente: Creación propia*.

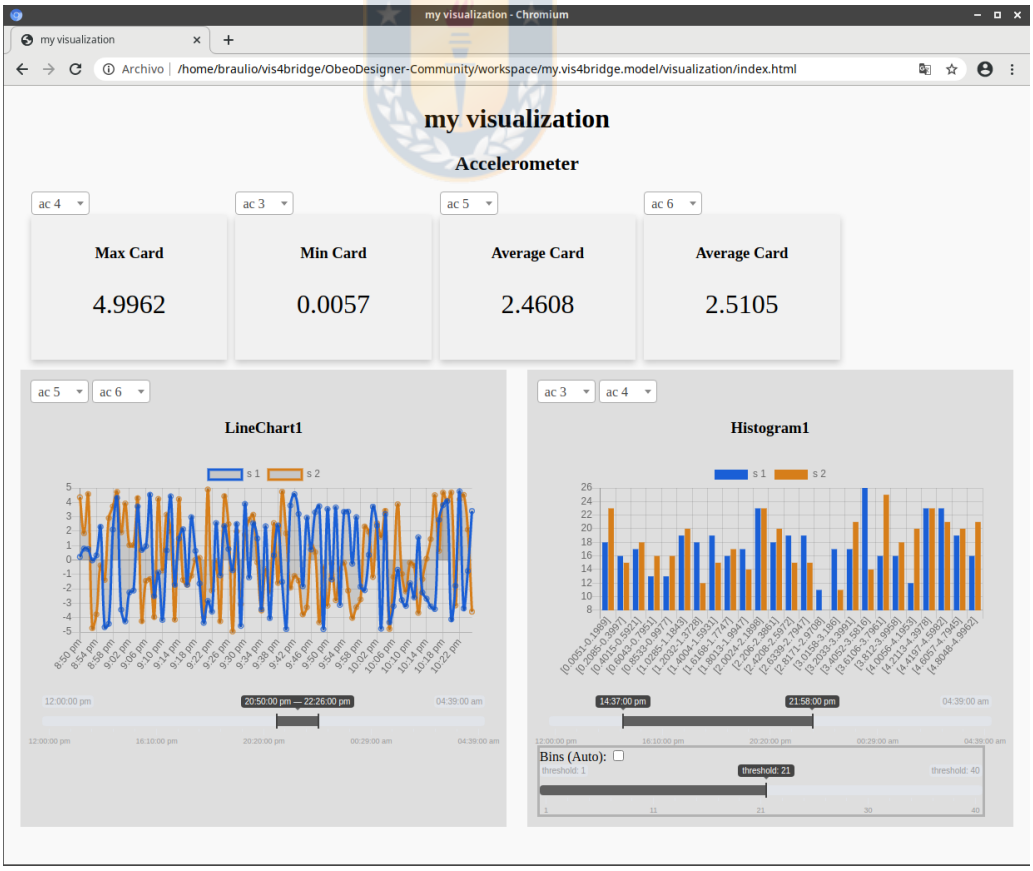


Figura 39: Aplicación generada a partir del DSL. *Fuente: Creación propia*.

8. EVALUACIÓN

En la Sección 7 se puede apreciar la factibilidad de generar distintas visualizaciones a partir de un Meta-modelo. En este apartado, se mostraran las evaluaciones que fueron realizadas a partir de este proyecto.

8.1. Cobertura de visualizaciones

8.1.1. Visualizaciones soportadas

En este proyecto, aunque es posible añadir más alternativas de visualización según lo expuesto en la Seccion 5, las alternativas de visualización posibles de generar son:

- Gráficos de Línea.
- Gráficos de Área.
- Histogramas.
- Tarjetas con información de máximos/mínimos, promedios y alertas (estas últimas deben definirse).

Cabe destacar, que las visualizaciones son dinámicas, es decir, es posible cambiar ciertos parámetros en los gráficos y estos se vean reflejados en tiempo de ejecución.

8.1.2. Visualizaciones soportadas por las bibliotecas utilizadas

Se utilizaron las bibliotecas Dash para Python y ChartJs para Javascript.

Las principales visualizaciones soportadas por Dash son:

- Gráficos de Barra.
- Gráficos de Línea.
- Gráficos de Área.
- Scatter.
- Pie
- Gráficos de Burbuja.

- Caja y Bigote.
- Histogramas.
- Heatmaps.
- OHLC.
- Otros.

Las visualizaciones soportadas por ChartJs son:

- Gráficos de Barra.
- Gráficos de Línea.
- Gráficos de Área.
- Scatter.
- Doughnut.
- Pie.
- Área Polar.
- Radar.

8.1.3. Añadir soporte a más visualizaciones

Es posible añadir soporte a más visualizaciones presentes en las bibliotecas utilizadas. Esto se puede hacer siguiendo estos pasos:

Añadir el gráfico al Meta-modelo:

Independiente del lenguaje de destino, el primer paso es añadir la visualización al Meta-modelo mostrado en el Anexo A. Para esto debe añadir una clase para su visualización como hijo de la clase Graph. Si su gráfico posee propiedades específicas, deberá añadir estas propiedades como atributo de la clase recién generada.

Añadir el gráfico a la herramienta gráfica:

Una vez añadido el gráfico al Meta-modelo puede crear una representación en la herramienta. Para esto debe crear el correspondiente nodo y herramienta de

creación en Sirius.

Añadir el gráfico a la trasformación de modelo a texto:

Una vez añadido el gráfico al Meta-modelo deberá añadir el código necesario para la transformación. Si se toma como ejemplo la transformación a Javascript deberá escribir una función que genere el gráfico en el módulo `generateLibJs` (ver Sección 6) y llame a esta función desde el módulo `generateGraphsInView`. Luego deberá generar un módulo con el código html del gráfico en el paquete `vis4bridge.acceleo.files.-html.graphs` y llamar al nuevo módulo desde `generateGraphInView`.

8.2. Evaluación de Usabilidad

Para evaluar la propuesta Dirigida por Modelos, en términos de usabilidad, se planteó un diseño de estudio de casos donde los stakeholders, generaron distintas alternativas de visualización mediante una herramienta construida en base al DSL que se propone en este proyecto. Posteriormente, los usuarios contestaron un cuestionario de usabilidad basado en una de las encuestas de usabilidad propuesta por IBM [11] que ha sido parcialmente modificado para la evaluación de este proyecto. Puede revisar el cuestionario de usabilidad en el Anexo D.

8.2.1. Participantes de la evaluación

Se tomó un grupo de stakeholders, entre ellos, ingenieros en las áreas de Ingeniería Civil e Ingeniería Civil Informática. Además, estudiantes de dichas áreas que accedieron a:

- Realizar una prueba de la herramienta de desarrollo construida mediante MDSD en este proyecto.
- Contestar una encuesta de usabilidad sobre dicha herramienta.

Los participantes en esta prueba fueron contactados de forma individual.

8.2.2. Diseño de las Pruebas

Debido a la crisis sanitaria del Covid-19, las pruebas fueron realizadas en modalidad online. Para esto, se preparó una página web con todo el material necesario

para que los participantes realizaran las pruebas (Puede visitar la página en : https://vis4bridge.gitlab.io/).

La página contiene lo siguiente:

- Herramienta software para descargar. Disponible para sistema operativo Windows 7 o superior y Ubuntu 18.04 o superior (y derivados).
- Manual de instalación de la herramienta (en texto y video).
- Tutorial de uso de la herramienta (en texto y video).
- Escenarios para prueba de experiencia de uso.
- Cuestionario de usabilidad.

Las pruebas se realizaron de la siguiente forma:

- Los participantes accedieron a la página web y descargaron la herramienta.
- Luego, los participantes instalaron la herramienta. Para esto, accedieron al manual de instalación disponible en la página web. Se dio la opción a los participantes de seguir el manual en texto o mediante un video de instalación. El video de instalación tiene una duración de 7 minutos con 52 segundos. En este manual de instalación se indican los pre-requisitos de instalación (software y hardware), el software adicional necesario y cada uno de los pasos de instalación.
- Antes de construir los escenarios, los participantes debieron realizar un tutorial de uso de la herramienta. El tutorial se entregó en video mediante la página web indicada anteriormente. Este video tiene una duración de 7 minutos y 48 segundos. En el tutorial se indica como crear un proyecto, como utilizar la paleta de herramientas y la ventana de propiedades. Adicionalmente se dejó a disposición un tutorial en texto de forma opcional.
- Luego de realizado el tutorial de uso, los participantes realizaron las pruebas de experiencia de uso que consisten en llevar a cabo 3 escenarios que aumentan gradualmente la cantidad de opciones a modificar para realizar una visualización, los escenarios se muestran en la sub-sección 8.2.3. En cada escenario se indicó, mediante texto, el modelo que el participante debía diseñar en la herramienta y el resultado esperado mediante una imagen.
- Una vez terminado los 3 escenarios, cada participante contestó un cuestionario de usabilidad compuesto de 22 sentencias (3 correspondientes al

grado de satisfacción con respecto a las tareas realizadas y 19 corresponden al grado de satisfacción con respecto al sistema). En cada pregunta los encuestados respondieron con el grado de satisfacción de acuerdo a lo indicado en cada sentencia. El grado de satisfacción va desde Totalmente en desacuerdo hasta Totalmente de acuerdo pasando por: En desacuerdo, Parcialmente en desacuerdo, Ni de acuerdo ni en desacuerdo, Parcialmente de acuerdo y De acuerdo.

8.2.3. Escenarios

Se plantearon 3 escenarios para la construcción de las visualizaciones. En cada escenario se plantearon las instrucciones en forma escrita y una imagen mostrando el resultado esperado. A continuación se muestran los escenarios planteados:

Escenario 1:

Se solicita construir una visualización con las siguientes características:

- Genere una vista (view) con un gráfico de línea y una tarjeta.
- Cambie el nombre de la vista (por ejemplo a 'my view') en la pestaña de propiedades.
- Cree un grupo de sensores que contenga un acelerómetro (accelerometer) y un straingauge.
- Conecte el grupo de sensores con el gráfico de línea.
- Genere la visualización.

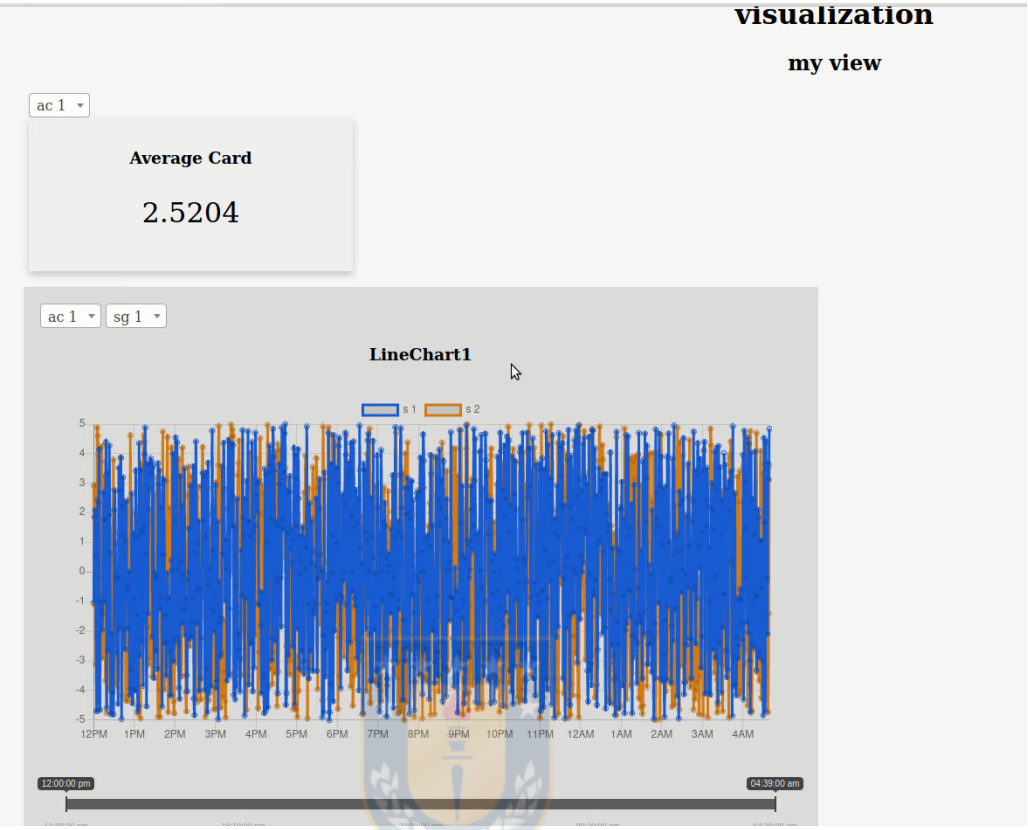


Figura 40: Resultado esperado para el escenario 1. *Fuente: Creación propia*.

Escenario 2:

Se solicita construir una visualización con las siguientes características:

- Genere una vista (view) con un gráfico de área, un histograma y una tarjeta.
- Cree un grupo de sensores que contenga 3 sensores a su elección. Conecte el grupo de sensores con el gráfico de área.
- Cree un grupo de sensores que contenga un acelerómetro (accelerometer) y un straingauge. Conecte el grupo de sensores al histograma.
- Genere la visualización.

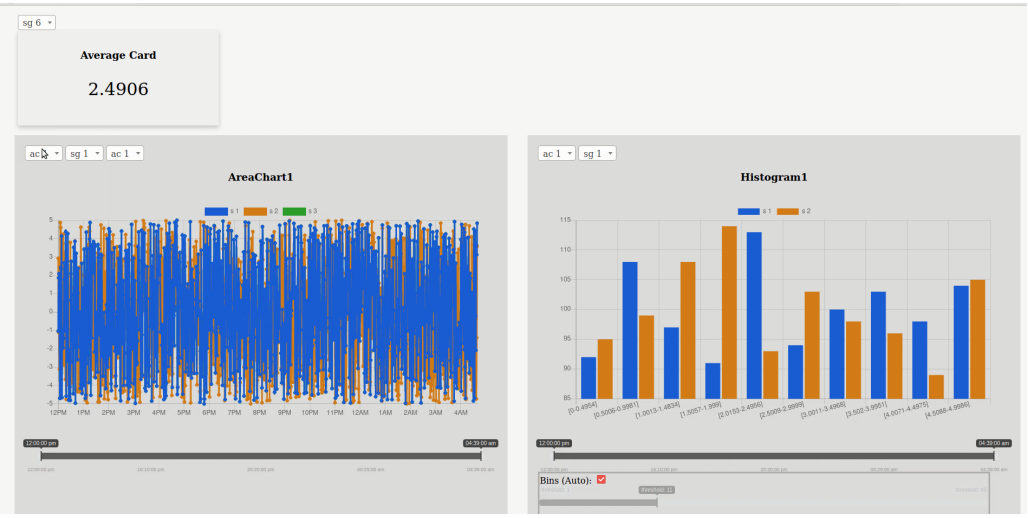


Figura 41: Resultado esperado para el escenario 2. *Fuente: Creación propia*.

Escenario 3:

Se solicita construir una visualización con las siguiente características:

- Genere 2 vistas (view).
- En la primera vista añada un gráfico de línea, un histograma y una tarjeta (card).
    - Cree un grupo de sensores que contenga 1 acelerómetro y un straingauge.Conecte el grupo de sensores a el gráfico de línea de la primera vista.
    - Cree un grupo de sensores que contenga 2 acelerómetros. Conecte el grupo de sensores al histograma de la primera vista.
    - Seleccione la tarjeta y en la pestaña properties cambie el valor Type a Max.
- En la segunda vista añada un gráfico de área, un histograma y una tarjeta.
    - Cree un grupo de sensores que contenga 1 acelerómetro y 2 straingauge. Conecte el grupo de sensores al gráfico de área de la segunda vista.
    - Cree un grupo de sensores que contenga 2 straingauge. Conecte el grupo de sensores al histograma de la segunda vista.
    - Seleccione la tarjeta y en la pestaña properties cambie el valor Type a Min.

- Genere la visualización.

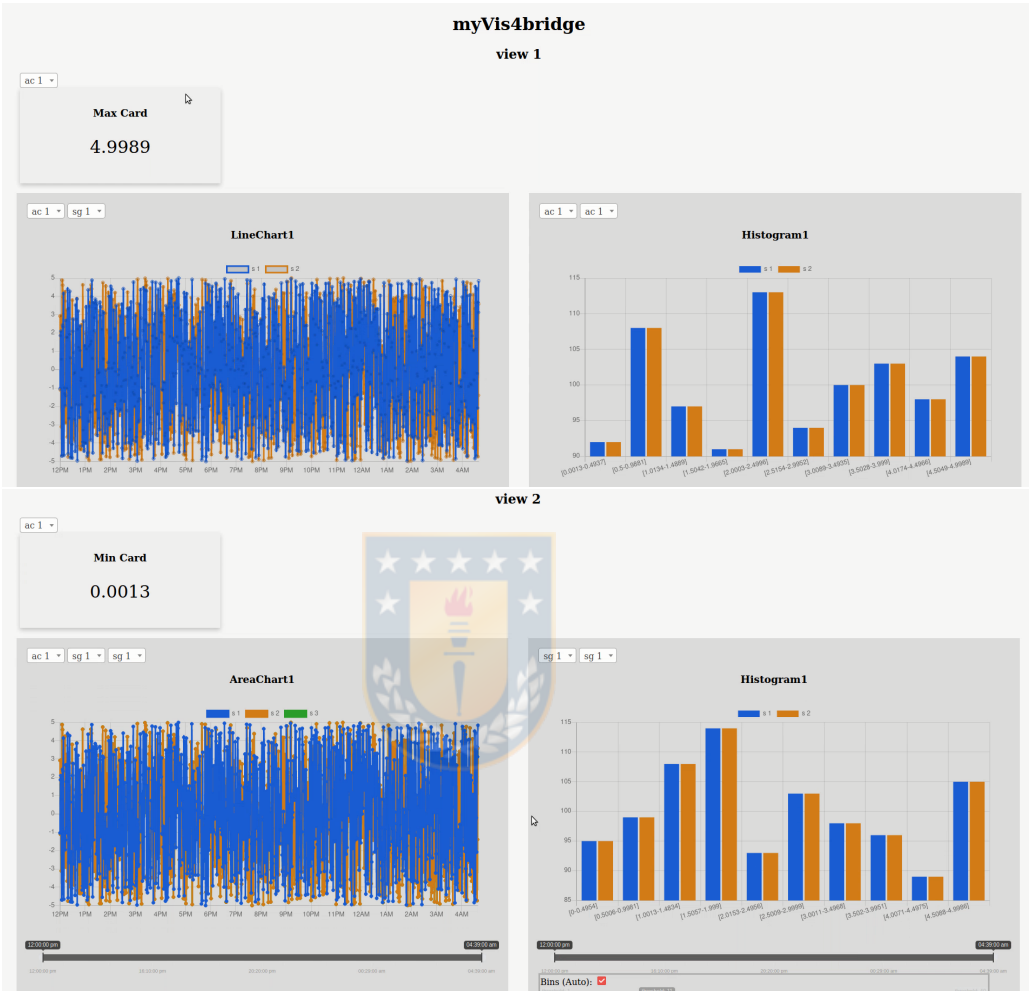


Figura 42: Resultado esperado para el escenario 3. *Fuente: Creación propia*.

8.2.4. Resultado de la Encuesta

La encuesta se realizó convocando a profesionales y estudiantes de ingeniería. En su mayoría, en la áreas de ingeniería civil e informática. En total participaron 17 personas.

A continuación, se muestra el resultado de la encuesta separada en dos partes. La primera parte, corresponde a las pregustas asociadas a la satisfacción del usuario al completar las tareas planteadas en los escenarios. La segunda parte, corresponde a la satisfacción del usuario con respecto al sistema.

Satisfacción al completar cada tarea:

- En general, estoy satisfecho con la facilidad al completar cada tarea.

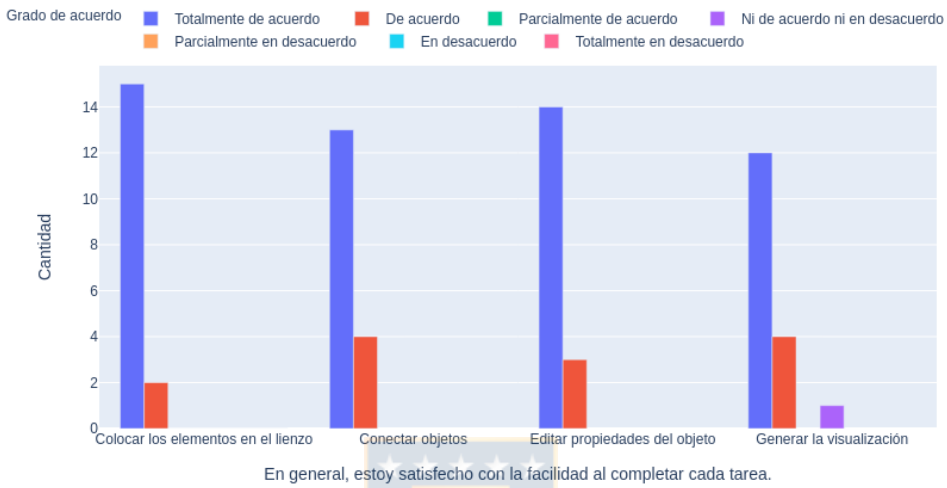


Figura 43: Grado de satisfacción al completar una tarea *Fuente: Creación propia*.

- En general, estoy satisfecho con la cantidad de tiempo que tomó completar cada tarea.

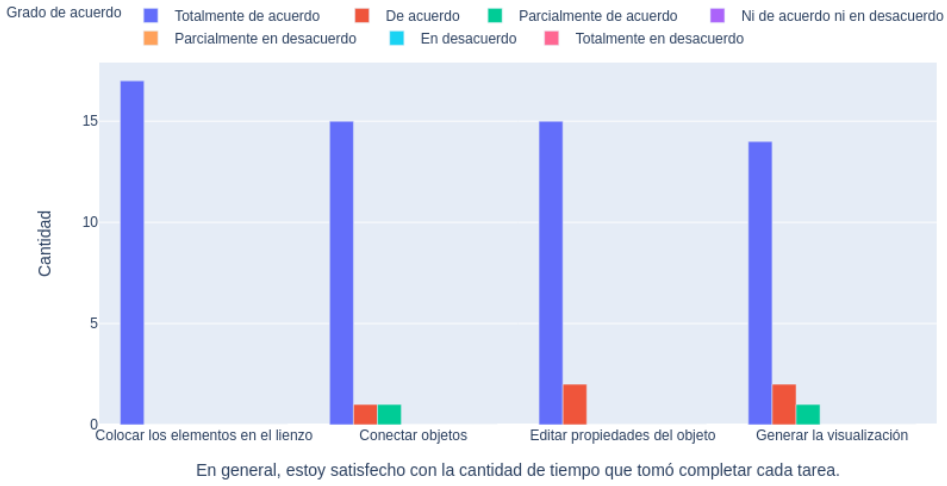


Figura 44: Grado de satisfacción del tiempo utilizado al completar una tarea *Fuente: Creación propia*.

- En general, estoy satisfecho con la información de soporte (ayuda en línea, mensajes, documentación) al completar cada tarea.

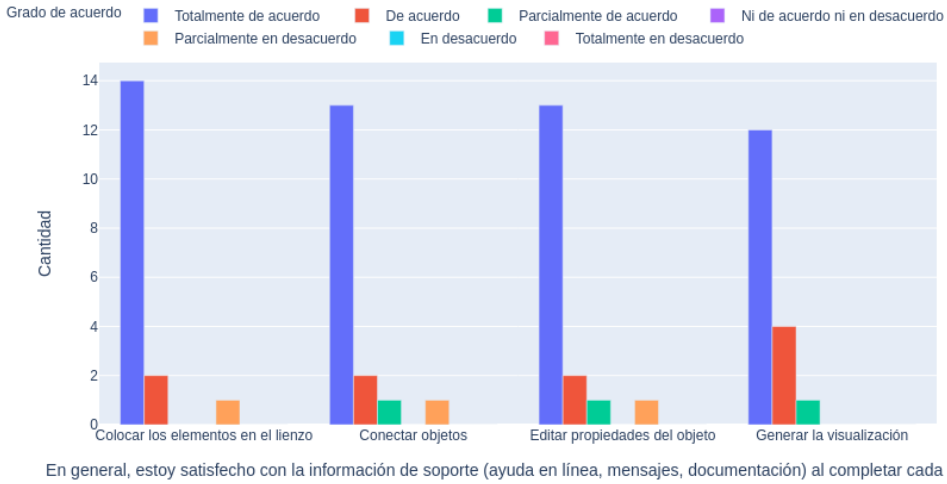


Figura 45: Grado de satisfacción con la información de soporte. *Fuente: Creación propia*.

Con respecto al grado de satisfacción al completar cada tarea, las tareas mejor y peor evaluadas fueron:

Mejor evaluada:

La tarea mejor evaluada fue: *Colocar los elementos en el lienzo*, en particular en el ítem 'En general, estoy satisfecho con la cantidad de tiempo que tomó completar cada tarea' (ver figura 44) donde el grado de acuerdo de todos los encuestados fue *Totalmente de acuerdo*.

Peor evaluada:

Por otro lado, las tareas peor evaluadas fueron conectar objetos y editar propiedades del objeto, en particular en el ítem 'En general, estoy satisfecho con la información de soporte (ayuda en línea, mensajes, documentación) al completar cada tarea' (ver figura 45) donde un 76,5 % de los encuestados estuvo *Totalmente de acuerdo*, 11,8 % estuvo *De acuerdo*, 5,9 % estuvo *Parcialmente de acuerdo* y 5,9 % estuvo *Parcialmente en desacuerdo*.

Aunque el resultado obtenido es favorable en la tarea *Conectar objetos*, el 5,9 % de los encuestados manifestó estar *Parcialmente en desacuerdo*. Es probable

que este resultado fuera obtenido porque los gráficos están asociados a un grupo de sensores y las tarjetas no. Un usuario comentó que esto fue confuso al principio (ver el Anexo D). Para solucionar esto, se podría añadir un mensaje indicando que no es posible conectar una tarjeta con un grupo de sensores cuando el usuario lo intente. Por otro lado, la característica de conectar un grupo de sensores con una tarjeta es algo que se puede implementar añadiendo una asociación de UserGroup a Card en el Meta-modelo mostrado en el Anexo A.

De igual manera, el resultado obtenido en la tarea *Editar propiedades del objeto* es favorable, aunque el 5,9 % de los encuestados estuvo *Parcialmente en desacuerdo*. Es probable que este resultado fuera obtenido porque las propiedades se deben editar en una 'pestaña de propiedades' de la aplicación y no en el mismo gráfico. Para solucionar esto, se puede añadir a cada elemento en el gráfico una ventana de edición al realizar la acción de doble click sobre él.

Satisfacción con respecto al sistema:

- En general, estoy satisfecho con lo fácil que es utilizar este sistema.

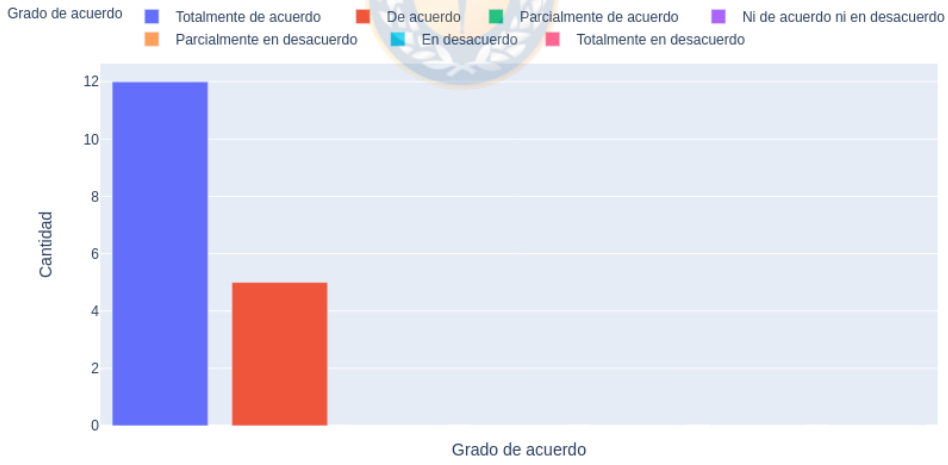


Figura 46: Grado de satisfacción con respecto a la facilidad de uso. *Fuente: Creación propia*.

- Fue sencillo utilizar este sistema.

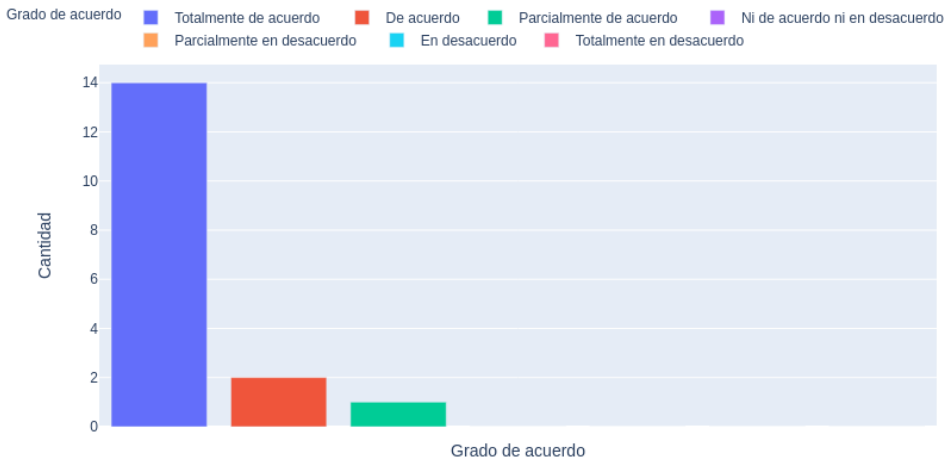


Figura 47: Grado de satisfacción con respecto a la sencillez de utilizar el sistema. *Fuente: Creación propia*.

- Pude completar de manera efectiva las tareas y escenarios usando este sistema.

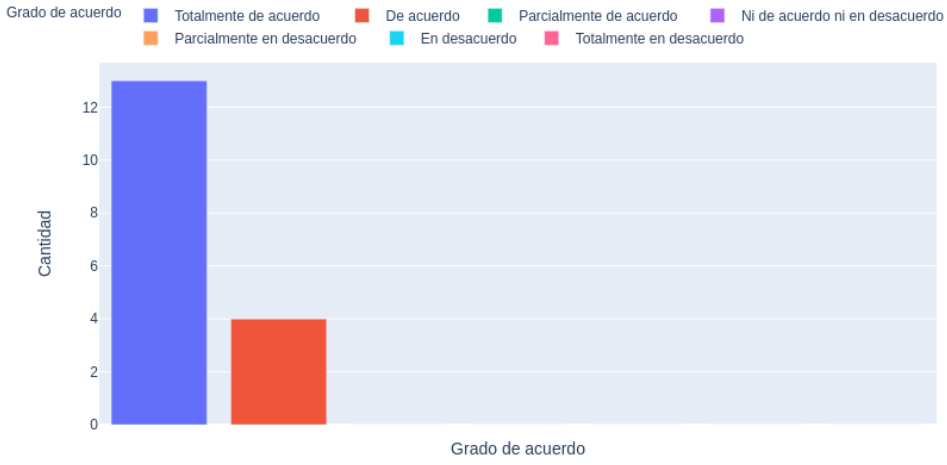


Figura 48: Grado de satisfacción con respecto a la efectividad al realizar los escenarios *Fuente: Creación propia*.

- Pude completar las tareas y escenarios rápidamente usando este sistema.

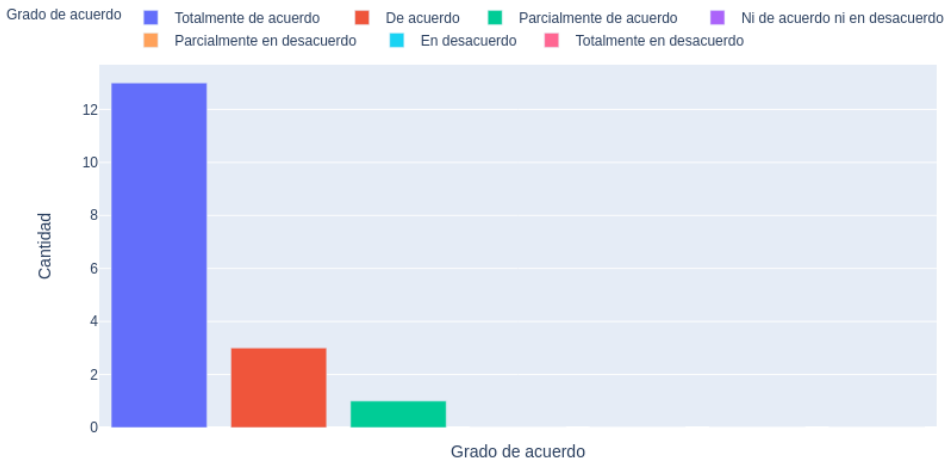


Figura 49: Grado de satisfacción con respecto a la rapidez al completar cada tarea. *Fuente: Creación propia*.

- Pude completar de manera eficiente las tareas y escenarios utilizando este sistema.

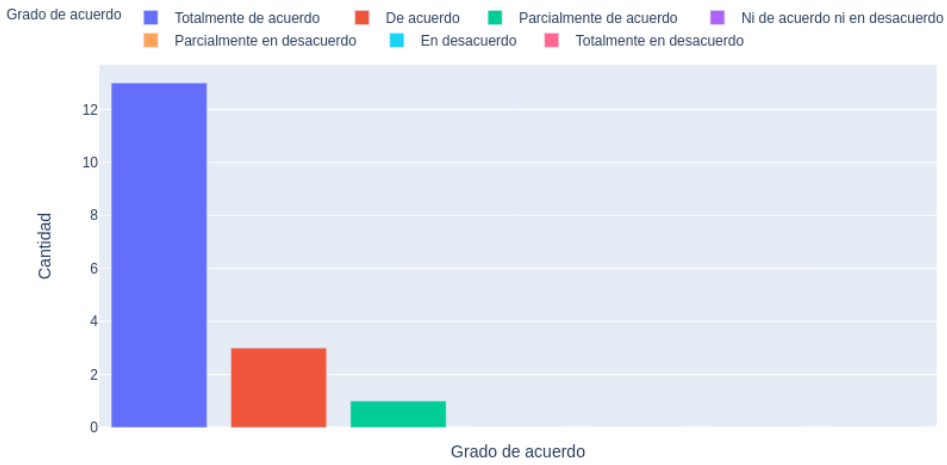


Figura 50: Grado de satisfacción con respecto a la eficiencia al completar los escenarios. *Fuente: Creación propia*.

- Me sentí cómodo o (cómoda) usando este sistema.

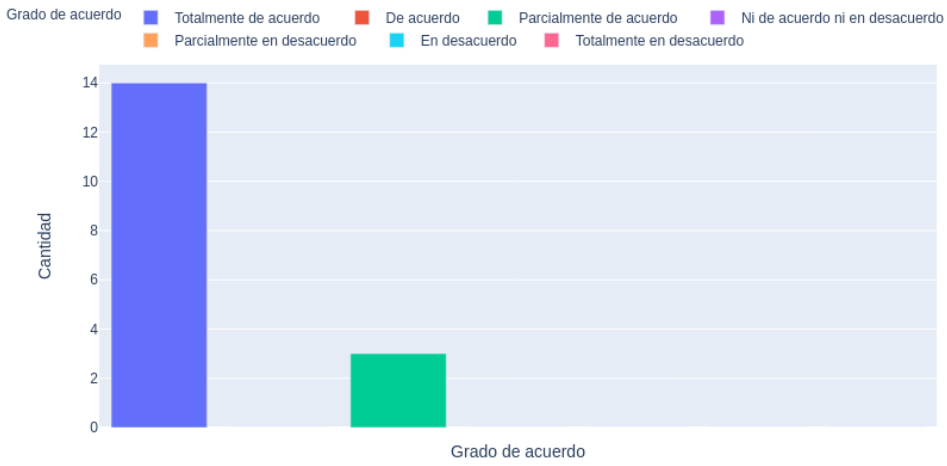


Figura 51: Grado de satisfacción con respecto a lo cómodo que se sintió el usuario al utilizar el sistema. *Fuente: Creación propia*.

- Fue fácil aprender a usar este sistema.

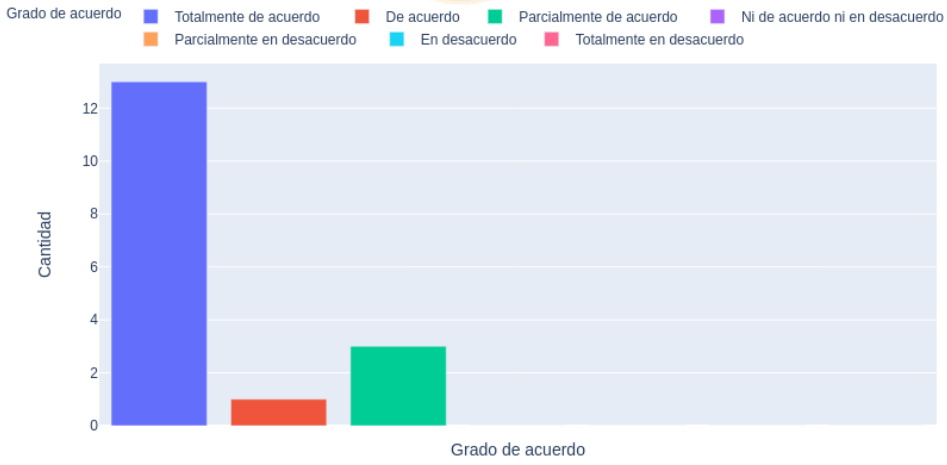


Figura 52: Grado de satisfacción con respecto a la facilidad de aprende a usar el sistema. *Fuente: Creación propia*.

- Creo que podría volverme productivo (o productiva) rápidamente usando este sistema.

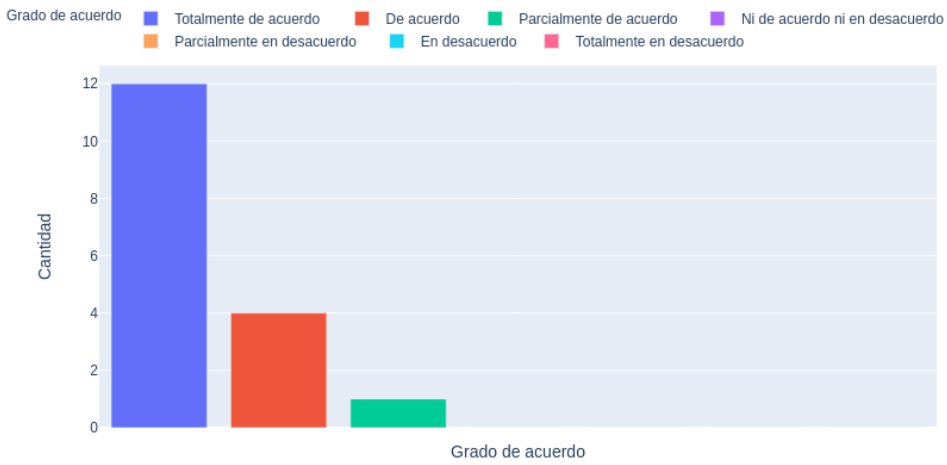


Figura 53: Grado de satisfacción con respecto a la productividad del usuario al utilizar este sistema. *Fuente: Creación propia*.

- El sistema dio mensajes de error que me indicaron claramente cómo solucionar problemas.

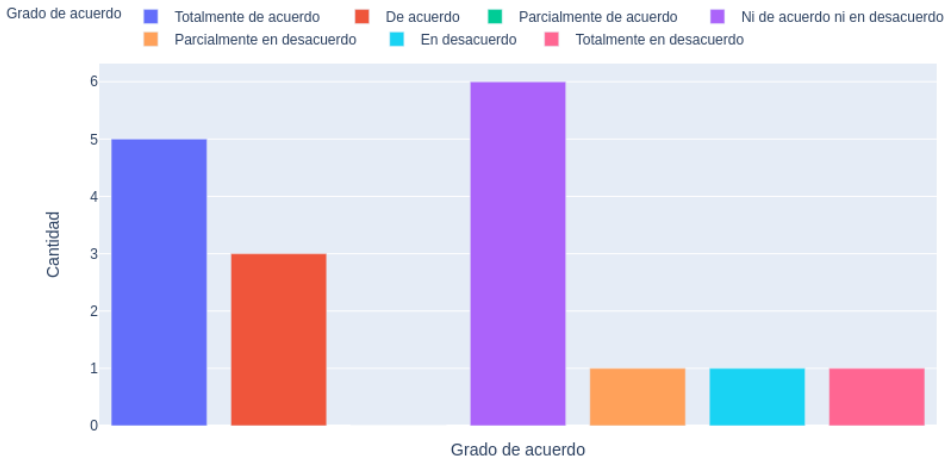


Figura 54: Grado de satisfacción con respecto a los mensajes de error que dió el sistema. *Fuente: Creación propia*.

- Siempre que cometía un error al utilizar el sistema, podía recuperarme fácil y rápidamente.

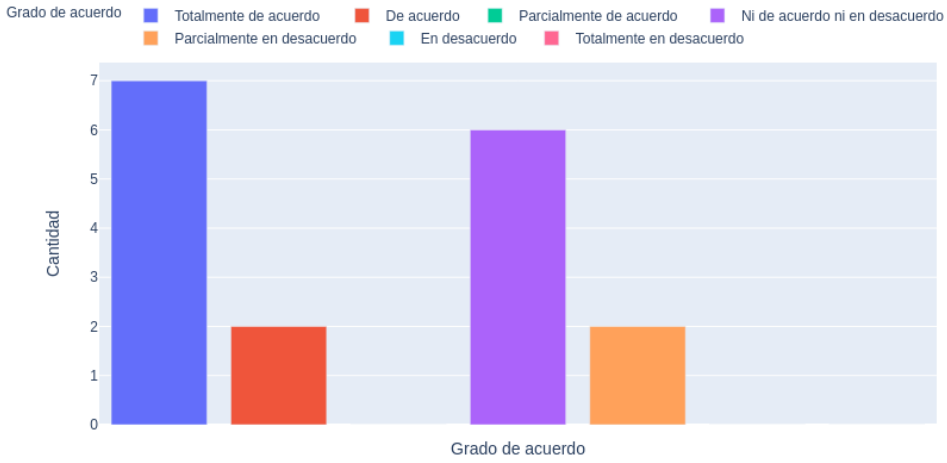


Figura 55: Grado de satisfacción con respecto a la recuperación de errores. *Fuente: Creación propia*.

- La información (como ayuda en línea, mensajes en pantalla y otra documentación) proporcionada con este sistema era clara.

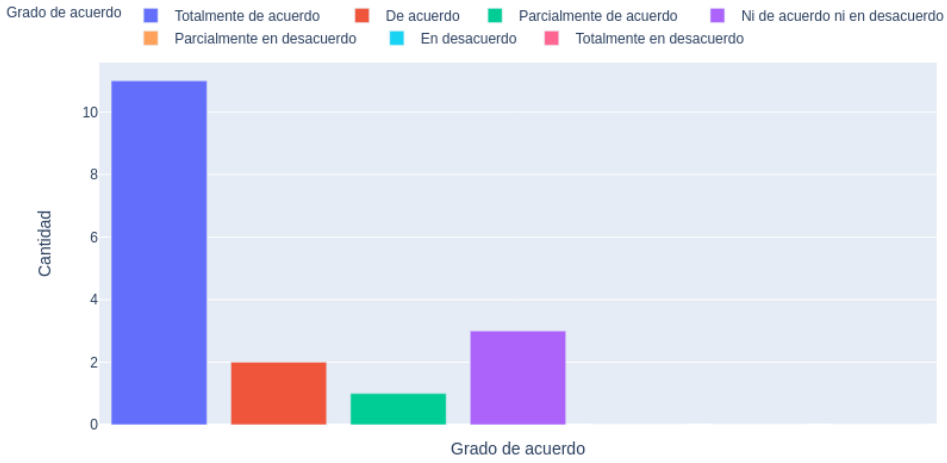


Figura 56: Grado de satisfacción con respecto a la información de soporte. *Fuente: Creación propia*.

- Fue fácil encontrar la información que necesitaba.

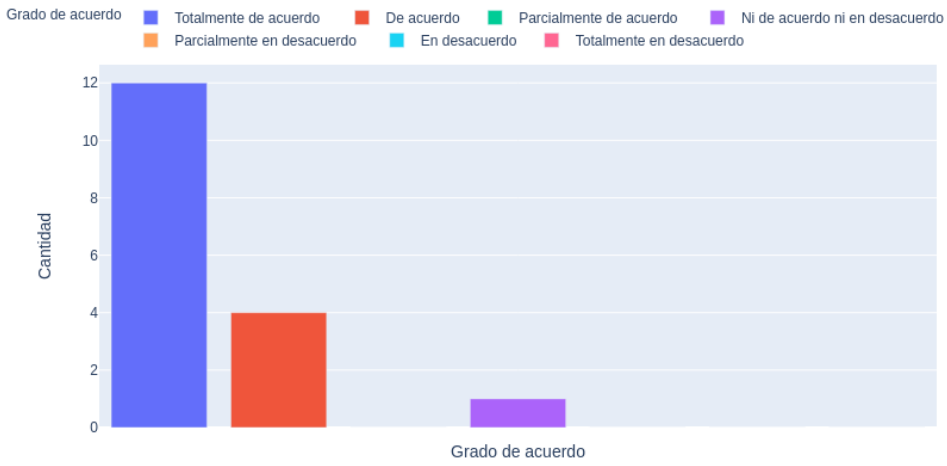


Figura 57: Grado de satisfacción con respecto a encontrar la información de soporte. *Fuente: Creación propia*.

- La información proporcionada para el sistema fue fácil de entender.

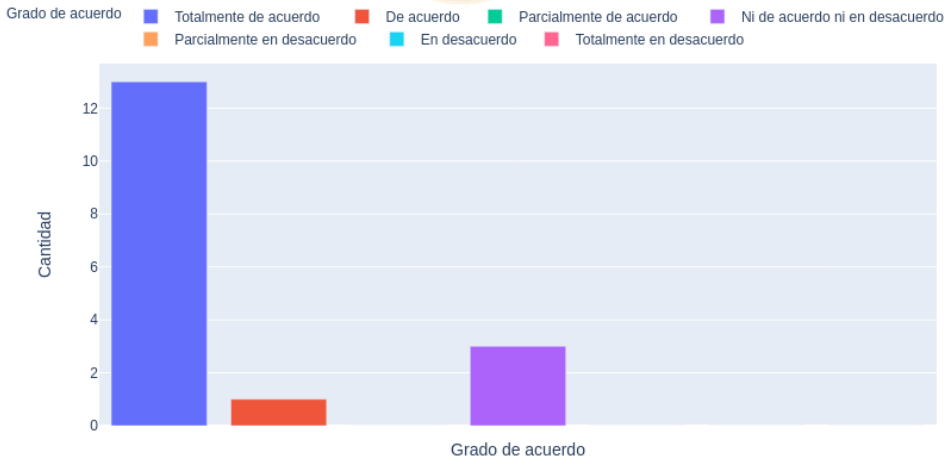


Figura 58: Grado de satisfacción con respecto a la facilidad de entender la información proporcionada para el sistema *Fuente: Creación propia*.

- La información fue eficaz para ayudarme a completar las tareas y los escenarios.

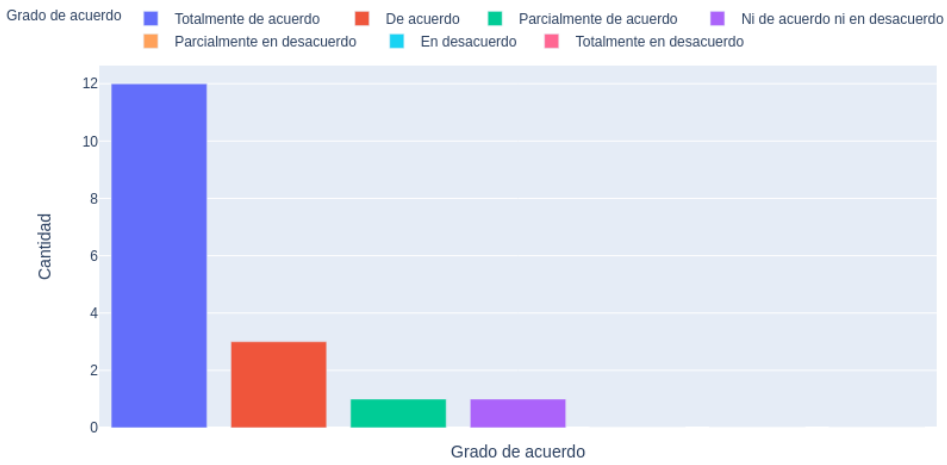


Figura 59: Grado de satisfacción con respecto a la eficacia de la información de soporte. *Fuente: Creación propia*.

- La organización de la información en las pantallas del sistema fue clara.

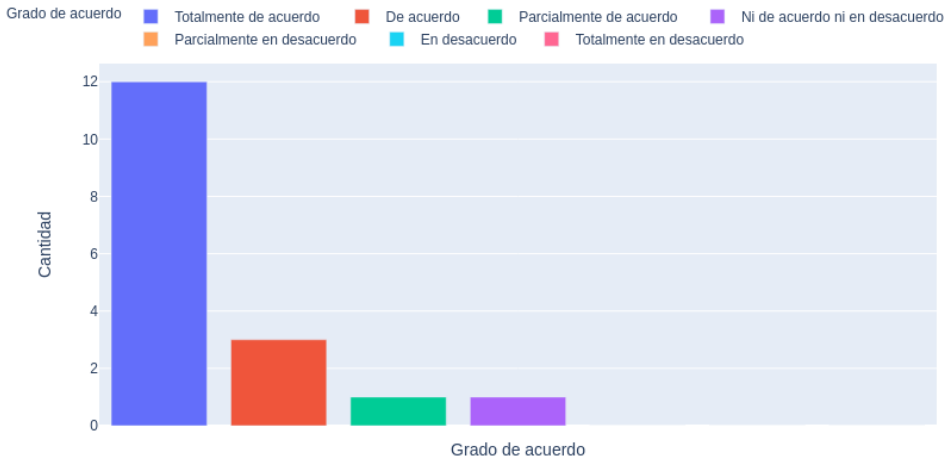


Figura 60: Grado de satisfacción con respecto a la organización de la información presentada por el sistema *Fuente: Creación propia*.

- La interfaz de este sistema fue agradable.

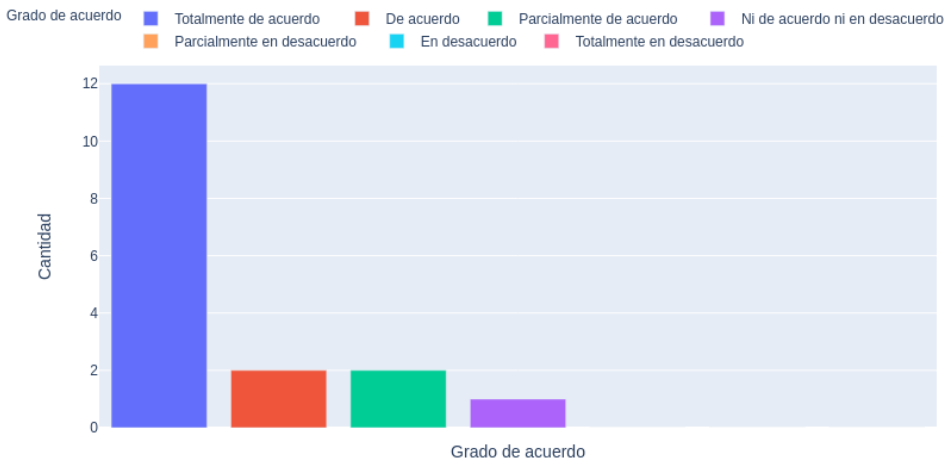


Figura 61: Grado de satisfacción con respecto a lo agradable de la interfaz del sistema *Fuente: Creación propia*.

- Me gustó usar la interfaz de este sistema.

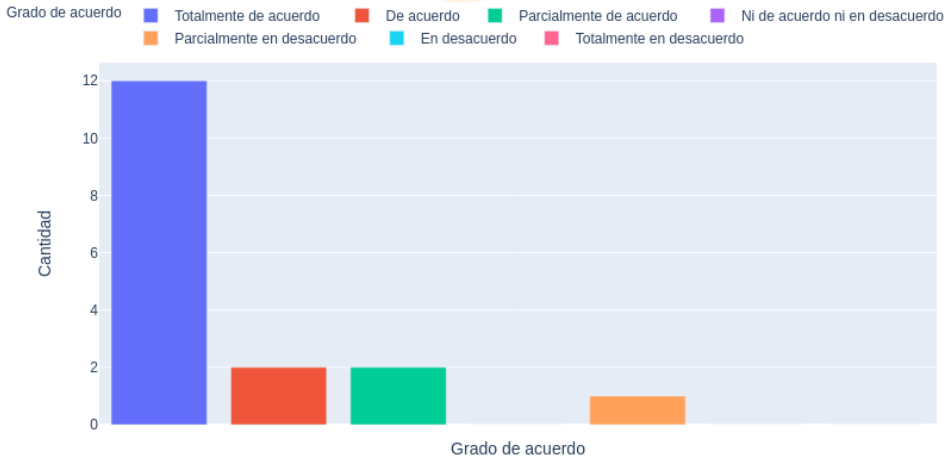


Figura 62: Grado de satisfacción con respecto al uso de la interfaz del sistema *Fuente: Creación propia*.

- Este sistema tiene todas las funciones y capacidades que espero que tenga.

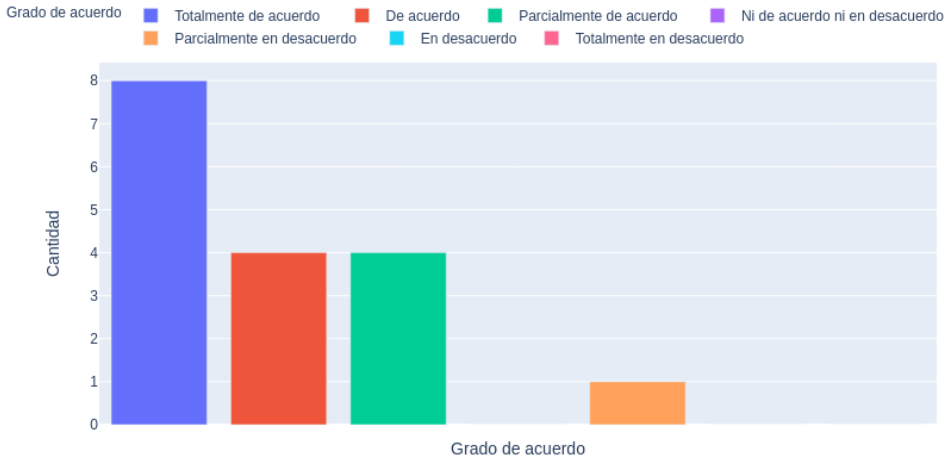


Figura 63: Grado de satisfacción con respecto a las funciones y capacidades del sistema. *Fuente: Creación propia*.

- En general, estoy satisfecho (o satisfecha) con este sistema.

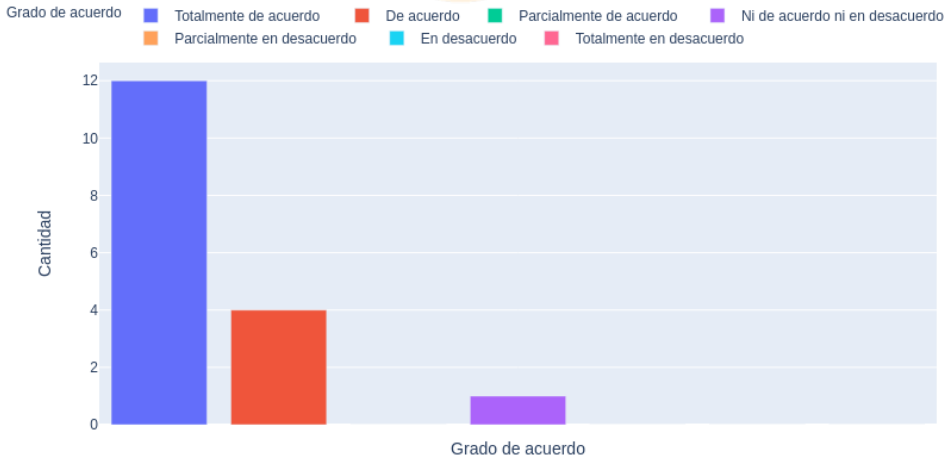


Figura 64: Grado de satisfacción general con respecto al sistema. *Fuente: Creación propia*.

Con respecto al grado de satisfacción sobre el sistema, lo mejor y peor evaluado fue:

Mejor evaluado:

El ítem 'Fue sencillo utilizar este sistema' fue el mejor evaluado, donde un 82,4 % de los usuarios está *Totalmente de acuerdo*, 11,8 % esta *De acuerdo* y 5,9 % está *Parcialmente de acuerdo*.
Peor evaluado:

El ítem 'El sistema dio mensajes de error que me indicaron claramente cómo solucionar problemas' fue el peor evaluado, donde un 29,4 % de los encuestados está *Totalmente de acuerdo*, 17,6 % está *De acuerdo*, 35,3 % respondió *Ni de acuerdo ni en desacuerdo*, un 5,9 % respondió *Parcialmente en desacuerdo*, un 5,9 % respondió *Desacuerdo* y un 5,9 % respondió *Totalmente en desacuerdo*. Con respecto a esta pregunta, varios de los usuarios comentaron que eligieron la opción *Ni de acuerdo ni en desacuerdo* debido a que nunca se presentó un error durante la realización de las pruebas (ver el Anexo D), un usuario reportó un error al arrastrar los elementos en el lienzo. Para mejorar este apartado se deben seguir realizando pruebas sobre el software en el entorno donde los usuarios reporten errores.

8.3. Comparativa con Alternativa de Código

Para realizar la comparativa de esta propuesta con una alternativa de código, se tomó como referencia el proyecto Generación de Reportes Dashboard para Sistemas de Monitoreo Estructural realizado por Diego Varas en el marco del proyecto FONDEF 'Plataforma de Monitoreo Estructural de Puentes' IT18I0112F, el proyecto mencionado aborda el desafío de la generación de reportes y visualización de datos para el Monitoreo Estructural de Puentes.

El tiempo estimado de desarrollo de la aplicación generada en el proyecto Generación de Reportes Dashboard para Sistemas de Monitoreo Estructural fue de 3 meses y 2 semanas. Este tiempo no incluye la elección de herramientas para el desarrollo de la aplicación. Dentro de este tiempo se incluye la generación de alternativas de visualización como linechart, boxplot, cards, OHCL y el diseño de la interfaz gráfica de usuario.

Aunque la comparación de tiempo de desarrollo no es fácil de realizar, para este caso es posible indicar que el tiempo que los desarrolladores demoran en realizar una visualización mediante la herramienta *vis4bridge* es: el tiempo de modelado

más el de compilación de la herramienta [18].

9. CONCLUSIONES Y TRABAJOS FUTUROS

El principal objetivo de este proyecto consiste en proponer un enfoque de Desarrollo de Software Dirigido por Modelos para la generación automática de alternativas de visualización para el Monitoreo de Salud Estructural de Puentes.

En esta sección se muestran las conclusiones y recomendaciones obtenidas a partir del desarrollo de los objetivos mencionados en en la Sección 1 del presente trabajo.

9.1. Definición de un Lenguaje Específico de Dominio

Para lograr esto, el primer objetivo planteado fue definir un Lenguaje Específico de Dominio (DSL) para visualizaciones de datos históricos capaz de permitir la generación de alternativas de visualización.

Esto fue logrado mediante un análisis del dominio de visualización de datos para el Monitoreo de Salud Estructural de Puentes, que se detalla en la Sección 5. Para realizar este análisis, los elementos del dominio fueron formulados mediante un modelo de características. Y luego cada elemento del dominio fue representado. Para los sensores se utilizó una representación física mientras que para las visualizaciones se utilizaron iconos mostrando la forma característica de la visualización representada.

Las reglas que definen al lenguaje se especifican a partir del Meta-modelo mostrado en la Sección 5 y en el Anexo A. Este modelo no solo representa las reglas actuales del DSL, sino que también propone reglas para la implementación de futuros elementos en las vistas (Tablas y filtros) y otros tipos de gráficos no representados en este trabajo.

[18] Puede revisar en: `https://drive.google.com/drive/folders/1RlKooivysKOctsdY3ExKrxWRyayGF_Vt?usp=sharing`. El enlace muestra 3 videos con la generación de alternativas de visualización que no superan los 4 minutos.

9.2. Definición de transformación del modelo a una alternativa de visualización

Luego de definir el DSL con el que los expertos en el dominio fueran capaces de representar las visualizaciones para el Monitoreo Estructural de Puentes, se definieron las transformaciones que generan las alternativas de visualización.

Como se indica en la Sección 6 para generar las salidas esperadas por los stakeholders se consideró para la elección del lenguaje de destino aspectos como bibliotecas de visualización disponibles en el lenguaje, conocimiento del lenguaje por expertos en el dominio (si bien, el código generado es automático, esto da la posibilidad a los expertos de modificar el código en caso de necesitarlo) y soporte para visualización en un entorno Web. En esta misma sección se puede observar la implementación de esta transformación que fue generada en base a dos lenguajes. Una implementación en Python mediante la biblioteca Dash y la otra en Javascript mediante la biblioteca ChartJS.

Con esto es muestra que:

- Mediante el Desarrollo Dirigido por Modelos es posible generar alternativas de visualización de datos históricos recientes para el Monitoreo de Salud Estructural de Puentes.
- Al obtener dos transformaciones mediante un mismo DSL se puede observar que este no está sujeto al lenguaje de destino que genera las visualizaciones de datos.
- Es posible generar una visualización totalmente automatizada.
- Como se muestra en las reglas implementadas (ver el Anexo C) la salida generada se encuentra ordenada y documentada para que el desarrollador de aplicaciones pueda utilizar el código disponible y hacer modificaciones que mejoren los requerimientos actuales o implementen nuevos requerimientos.

9.3. Comparación con respecto a desarrollo basado en la codificación

En la Sección 8 se mostró una comparativa del desarrollo tradicional frente al desarrollo propuesto en este trabajo. Se concluye que el tiempo de desarrollo mediante esta metodología fue inferior al tiempo empleado en un desarrollo basado en la codificación para los stakeholders. Además, existen ventajas en el uso de esta metodología que se detallan a continuación:

- El desarrollo basado en la codificación genera una alternativa de visualización fija, mientras que con el DSL el usuario final de la aplicación puede generar múltiples salidas como se muestra en la Sección 6.
- Una vez construido el DSL y la Transformación, generar una nueva alternativa de visualización solo dependerá del tiempo de compilación de la alternativa y no supondrá otro desarrollo (como en el caso del desarrollo basado en la codificación).
- El desarrollo dirigido por modelos no es incompatible con el desarrollo tradicional, las salidas generadas en este trabajo pueden ser modificadas por un desarrollador. Es decir, puede utilizar las salidas generadas por la herramienta para agilizar el desarrollo como fue planteado por stakeholders del proyecto.

9.4. Recomendaciones y Trabajos Futuros

Como se indicó en una de las conclusiones, el desarrollo dirigido por modelos no es incompatible con el desarrollo tradicional. Al observar que es posible generar código base que ayude a desarrolladores expertos en el lenguaje de destino, me lleva a plantear la posibilidad de utilizar el desarrollo dirigido por modelos como metodología para el desarrollo ágil de aplicaciones en el dominio de visualizaciones o la generación de aplicaciones web identificando los nuevos elementos que componen este dominio.

El meta-modelo está construido de forma modular. Por este motivo, es posible cambiar los sensores utilizados en puentes por sensores usados en otros dominios. Con esto, es posible extender este trabajo para ser utilizado en visualizaciones que pertenezcan a otros dominios distintos al Monitoreo de Salud Estructural de Puentes.

Durante el desarrollo de este proyecto también se observó que el experto en Desarrollo de Software Dirigido por Modelos (MDSD) no necesariamente debe ser especialista en el lenguaje de destino, por este motivo, el experto en MDSD debe generar este desarrollo en conjunto con especialistas en el Lenguaje de Destino. Así, los expertos en MDSD se encargarían de las etapas de Metamodelado, y generación del DSL mientras que la etapa de transformación de modelo a texto sería un trabajo conjunto con los expertos en lenguaje de salida. Estos últimos se encargarían de generar el API (Application Programming Interface) para los elementos del dominio representados en el lenguaje de destino y los expertos en MDSD se encargarían de utilizar esta API para generar los elementos de la transformación. Así, se evitaría que los especialistas en MDSD tengan

que estudiar a fondo las bibliotecas de destino y se concentren en la generación de código a partir del Meta-modelo y el DSL.

10. GLOSARIO

Daño Estructural: Cambios en el material o estructuras geométricas [6].

Desarrollo de Software Dirigido por Modelos: Área de la Ingeniería de Software en que los modelos del sistema se establecen con suficiente detalle para que la implementación completa del sistema pueda generarse a partir de los modelos [26].

Lenguaje Específico de Dominio: Lenguaje expresa la solución en el idioma y nivel de abstracción del experto en el dominio, esto permite que los expertos puedan entender por ellos mismos los programas basados en un DSL [27]. Puede ser expresado mediante texto o gráficos.

Meta-modelo: Define un lenguaje que describe la semántica del dominio [1]

Monitoreo de Salud Estructural: Disciplina que permite identificar el daño y prolongación de este en estructuras de ingeniería civil, aeroespacial y mecánica, donde el daño es definido como cambios en el material o estructuras geométricas [6].

Plataforma de Monitoreo Estructural de Puentes: Sistemas de Monitoreo de Salud Estructural de puentes, usualmente compuestos por una arquitectura capaz de soportar la adquisición, fusión y limpieza de datos, características de extracción y condensación de la información, además de modelos estadísticos para el análisis de datos [29].

Visualización de Datos: La visualización define cómo los modelos son representados y cómo todos aquellos interesados en el sistema tendrán interacción con él [30]. En el caso de la visualización de datos, define como los datos serán representados y como los usuarios tendrán interacción con ellos.

11. REFERENCIAS BIBLIOGRÁFICAS

Referencias

[1] Object Management Group (OMG), “Meta-modeling and the OMG meta object facility (MOF),” pp. 1–7, 2017.

[2] H.-c. Chung, T. Enomoto, M. Shinozuka, P. Chou, and C. Park, “13 th World Conference on Earthquake Engineering REAL TIME VISUALIZATION OF STRUCTURAL RESPONSE WITH WIRELESS MEMS SENSORS,” *Time*, no. 121, pp. 1–10, 2004.

[3] S. F. Masri, L.-H. Sheng, J. P. Caffrey, R. L. Nigbor, M. Wahbeh, and A. M. Abdel-Ghaffar, “Application of a Web-enabled real-time structural health monitoring system for civil infrastructure systems,” *Smart Materials and Structures*, vol. 13, pp. 1269–1283, dec 2004.

[4] S. L. Desjardins, N. A. Londoño, D. T. Lau, and H. Khoo, “Real-Time Data Processing, Analysis and Visualization for Structural Monitoring of the Confederation Bridge,” *Advances in Structural Engineering*, vol. 9, no. 1, pp. 141–157, 2006.

[5] M. Brambilla and E. Umuhoza, “Model-driven development of user interfaces for IoT systems via domain-specific components \& patterns,” in *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems*, vol. 2, pp. 246–253, SpringerOpen, dec 2017.

[6] C. R. Farrar and K. Worden, “An introduction to structural health monitoring,” *CISM International Centre for Mechanical Sciences, Courses and Lectures*, vol. 520, no. 1851, pp. 1–17, 2010.

[7] Y. Bao, Z. Chen, S. Wei, Y. Xu, Z. Tang, and H. Li, “The State of the Art of Data Science and Engineering in Structural Health Monitoring,” *Engineering*, vol. 5, pp. 234–242, jun 2019.

[8] D. Montalvão, N. M. M. Maia, and A. M. R. Ribeiro, “A review of vibration-based structural health monitoring with special emphasis on composite materials,” *Shock and Vibration Digest*, vol. 38, no. 4, pp. 295–324, 2006.

[9] F. Ruiz, M. Polo, M. Piattini, and G. Alarcos, “Utilización de Investigación-Acción en la Definición de un Entorno para la Gestión del Proceso de Mantenimiento del Software,” *. . . la Ingeniería del Software y . . .*, 2002.

[10] M. J. Grant and A. Booth, “A typology of reviews: An analysis of 14 review types and associated methodologies,” *Health Information and Libraries Journal*, vol. 26, no. 2, pp. 91–108, 2009.

[11] J. R. Lewis, “IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use,” *International Journal of Human-Computer Interaction*, vol. 7, no. 1, pp. 57–78, 1995.

[12] J. Nielsen, “10 usability heuristics for user interface design,” *Nielsen Norman Group*, vol. 1, no. 1, 1995.

[13] S. Beskhyroun, L. D. Wegner, and B. F. Sparling, “Field dynamic test and Bayesian modal identification of a special structure – the Palms Together Dagoba,” *Structural Control and Health Monitoring*, no. May 2011, pp. n/a–n/a, 2011.

[14] X. Hu, B. Wang, and H. Ji, “A Wireless Sensor Network-Based Structural Health Monitoring System for Highway Bridges,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 28, pp. 193–209, mar 2013.

[15] R. I. Bull, M. A. Storey, J. M. Favre, and M. Litoiu, “An architecture to support model driven software visualization,” *IEEE International Conference on Program Comprehension*, vol. 2006, pp. 100–104, 2006.

[16] M. Brambilla and P. Fraternali, “Large-scale Model-Driven Engineering of web user interaction: The WebML and WebRatio experience,” *Science of Computer Programming*, vol. 89, no. PART B, pp. 71–87, 2014.

[17] V. Dibia and Ç. Demiralp, “Data2Vis: Automatic Generation of Data Visualizations Using Sequence to Sequence Recurrent Neural Networks,” apr 2018.

[18] O. Abdeljaber, O. Avci, S. Kiranyaz, M. Gabbouj, and D. J. Inman, “Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks,” *Journal of Sound and Vibration*, vol. 388, pp. 154–170, 2017.

[19] F. G. Baptista, J. V. Filho, and D. J. Inman, “Real-time multi-sensors measurement system with temperature effects compensation for impedance-based structural health monitoring,” *Structural Health Monitoring*, vol. 11, no. 2, pp. 173–186, 2012.

[20] Y. Bao, Z. Chen, S. Wei, Y. Xu, Z. Tang, and H. Li, “The State of the Art of Data Science and Engineering in Structural Health Monitoring,” *Engineering*, vol. 5, pp. 234–242, jun 2019.

[21] D. Balageas, “Introduction to Structural Health Monitoring,” pp. 13–43, 2006.

[22] H.-N. Li, T.-H. Yi, L. Ren, D.-S. Li, and L.-S. Huo, “Reviews on innovations and applications in structural health monitoring for infrastructures,” *Structural Monitoring and Maintenance*, vol. 1, no. 1, pp. 1–45, 2015.

[23] I. Khemapech, W. Sansrimahachai, and M. Toahchoodee, “A real-time Health Monitoring and warning system for bridge structures,” in *TENCON 2016 - 2016 IEEE Region 10 Conference*, pp. 3010–3013, IEEE, jun 2016.

[24] M. Kurata, J. Kim, J. P. Lynch, G. W. van der Linden, H. Sedarat, E. Thometz, P. Hipley, and L.-H. Sheng, “Internet-Enabled Wireless Structural Monitoring Systems: Development and Permanent Deployment at the New Carquinez Suspension Bridge,” *Journal of Structural Engineering*, vol. 139, no. 10, pp. 1688–1702, 2013.

[25] I. Sommerville, *Ingeniería del software*. Pearson educación, 2005.

[26] S. Beydeda, M. Book, V. Gruhn, and Others, *Model-driven software development*, vol. 15. Springer, 2005.

[27] A. V. Deursen, P. Klint, and J. Visser, “Domain-Specific Languages : An Annotated Bibliography *,” vol. 35, no. June, pp. 26–36, 2000.

[28] S. Sendall and W. Kozaczynski, “Model transformation: The heart and soul of model-driven software development,” *IEEE software*, vol. 20, no. 5, pp. 42–45, 2003.

[29] H. Sohn, C. R. Farrar, F. Hemez, and J. Czarnecki, “A Review of structural health,” *Library.Lanl.Gov*, pp. 1–7, 2001.

[30] N. Medvidovic and R. N. Taylor, “Software architecture: foundations, theory, and practice,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pp. 471–472, ACM, 2010.

[31] W. Lidong, W. Guanghui, and A. Cheryl Ann, “Big Data and Visualization: Methods, Challenges and Technology Progress,” *Digital Technologies*, vol. 1, no. 1, pp. 33–38, 2015.

[32] Ministerio de Obras Públicas, “Planificación Mantenimiento Puentes y Caminos,” tech. rep., 2018.

[33] Ministerio de Obras Públicas, “Informe de Puentes Dirección de Vialidad.” https://www.mop.cl/Prensa/Paginas/InformePuentes.aspx, 2018.

[34] Televisión Nacional de Chile, “Colapso de puente Cancura deja un fallecido y seis heridos en Osorno,” jun 2018.

[35] R. S. Pressman, *Ingeniería del software. Un enfoque práctico*. McGraw Hill, 2002.

[36] D. Keim, H. Qu, and K. L. Ma, "Big-data visualization," *IEEE Computer Graphics and Applications*, vol. 33, no. 4, pp. 20–21, 2013.

[37] A. Rodrigues Da Silva, "Model-driven engineering: A survey supported by the unified conceptual model," *Computer Languages, Systems and Structures*, vol. 43, pp. 139–155, 2015.

[38] S. Li, H. Li, Y. Liu, C. Lan, W. Zhou, and J. Ou, "SMC structural health monitoring benchmark problem using monitored data from an actual cable-stayed bridge," *Structural Control and Health Monitoring*, vol. 21, no. 2, pp. 156–172, 2014.

12. ANEXOS

A. Meta-modelo

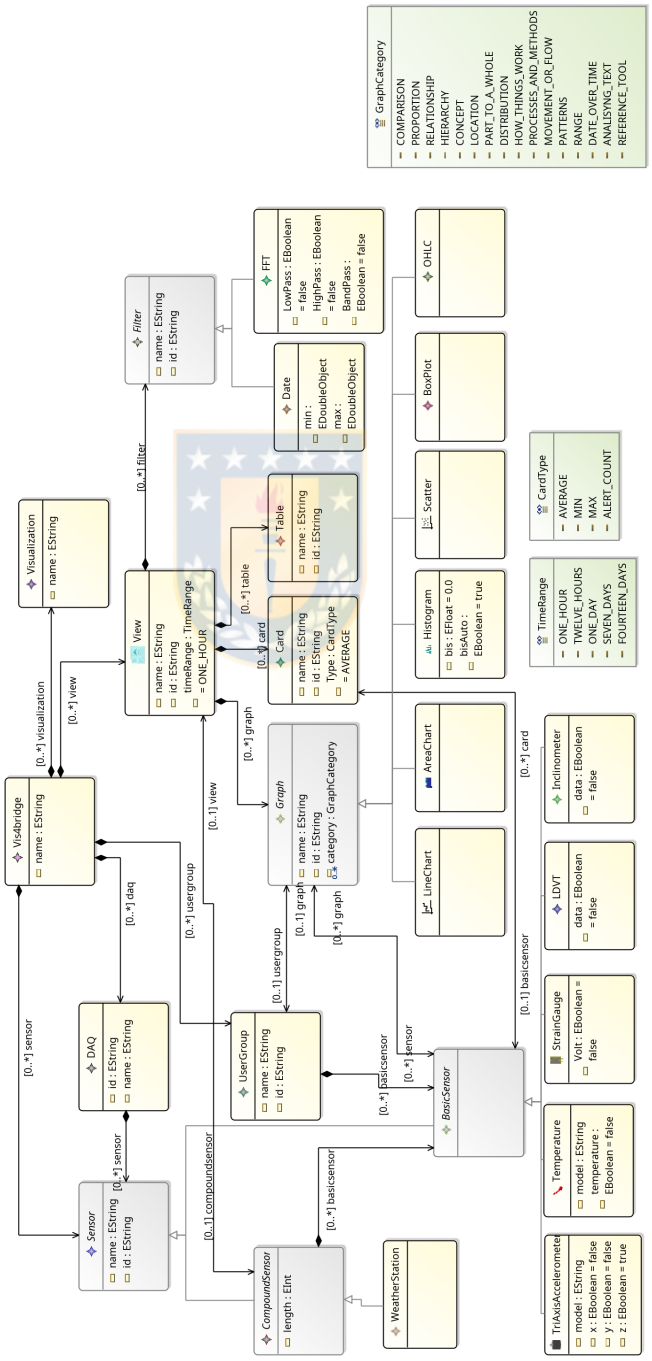


Figura 65: Meta-modelo generado con ecore tool. *Fuente: Creación propia*

B. Diagrama de Paquetes

B.1. Esquema de Transformación del Modelo a Python

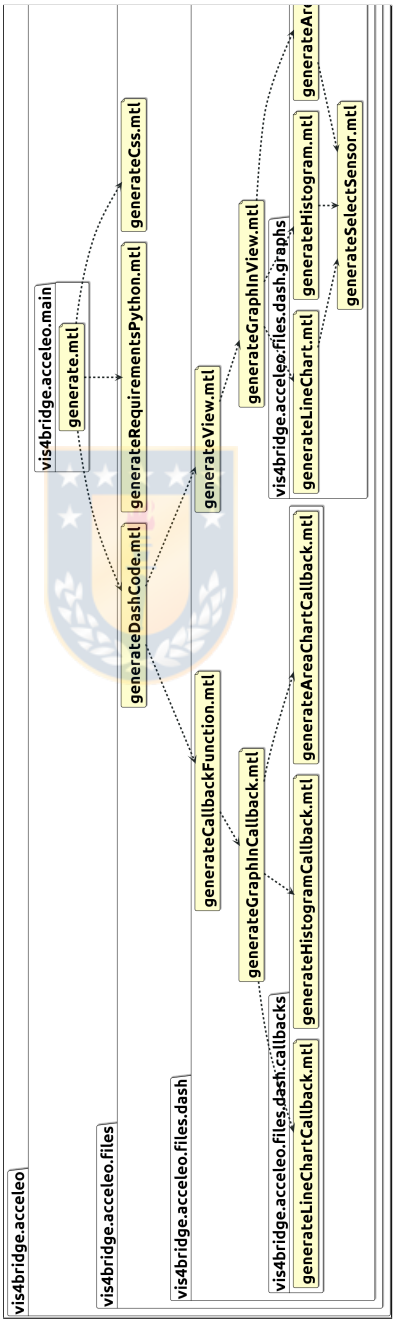


Figura 66: Diagrama de Paquetes de la transformación de Modelo a Python. *Fuente: Creación propia*

B.2. Esquema de Transformación del Modelo a Javascript

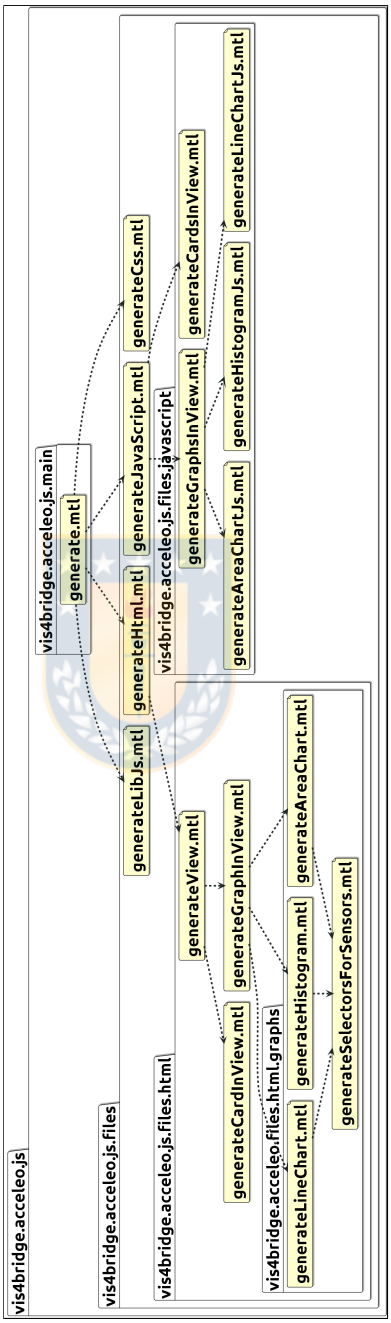


Figura 67: Diagrama de Paquetes de la transformación de Modelo a Javascript.
Fuente: Creación propia

C. Transformación de Modelo a Texto

Aquí se encuentra el código fuente Acceleo que genera la transformación de Modelo a Texto.

C.1. Transformación de Modelo a Python

C.1.1. Paquete vis4bridge.acceleo.main

Módulo generate

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generate.
4   */]
5  [module generate('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7  [import braulioqh::vis4bridge::acceleo::files::generateCss /]
8  [import braulioqh::vis4bridge::acceleo::files::generateDashCode /]
9  [**
10  * The documentation of the template generateElement.
11  * @param aVis4bridge
12  */]
13 [template public generateElement(aVis4bridge : Vis4bridge)]
14 [comment @main/]
15 [generateCss(aVis4bridge)/]
16 [generateDashCode(aVis4bridge)/]
17 [/template]
```

C.1.2. Paquete vis4bridge.acceleo.files

Módulo generateDashCode

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateDashCode.
4   */]
5  [module generateDashCode('https://gitlab.com/braulioqh/vis4bridge.git')]
6  [import braulioqh::vis4bridge::acceleo::files::dash::generateView /]
7  [import braulioqh::vis4bridge::acceleo::files::dash::generateCallbackFunction /]
```

```
8  [**
9   * The documentation of the template generateDashCode.
10  * @param aVis4bridge
11  */]
12 [template public generateDashCode(aVis4bridge : Vis4bridge)]
13 [file ('app.py', false, 'UTF-8')]
14 [initialStaticContent(aVis4bridge)/]
15
16 # The component tree is defined using app.layout
17 app.layout = html.Div(children=['['/]
18     html.H1(children='Bridge DashBoard'),
19
20 [for (itView : View | self.view)]
21 [generateView(itView)/]
22 [/for]
23 [']'/],
24 id='visualization')
25
26 [for (itView : View | self.view)]
27 [generateCallbackFunction(itView)/]
28 [/for]
29
30 # Allows to run the application
31 if __name__ == '__main__':
32     app.run_server(debug=True)
33 [/file]
34 [/template]
35 [template private initialStaticContent(aVis4bridge : Vis4bridge)]
36     # -*- coding: utf-8 -*-
37 # ---------- Bibliotecas ----------------
38 # base library for dash
39 import dash
40 import dash_core_components as dcc #contains base dash components
41 import dash_html_components as html #Contains the html tags
42 import plotly.express as px
43 import plotly.graph_objs as go
44 import pandas as pd #For data management
45 from dash.dependencies import Input, Output #For callbacks
46 import numpy as np #For data processing. ex: To pass data from matlab to
       pandas
47 from scipy.io import loadmat as lm #To load matlab data
48 import calendar #For handling dates
```

```
49 import psycopg2 #For connection to the database
50 import psycopg2.extras
51 import getpass #For entering passwords
52
53
54 # ---------- EXTERNAL CSS ---------------
55 # external css
56 external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
57
58 #------------ LOADING DATA-------------
59 #The data that will be processed with pandas is loaded
60 #mat = lm('/home/braulio/Documentos/workspace/python/pruebasDash/
       d_08_2_1_1.mat')
61 #df = pd.DataFrame(np.stack((mat['Data'])))
62 #df['time']= pd.to_datetime(df['time'], format='%Y%m%d%H%M%S'
       )
63
64 # connection to the database
65
66 pw = getpass.getpass();
67 # establish connection
68 conn = psycopg2.connect(dbname='tutorial', user='postgres', password=pw,
       host='localhost')
69
70 # cursor object allows querying of database
71 # server-side cursor is created to prevent records to be downloaded until
       explicitly fetched
72 cursor_datasetBridge = conn.cursor('datasetBridge', cursor_factory=psycopg2.
       extras.DictCursor)
73 cursor_datasetBridge.execute("SELECT * FROM datasetPuente WHERE time
       BETWEEN '2008-03-31 23:00' and '2008-03-31 23:59'")
74
75 # fetch records from database
76 ride_length_datasetBridge = cursor_datasetBridge.fetchall()
77
78 df = pd.DataFrame(np.array(ride_length_datasetBridge),
79  columns = ['time', 'sensor_01','sensor_02','sensor_03','sensor_04','sensor_05',
       'sensor_06','sensor_07','sensor_08','sensor_09','sensor_10','sensor_11','
       sensor_12','sensor_13','sensor_14','sensor_15','sensor_16'])
80
81
82 daterange = pd.date_range(start=df['time'].min(),end=df['time'].max(),
```

```
    freq='T')
sensor=list(df)['[']1:[']']

#-------------- INIT APP ------------------
# The app is initialized
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

#---------- LOCAL FUNCTIONS ------------
#general functions
def unixTimeMillis(dt):
    ''' Convert datetime to unix timestamp '''
    return calendar.timegm(dt.timetuple())

def unixToDatetime(unix):
    ''' Convert unix timestamp to datetime. '''
    return pd.to_datetime(unix,unit='s',origin='unix')

def getMarks(start, end, Nth=100):
    ''' Returns the marks for labeling.
        Every Nth value will be used.
    '''

    result = {}
    for i, date in enumerate(daterange):
        if(i%Nth == 1):
            # Append value to dict
            result['[']unixTimeMillis(date)[']'] = str(date.strftime('%Y-%m-%d'))

    return result
[/template]
```

Módulo generateCss

```
[comment encoding = UTF-8 /]
[**
 * The documentation of the module generateCss.
 */]
[module generateCss('https://gitlab.com/braulioqh/vis4bridge.git')]


[**
```

```
 * The documentation of the template generateCss.
 * @param aVis4bridge
 */]
[template public generateCss(aVis4bridge : Vis4bridge)]
[file ('assets/custom.css', false, 'UTF-8')]
/*

CSS file for app.py
You can modify or add new CSS code in order to customize style for your app

*/
/*Body background #BBBBBB*/
body{
  background: #F9F9F9;

}
H1{
    text-align: center;
}
H2{
    text-align: center;
}
.sensor{
  display: inline-block;
  width: 110px;
}
/*
.view-graph{
  background:#F9F9F9;
  display: flex;
  flex-wrap: wrap;
}

*/

.view-graph{
  display: grid;
  grid-template-columns: 1fr 1fr;

}
.graph{
  margin: 2%;
```

```
51   padding: 2%;
52   height: auto;
53   background:#DEDEDE;
54  }
55
56  /*
57  .graph{
58
59   width: 40%;
60   margin: 2%;
61   padding: 10px;
62   background:#FFFFFF;
63   flex-grow:1;
64   -webkit-box-shadow: 6px 6px 5px 0px rgba(0,0,0,0.3);
65   -moz-box-shadow: 6px 6px 5px 0px rgba(0,0,0,0.3);
66   box-shadow: 6px 6px 5px 0px rgba(0,0,0,0.3);
67
68  }*/
69  /*
70  #div-histogram-3{
71   display: inline-block;;
72   width: 40%;
73   margin: 2%;
74   padding: 5px;
75   background:#FFFFFF;
76
77  }
78  */
79
80
81  [/file]
82  [/template]
```

Módulo generateRequirementsPython

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateRequirementsPython.
4   */]
5  [module generateRequirementsPython('https://gitlab.com/braulioqh/vis4bridge.
       git')]
```

```
6
7
8  [**
9   * The documentation of the template generateRequirementsPython.
10  * @param aVis4bridge
11  */]
12 [template public generateRequirementsPython(aVis4bridge : Vis4bridge)]
13
14 [file ('requirements.txt', false, 'UTF-8')]
15 attrs==19.3.0
16 backcall==0.1.0
17 bleach==3.1.0
18 Click==7.0
19 cycler==0.10.0
20 dash==1.4.1
21 dash-core-components==1.3.1
22 dash-daq==0.2.1
23 dash-html-components==1.0.1
24 dash-renderer==1.1.2
25 dash-table==4.4.1
26 decorator==4.4.1
27 defusedxml==0.6.0
28 entrypoints==0.3
29 Flask==1.1.1
30 Flask-Compress==1.4.0
31 future==0.18.1
32 importlib-metadata==1.3.0
33 ipykernel==5.1.3
34 ipython==7.10.1
35 ipython-genutils==0.2.0
36 ipywidgets==7.5.1
37 itsdangerous==1.1.0
38 jedi==0.15.1
39 Jinja2==2.10.3
40 jsonschema==3.2.0
41 jupyter==1.0.0
42 jupyter-client==5.3.4
43 jupyter-console==6.0.0
44 jupyter-core==4.6.1
45 kiwisolver==1.1.0
46 MarkupSafe==1.1.1
47 matplotlib==3.1.2
```

```
mistune==0.8.4
more-itertools==8.0.2
mpmath==1.1.0
nbconvert==5.6.1
nbformat==4.4.0
nose==1.3.7
notebook==6.0.2
numpy==1.17.3
pandas==0.25.2
pandocfilters==1.4.2
parso==0.5.1
pexpect==4.7.0
pickleshare==0.7.5
pkg-resources==0.0.0
plotly==4.1.0
prometheus-client==0.7.1
prompt-toolkit==2.0.10
psycopg2==2.8.4
ptyprocess==0.6.0
Pygments==2.5.2
pyparsing==2.4.5
pyrsistent==0.15.6
python-dateutil==2.8.0
pytz==2019.3
pyzmq==18.1.1
qtconsole==4.6.0
retrying==1.3.3
scipy==1.3.3
scipy-stack==0.0.5
Send2Trash==1.5.0
six==1.12.0
sympy==1.4
terminado==0.8.3
testpath==0.4.4
tornado==6.0.3
traitlets==4.3.3
wcwidth==0.1.7
webencodings==0.5.1
Werkzeug==0.16.0
widgetsnbextension==3.5.1
zipp==0.6.0

```

```
90 [/file]
91 [/template]
```

C.1.3. Paquete vis4bridge.acceleo.files.dash

Módulo generateView

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateView.
4   */]
5  [module generateView('https://gitlab.com/braulioqh/vis4bridge.git')]
6  [import braulioqh::vis4bridge::acceleo::files::dash::generateGraphInView /]
7
8  [**
9   * The documentation of the template generateView.
10  * @param aView
11  */]
12 [template public generateView(aView : View)]
13    #A Div is generated for each view
14    html.Div(['']
15       html.H2(children='''
16 ␣␣␣␣␣␣␣␣␣␣[aView.name/]
17 ␣␣␣␣␣␣␣''').
18       html.Div(['']
19 [if (aView.graph->notEmpty())]
20    [for (itGraph : Graph | graph)]
21       [generateGraphInView(itGraph)/]
22    [/for]
23 [/if]
24       [']'/],
25       className='view-graph',
26       id='graph-[aView.id/]').
27    [']'/],
28    className='view',
29    id='div-[aView.id/]').
30 [/template]
```

Módulo generateGraphInView

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateGraphInView.
4   */]
5  [module generateGraphInView('https://gitlab.com/braulioqh/vis4bridge.git')]
6  [import braulioqh::vis4bridge::acceleo::files::dash::graphs::generateLineChart /]
7  [import braulioqh::vis4bridge::acceleo::files::dash::graphs::generateAreaChart /]
8  [import braulioqh::vis4bridge::acceleo::files::dash::graphs::generateHistogram /]
9
10 [**
11  * The documentation of the template generateElement.
12  * @param aGraph
13  */]
14 [template public generateGraphInView(aGraph : Graph)]
15
16 #A div is generated for the chart [aGraph.id/](contains the chart and associated
       widget)
17         html.Div([']
18 [if (aGraph.eClass().name.equalsIgnoreCase('LineChart'))]
19     [generateLineChart(aGraph)/]
20 [elseif(aGraph.eClass().name.equalsIgnoreCase('AreaChart'))]
21     [generateAreaChart(aGraph)/]
22 [elseif(aGraph.eClass().name.equalsIgnoreCase('Histogram'))]
23     [generateHistogram(aGraph)/]
24 [/if]
25         [']'/],
26         className='[aGraph.eClass().name/] graph',
27         id='div-[aGraph.id/]'),
28 [/template]
```

Módulo generateCallbackFunction

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateCallbackFunction.
4   */]
5  [module generateCallbackFunction('https://gitlab.com/braulioqh/vis4bridge.git')]
6  [import braulioqh::vis4bridge::acceleo::files::dash::generateGraphInCallback /]
7
8  [**
```

```
 * The documentation of the template generateCallbackFunction.
 * @param aView
 */]
[template public generateCallbackFunction(aView : View)]
[if (aView.graph->notEmpty())]
    [for (itGraph : Graph | graph)]
[generateGraphInCallback(itGraph)/]
    [/for]
[/if]
[/template]
```

Módulo generateGraphInCallback

```
[comment encoding = UTF-8 /]
[**
 * The documentation of the module generateGraphInCallback.
 */]
[module generateGraphInCallback('https://gitlab.com/braulioqh/vis4bridge.git')]
[import braulioqh::vis4bridge::acceleo::files::dash::callbacks::
    generateLineChartCallback /]
[import braulioqh::vis4bridge::acceleo::files::dash::callbacks::
    generateAreaChartCallback /]
[import braulioqh::vis4bridge::acceleo::files::dash::callbacks::
    generateHistogramCallback /]

[**
 * The documentation of the template generateGraphInCallback.
 * @param aGraph
 */]
[template public generateGraphInCallback(aGraph : Graph)]
[if (aGraph.eClass().name.equalsIgnoreCase('LineChart'))]
[generateLineChartCallback(aGraph)/]

[elseif(aGraph.eClass().name.equalsIgnoreCase('AreaChart'))]
[generateAreaChartCallback(aGraph)/]

[elseif(aGraph.eClass().name.equalsIgnoreCase('Histogram'))]
[generateHistogramCallback(aGraph)/]

[/if]

```

```
26 [/template]
```

C.1.4. Paquete vis4bridge.acceleo.files.dash.graphs

Módulo generateSelectSensor

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateSelectSensor.
4   */]
5  [module generateSelectSensor('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7
8  [**
9   * The documentation of the template generateSelectSensor.
10  * @param anUserGroup
11  */]
12 [template public generateSelectSensor(anUserGroup : UserGroup)]
13    [if (anUserGroup.basicsensor->notEmpty())]
14          # A Div is generated for the selection of sensors
15          html.Div(['']
16                [for (itSensor : BasicSensor | self.basicsensor)]
17             #Se genera un Dropdown por cada sensor
18             html.Div(['']
19                dcc.Dropdown(
20                id='dropdown-[itSensor.id/]',
21                options=['']{'label':str(i),'value': i} for i in sensor['']''],
22                value='sensor_0[i/]'
23                ),
24             [']''],
25             id='div-[itSensor.id/]',
26             className='sensor'),
27                [/for]
28          [']''],
29          id='div-[anUserGroup.id/]',
30          className='group'),
31    [/if]
32 [/template]
```

Módulo generateLineChart

```
[comment encoding = UTF-8 /]
[**
 * The documentation of the module generateLineChart.
 */]
[module generateLineChart('https://gitlab.com/braulioqh/vis4bridge.git')]
[import braulioqh::vis4bridge::acceleo::files::dash::graphs::
    generateSelectSensor /]

[**
 * The documentation of the template generateElement.
 * @param aGraph
 */]
[template public generateLineChart(aGraph : Graph)]
[if (aGraph.usergroup->notEmpty())]
  [for (itUsergroup : UserGroup | self.usergroup)]
    [generateSelectSensor(itUsergroup)/]
  [/for]
[/if]
      # Div to generate chart and range selection bar
      html.Div(['']
          dcc.Graph(
              id='[aGraph.id/]',
              config=dict(responsive=False),
          ),
          dcc.RangeSlider(
              count=1,
              min = unixTimeMillis(daterange.min()),#unix_time_millis(time.min()),
              max = unixTimeMillis(daterange.max()),#unix_time_millis(time.max()),
              value=['']unixTimeMillis(daterange.min()),unixTimeMillis(daterange.
    max())['']],
              marks=getMarks(daterange.min(),daterange.max()),
              id='rangeSlider-[aGraph.id/]'
          ),
      ['']]),
[/template]
```

Módulo generateAreaChart

```
[comment encoding = UTF-8 /]
[**
```

```
 * The documentation of the module generateAreaChart.
 */]
[module generateAreaChart('https://gitlab.com/braulioqh/vis4bridge.git')]
[import braulioqh::vis4bridge::acceleo::files::dash::graphs::
    generateSelectSensor /]

[**
 * The documentation of the template generateElement.
 * @param aGraph
 */]
[template public generateAreaChart(aGraph : Graph)]
[if (aGraph.usergroup->notEmpty())]
  [for (itUsergroup : UserGroup | self.usergroup)]
    [generateSelectSensor(itUsergroup)/]
  [/for]
[/if]
      # Div to generate chart and range selection bar
      html.Div(['['/]
          dcc.Graph(
              id='[aGraph.id/]',
              config=dict(responsive=False),
          ),
          dcc.RangeSlider(
              count=1,
              min = unixTimeMillis(daterange.min()),#unix_time_millis(time.min()),
              max = unixTimeMillis(daterange.max()),#unix_time_millis(time.max()),
              value=['['/]unixTimeMillis(daterange.min()),unixTimeMillis(daterange.
    max())[']'/],
              marks=getMarks(daterange.min(),daterange.max()),
              id='rangeSlider-[aGraph.id/]'
          ),
       [']'/]),
[/template]
```

Módulo generateHistogram

```
[comment encoding = UTF-8 /]
[**
 * The documentation of the module generateLineChart.
 */]
[module generateHistogram('https://gitlab.com/braulioqh/vis4bridge.git')]
```

```
6  [import braulioqh::vis4bridge::acceleo::files::dash::graphs::
       generateSelectSensor /]
7
8  [**
9   * The documentation of the template generateElement.
10  * @param aGraph
11  */]
12 [template public generateHistogram(aGraph : Graph)]
13 [if (aGraph.usergroup->notEmpty())]
14    [for (itUsergroup : UserGroup | self.usergroup)]
15       [generateSelectSensor(itUsergroup)/]
16    [/for]
17 [/if]
18             html.Div(['['/]
19                html.Div(['['/]
20                   dcc.Slider(
21                      id='bin-slider-[aGraph.id/]',
22                      min=1,
23                      max=40,
24                      step=1,
25                      value=12,
26                      updatemode='drag',
27                      marks={
28                         10:{'label': '10'},
29                         20:{'label': '20'},
30                         30:{'label': '30'},
31                         40:{'label': '40'},
32                      },
33                   ),
34                ']'/]),
35                html.Div(['['/]
36                   dcc.Checklist(
37                      id='bin-auto-[aGraph.id/]',
38                      options=['['/]
39                         {'label':'Auto','value':'Auto'}
40                      ']'/],
41                      value=['['/]'Auto'[']'/],
42                      inputClassName='auto__checkbox',
43                      labelClassName='auto__label',
44                   ),
45                   html.P(
46                         '# of Bins: Auto',
```

```
47                      id='bin-size-[aGraph.id/]',
48                      className='auto--p',
49                      ),
50              ['']'/'],style={'display':'inline'}),
51              html.Div(['['/']
52                  dcc.Graph(
53                      id='[aGraph.id/]',
54                      config=dict(responsive=False),
55                  ),
56                  dcc.RangeSlider(
57                              count=1,
58                              min = unixTimeMillis(daterange.min()),#unix_time_millis(
    time.min()),
59                              max = unixTimeMillis(daterange.max()),#unix_time_millis(
    time.max()),
60                              value=['['/']unixTimeMillis(daterange.min()),unixTimeMillis(
    daterange.max())[']'/'],
61                              marks=getMarks(daterange.min(),daterange.max()),
62                              id='rangeSlider-[aGraph.id/]'
63                          ),
64              [']'/']),
65          [']'/'],className='Div-slider',
66          style={'display':'block'}),
67  [/template]
```

C.1.5. Paquete vis4bridge.acceleo.files.dash.callbacks

Módulo generateLineChartCallback

```
1   [comment encoding = UTF-8 /]
2   [**
3    * The documentation of the module generateLineChartCallback.
4    */]
5   [module generateLineChartCallback('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7
8   [**
9    * The documentation of the template generateLineChartCallback.
10   * @param aGraph
11   */]
12  [template public generateLineChartCallback(aGraph : Graph)]
```

```
@app.callback(
    Output(component_id='[aGraph.id/]', component_property='figure'),
    ['['/']
    [for (itUserGroup : UserGroup | self.usergroup)]
        [for (itSensor : Sensor | self.basicsensor)]
     Input(component_id='dropdown-[itSensor.id/]', component_property='value'),
        [/for]
    [/for]
     Input(component_id='rangeSlider-[aGraph.id/]', component_property='value')[
      ']'/']
)
def update_grahp([for (itUserGroup : UserGroup | self.usergroup)] [for (itSensor
      : Sensor | self.basicsensor)]_[itSensor.id.replaceAll('-', '_')/],[/for][/for]_[
      aGraph.id.replaceAll('-', '_')/]):

    mask=(df['['/']'time'[']'/']>=unixToDatetime(_[aGraph.id.replaceAll('-', '_')/][']
      '/]).tz_localize('Etc/GMT+4') & (df['['/']'time'[']'/']<=unixToDatetime(_[aGraph.id.
      replaceAll('-', '_')/][']'/']1[']'/']).tz_localize('Etc/GMT+4'))
    myData=df.loc['['/']mask[']'/']
    xaxis=myData['['/']'time'[']'/']
    [for (itUserGroup : UserGroup | self.usergroup)]
        [for (itSensor : Sensor | self.basicsensor)]
    y[i/]axis=myData['['/']_[itSensor.id.replaceAll('-', '_')/][']'/']
        [/for]
    [/for]
    return dict(
    data=['['/']
    [for (itUserGroup : UserGroup | self.usergroup)]
        [for (itSensor : Sensor | self.basicsensor)]
        dict(
            x=xaxis,
            y=y[i/]axis,
            name=str(_[itSensor.id.replaceAll('-', '_')/]),
            type='[if (aGraph.eClass().name.equalsIgnoreCase('LineChart'))]line[/if]',
        ),
        [/for]
    [/for]
    [']'/'],
    layout=dict(
        title='[aGraph.name/]',
        showlegend=True,
        legend=dict(
```

```
50            x=0,
51            y=1.0
52        ),
53        margin=dict(l=40, r=0, t=40, b=30),
54        transition = {'duration': 500},
55    )
56 )
57 [/template]
```

Módulo generateAreaChartCallback

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateLineChartCallback.
4   */]
5  [module generateAreaChartCallback('https://gitlab.com/braulioqh/vis4bridge.git')
       ]
6
7
8  [**
9   * The documentation of the template generateAreaChartCallback.
10  * @param aGraph
11  */]
12 [template public generateAreaChartCallback(aGraph : Graph)]
13 @app.callback(
14    Output(component_id='[aGraph.id/]', component_property='figure'),
15    ['['/']
16    [for (itUserGroup : UserGroup | self.usergroup)]
17       [for (itSensor : Sensor | self.basicsensor)]
18     Input(component_id='dropdown-[itSensor.id/]', component_property='value'),
19       [/for]
20    [/for]
21     Input(component_id='rangeSlider-[aGraph.id/]', component_property='value')[
       ']'/]
22 )
23 def update_grahp([for (itUserGroup : UserGroup | self.usergroup)] [for (itSensor
       : Sensor | self.basicsensor)]_[itSensor.id.replaceAll('-', '_')/],[/for][/for]_[
       aGraph.id.replaceAll('-', '_')/]):
24
25    mask=(df['['/']'time'[']'/']>=unixToDatetime(_[aGraph.id.replaceAll('-', '_')/][']
       '/]).tz_localize('Etc/GMT+4')) & (df['['/']'time'[']'/']<=unixToDatetime(_[aGraph.id.
```

```
    replaceAll('-', '_')/][']1[']/]).tz_localize('Etc/GMT+4'))
26  myData=df.loc[']mask[']
27  xaxis=myData[']'time'[']
28  [for (itUserGroup : UserGroup | self.usergroup)]
29      [for (itSensor : Sensor | self.basicsensor)]
30  y[i/]axis=myData[']_[itSensor.id.replaceAll('-', '_')/][']
31      [/for]
32  [/for]
33  return dict(
34  data=[']
35  [for (itUserGroup : UserGroup | self.usergroup)]
36      [for (itSensor : Sensor | self.basicsensor)]
37      go.Scatter(
38          x=xaxis,
39          y=y[i/]axis,
40          name=str(_[itSensor.id.replaceAll('-', '_')/]),
41          stackgroup='one',
42      ),
43      [/for]
44  [/for]
45  ']/],
46  layout=dict(
47      title='[aGraph.name/]',
48      showlegend=True,
49      legend=dict(
50          x=0,
51          y=1.0
52      ),
53      margin=dict(l=40, r=0, t=40, b=30),
54      transition = {'duration': 500},
55  )
56  )
57  [/template]
```

Módulo generateHistogramCallback

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateLineChartCallback.
4   */]
5  [module generateHistogramCallback('https://gitlab.com/braulioqh/vis4bridge.git')
```

```
    ]
6
7
8  [**
9   * The documentation of the template generateLineChartCallback.
10  * @param aGraph
11  */]
12 [template public generateHistogramCallback(aGraph : Graph)]
13 @app.callback(
14    Output(component_id='[aGraph.id/]', component_property='figure'),
15    ['['/']
16    [for (itUserGroup : UserGroup | self.usergroup)]
17       [for (itSensor : Sensor | self.basicsensor)]
18     Input(component_id='dropdown-[itSensor.id/]', component_property='value'),
19       [/for]
20    [/for]
21    [for (itSensor : Sensor | self.sensor)]
22     Input(component_id='dropdown-[itSensor.id/]', component_property='value'),
23    [/for]
24     Input(component_id='rangeSlider-[aGraph.id/]', component_property='value'),
25     Input(component_id='bin-slider-[aGraph.id/]', component_property='value'),
26     Input(component_id='bin-auto-[aGraph.id/]', component_property='value'),['']
        /]
27 )
28 def update_graph([for (itUserGroup : UserGroup | self.usergroup)][for (itSensor :
        Sensor | self.basicsensor)]_[itSensor.id.replaceAll('-', '_')/],[/for][/for]_[
       aGraph.id.replaceAll('-', '_')/],bin_value,auto_state):
29
30    mask=(df['['/']'time'']'/']>=unixToDatetime(_[aGraph.id.replaceAll('-', '_')/]['['/']0[']
       '/']).tz_localize('Etc/GMT+4')) & (df['['/']'time'']'/']<=unixToDatetime(_[aGraph.id.
       replaceAll('-', '_')/]['['/']1[']'/']).tz_localize('Etc/GMT+4'))
31    myData=df.loc['['/']mask[']'/']
32
33    xaxis=myData['['/']'time'']'/']
34    [for (itUserGroup : UserGroup | self.usergroup)]
35       [for (itSensor : Sensor | self.basicsensor)]
36    y[i/]axis=myData['['/']_[itSensor.id.replaceAll('-', '_')/][']'/']
37    if "Auto" in auto_state:
38       hist[i/],bins[i/]=np.histogram(myData['['/']_[itSensor.id.replaceAll('-', '_')/][']'/'],
       range(int(round(min(myData['['/']_[itSensor.id.replaceAll('-', '_')/][']'/']))), int(
       round(max(myData['['/']_[itSensor.id.replaceAll('-', '_')/][']'/'])))))
39    else:
```

```
40      hist[i/],bins[i/]=np.histogram(myData[’[’/]_[itSensor.id.replaceAll(’−’, ’_’)/][’]’/],
        bins=bin_value)
41
42      [/for]
43    [/for]
44
45    return dict(
46      data=[’[’/]
47      [for (itUserGroup : UserGroup | self.usergroup)]
48        [for (itSensor : Sensor | self.basicsensor)]
49        dict(
50          x=bins[i/],
51          y=hist[i/],
52          name=str(_[itSensor.id.replaceAll(’−’, ’_’)/]),
53          type=’[if␣(aGraph.eClass().name.equalsIgnoreCase(’Histogram’))]bar[/
        if]’,
54        ),
55        [/for]
56      [/for]
57      [for (itSensor : Sensor | self.sensor)]
58        dict(
59          x=bins[i/],
60          y=hist[i/],
61          name=str(_[itSensor.id.replaceAll(’−’, ’_’)/]),
62          type=’[if␣(aGraph.eClass().name.equalsIgnoreCase(’Histogram’))]bar[/
        if]’,
63        ),
64      [/for]
65      [’]’/],
66      layout=dict(
67        title=’[aGraph.name/]’,
68        barmode=’stack’,
69        showlegend=True,
70        legend=dict(
71          x=0,
72          y=1.0
73        ),
74        margin=dict(l=40, r=0, t=40, b=30),
75        transition = {’duration’: 500},
76      )
77    )
78  [/template]
```

C.2. Transformación de Modelo a Javascript

Aquí se encuentra el código fuente Acceleo que genera la transformación de Modelo a Texto.

C.2.1. Paquete vis4bridge.acceleo.js.main

Módulo generate

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generate.
4   */]
5  [module generate('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7  [import braulioqh::vis4bridge::acceleo::js::files::generateHtml /]
8  [import braulioqh::vis4bridge::acceleo::js::files::generateCss /]
9  [import braulioqh::vis4bridge::acceleo::js::files::generateJavaScript /]
10 [import braulioqh::vis4bridge::acceleo::js::files::generateLibJs/]
11 [**
12  * The documentation of the template generateElement.
13  * @param aVis4bridge
14  */]
15 [template public generate(aVis4bridge : Vis4bridge)]
16 [comment @main/]
17 [generateHtml(aVis4bridge)/]
18 [generateCSS(aVis4bridge)/]
19 [generateJavaScript(aVis4bridge)/]
20 [generateLibJs(aVis4bridge)/]
21 [/template]
```

C.2.2. Paquete vis4bridge.acceleo.js.files

Módulo generateHtml

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateHtml.
```

```
4   */]
5   [module generateHtml('https://gitlab.com/braulioqh/vis4bridge.git')]
6   [import braulioqh::vis4bridge::acceleo::js::files::html::generateView/]
7
8   [**
9    * The documentation of the template generateHtml.
10   * @param aVis4bridge
11   */]
12  [template public generateHtml(aVis4bridge : Vis4bridge)]
13
14  [file ('index.html', false, 'UTF-8')]
15  <!DOCTYPE html>
16  <html lang="en" dir="ltr">
17    <head>
18      <meta charset="utf-8">
19      <link rel="stylesheet" href="css/master.css">
20          <link href="https://cdn.jsdelivr.net/npm/select2@4.1.0-beta.1/dist/css/
          select2.min.css" rel="stylesheet" />
21      <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/ion-
          rangeslider/2.3.1/css/ion.rangeSlider.min.css"/>
22      <title>Vis4bridge</title>
23    </head>
24    <body>
25      <header>
26        <h1>Vis4bridge</h1>
27      </header>
28      <div class="main">
29  [for (itView : View | self.view)]
30  [generateView(itView)/]
31  [/for]
32      </div>
33  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js" type=
        "text/javascript"></script>
34      <script src="https://cdn.jsdelivr.net/npm/select2@4.1.0-beta.1/dist/js/select2.
        min.js"></script>
35      <script src="https://cdnjs.cloudflare.com/ajax/libs/ion-rangeslider/2.3.1/js/ion.
        rangeSlider.min.js"></script>
36      <script src="https://cdnjs.cloudflare.com/ajax/libs/d3/5.7.0/d3.min.js" charset=
        "utf-8"></script>
37      <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.20.1/moment.
        min.js"></script>
38      <script src="https://cdnjs.cloudflare.com/ajax/libs/moment-timezone/0.5.31/
```

```
      moment-timezone.min.js" integrity="sha512-
      GqWOXT1UPIvzojfXEPf2ByPu4S0iwX0SfFfZ985fePNpTJPuiWKn47mXd0iyfcpcjcmM
      /HIRtvrd5TsR87A0Zg==" crossorigin="anonymous"></script>
39    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.min.js"
      charset="utf-8"></script>
40    <script src="js/lib.js" charset="utf-8"></script>
41    <script src="js/scripts.js" charset="utf-8"></script>
42  </body>
43 </html>
44 [/file]
45 [/template]
```

Módulo generateCss

```
 1 [comment encoding = UTF-8 /]
 2 [**
 3  * The documentation of the module generateCss.
 4  *]
 5 [module generateCss('https://gitlab.com/braulioqh/vis4bridge.git')]
 6
 7
 8 [**
 9  * The documentation of the template generateCSS.
10  * @param aVis4bridge
11  *]
12 [template public generateCSS(aVis4bridge : Vis4bridge)]
13
14 [file ('css/master.css', false, 'UTF-8')]
15 /*
16
17 CSS file for app.py
18 You can modify or add new CSS code in order to customize style for your app
19
20 */
21 body{
22   background: #F9F9F9;
23
24 }
25 h1, h2, h3{
26     text-align: center;
27 }
```

```
28 .dropdown-sensor{
29   width: 70px;
30 }
31 .dropdown-card{
32   width: 70px;
33 }
34 .sensor{
35   display: inline-block;
36   width: 110px;
37 }
38
39 .view-graph{
40   display: grid;
41   grid-template-columns: 1fr 1fr;
42
43 }
44 .graph{
45   margin: 2%;
46   padding: 2%;
47   height: auto;
48   background:#DEDEDE;
49 }
50 /* Float four columns side by side */
51 .column {
52   float: left;
53   width: 20%;
54   padding: 0 5px;
55 }
56
57 /* Remove extra left and right margins, due to padding */
58 .row {margin: 0 20px;}
59
60 /* Clear floats after the columns */
61 .row:after {
62   content: "";
63   display: table;
64   clear: both;
65 }
66
67 /* Responsive columns */
68 @media screen and (max-width: 600px) {
69   .column {
```

```
70      width: 100%;
71      display: block;
72      margin-bottom: 20px;
73    }
74  }
75
76  /* Style the counter cards */
77  .card {
78    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);
79    padding: 16px;
80    text-align: center;
81    background-color: #f1f1f1;
82  }
83  .content{
84    font-size: xx-large;
85  }
86  /* range slider styles */
87  .rangeSlider_container{
88    width: 95%;
89    margin:0 auto;
90    align-items: center;
91  }
92
93  .bins_container{
94    width: 95%;
95    margin:0 auto;
96    display: inline-block;
97    position: relative;
98    border-style: solid;
99    border-color: #B2B2B2;
100 }
101
102 .irs--flat .irs-handle>i:first-child {
103   background-color: #444444!important;
104 }
105 .irs--flat .irs-bar {
106   background-color: #5F5F5F!important;
107 }
108 .irs--flat .irs-from, .irs--flat .irs-to, .irs--flat .irs-single {
109   background-color: #444444!important;
110 }
111 .irs--flat .irs-from:before, .irs--flat .irs-to:before, .irs--flat .irs-single:
```

```
    before {
112   border-top-color: #5F5F5F!important;
113 }
114 [/file]
115 [/template]
```

Módulo generateJavascript

```
 1 [comment encoding = UTF-8 /]
 2 [**
 3  * The documentation of the module generateJavaScript.
 4  */]
 5 [module generateJavaScript('https://gitlab.com/braulioqh/vis4bridge.git')]
 6 [import braulioqh::vis4bridge::acceleo::js::files::javascript::
       generateGraphsInView /]
 7 [import braulioqh::vis4bridge::acceleo::js::files::javascript::generateCardsInView
       /]
 8
 9 [**
10  * The documentation of the template generateJavaScript.
11  * @param aVis4bridge
12  */]
13 [template public generateJavaScript(aVis4bridge : Vis4bridge)]
14
15 [file ('js/scripts.js', false, 'UTF-8')]
16 // Data source
17 var timeOfSensors      = 'https://gist.githubusercontent.com/bquiero/
       b9e63996e7159ad5e96eb607de092945/raw/
       bddbaa612cad3f4233d089763cce62f5bae8deb9/time.csv';
18 var accelerometersData = 'https://gist.githubusercontent.com/bquiero/
       b9e63996e7159ad5e96eb607de092945/raw/3
       a4483acb104b241af3538d41c7fe66af585acec/accelerometers.csv';
19 var straingaugesData   = 'https://gist.githubusercontent.com/bquiero/
       b9e63996e7159ad5e96eb607de092945/raw/3
       a4483acb104b241af3538d41c7fe66af585acec/straingauges.csv';
20 //get time zone
21 var td = new Date().getTimezoneOffset();
22 td=td+60;
23 //widget
24 //For selector2
25 $(document).ready(function() {
```

```
26      $('.selectpicker').select2();
27  });
28
29  //Main Function
30  main();
31  async function main(){
32      //get data
33      var data = await getData();
34      var time = data.time;
35      var accelerometer= data.accelerometer;
36      var straingauge= data.straingauge;
37
38      //set selectors
39      setDropdown('.dropdown-accelerometer',accelerometer.length-1,'ac');
40      setDropdown('.dropdown-straingauge',straingauge.length-1,'sg');
41      setDropdown('.dropdown-card',accelerometer.length-1,'ac');
42      setDropdown('.dropdown-card',straingauge.length-1,'sg');
43  [for (itView : View | self.view)]
44      [generateCardsInView(itView)/]
45      [generateGraphsInView(itView)/]
46  [/for]
47  }
48
49
50  [getData(aVis4bridge)/]
51
52  [/file]
53  [/template]
54  [template private getData(aVis4bridge : Vis4bridge)]
55
56  async function getData(){
57      // Note: The data source is globally defined.
58
59      // variables to save data
60      var myTime = new Array();
61      var accelerometer= new Array();
62      var straingauge= new Array();
63
64      //Asynchronous calls to get the data
65
66      //Get time
67      const response1 = await fetch(timeOfSensors);
```

```
68    const data_time = await response1.text();
69    const table1 = data_time.split('\n').slice(1);
70    table1.forEach(row => {
71        const columns = row.split(',');
72        const id = columns[ '[' /]0[ ']' /];
73        myTime.push(columns[ '[' /]1[ ']' /]);
74    })
75    //Get the accelerometers
76    const response2 = await fetch(accelerometersData);
77    const data_accelerometers = await response2.text();
78    const table2 = data_accelerometers.split('\n').slice(1);
79    var k = true;
80    table2.forEach(row => {
81        const columns = row.split(',');
82        if (k == true) {
83            for (let index = 0; index < columns.length; index++) {
84                accelerometer[ '[' /]index[ ']' /] = new Array();
85
86            }
87            k = false;
88        }
89        for (let index = 0; index < columns.length; index++) {
90            accelerometer[ '[' /]index[ ']' /].push(columns[ '[' /]index[ ']' /]);
91
92        }
93    })
94    //Obtiene los straingauge
95    const response3 = await fetch(straingaugesData);
96    const data_straingauges = await response3.text();
97    const table3 = data_straingauges.split('\n').slice(1);
98    var k = true;
99    table3.forEach(row => {
100       const columns = row.split(',');
101       if (k == true) {
102           for (let index = 0; index < columns.length; index++) {
103               straingauge[ '[' /]index[ ']' /] = new Array();
104
105           }
106           k = false;
107       }
108       for (let index = 0; index < columns.length; index++) {
109           straingauge[ '[' /]index[ ']' /].push(columns[ '[' /]index[ ']' /]);
```

```
110
111         }
112     })
113     //Retorna un diccionario con todos los valores
114     return {time: myTime, accelerometer: accelerometer, straingauge:straingauge
          }
115 }
116 [/template]
```

Módulo generateLibJs

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateLibJs.
4   */]
5  [module generateLibJs('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7
8  [**
9   * The documentation of the template generateLibJs.
10  * @param aVis4bridge
11  */]
12 [template public generateLibJs(aVis4bridge : Vis4bridge)]
13
14 [file ('js/lib.js', false, 'UTF-8')]
15 //color for charts
16 var color = new Array();
17 color.push('#1A5FD6');
18 color.push('#D67E1A');
19 color.push('#2CA02C');
20 color.push('#D62728');
21 color.push('#9467BD');
22 color.push('#8C564B');
23 //defauld bins
24 var defaultBins = 11;
25 /**
26  * Start a line graph
27  * @param {String} canvas - id of canvas element for graph
28  * @param {Array} sensors - array of sensors, each sensor if an array with
       data. example: sensors = [ '[' /] accelerometer[ '[' /] 2[ ']' /] ,straingauge[ '[' /]
       4[ ']' /][ ']' /]
```

```
29  * @param {Array} time - array of time data.
30  * @param {Number} td - diff of Local Date and UTC in minutes
31  * @param {Array} labels - labels for each sensor (optional) example: [ '[' /]'ac
        .2','sg.4'[ ']' /]
32  */
33 function initLineGraph(canvas,sensors,time,td,labels) {
34
35     var sensorDataSet = new Array();
36     //init each dataset in the graph
37     for (let index = 0; index < sensors.length; index++) {
38         var sensor = sensors[ '[' /] index[ ']' /] ;
39         var sensorData = new Array();
40         for (let index = 0; index < time.length; index++)
41             {
42                 var aData = {x:moment.unix(parseInt(time[ '[' /]index[ ']' /] )/1000).utc().
        add(td,'m'), y:parseFloat(sensor[ '[' /] index[ ']' /] )};
43                 sensorData.push(aData);
44             }
45         var lb = 's '+(index+1);
46         if (labels!=null) {
47             lb = labels[ '[' /]index[ ']' /];
48         }
49         sensorDataSet[ '[' /]index[ ']' /] =
50         {
51             label: lb,
52             borderColor: color[ '[' /]index[ ']' /],
53             data: sensorData,
54         };
55     }
56     //get canvas of graph
57     var ctx = document.getElementById(canvas).getContext('2d');
58     //generate the graph with the dataset
59     graph = new Chart(ctx, {
60         type: 'line',
61         data: {
62             datasets: sensorDataSet,
63         },
64         options: {
65             scales: {
66                 xAxes: ['['/]{
67                     type: 'time',
68                 }[ ']' /]
```

```
69            }
70        }
71    });
72    return graph;
73 }
74 /**
75  * Start a area graph
76  * @param {String} canvas - id of canvas element for graph
77  * @param {Array} sensors - array of sensors, each sensor if an array with
      data. example: sensors = ['']accelerometer['']2[ '' /] ,straingauge['']4[ '' /]
      [ '' /]
78  * @param {Array} time - array of time data.
79  * @param {Number} td - diff of Local Date and UTC in minutes
80  * @param {Array} labels - labels for each sensor (optional) example: [ '' /] 'ac
      .2','sg.4'[ '' /]
81  */
82 function initAreaGraph(canvas,sensors,time,td,labels) {
83
84    var sensorDataSet = new Array();
85    //init each dataset in the graph
86    for (let index = 0; index < sensors.length; index++) {
87        var sensor = sensors[ '' /]index[ '' /];
88        var sensorData = new Array();
89        for (let index = 0; index < time.length; index++)
90            {
91                var aData = {x:moment.unix(parseInt(time[ '' /]index[ '' /])/1000).utc().
      add(td,'m'), y:parseFloat(sensor[ '' /]index[ '' /])};
92                sensorData.push(aData);
93            }
94        var lb = 's '+(index+1);
95        if (labels!=null) {
96            lb = labels[ '' /]index[ '' /];
97        }
98        sensorDataSet[ '' /]index[ '' /] =
99        {
100           label: lb,
101           borderColor: color[ '' /]index[ '' /],
102           backgroundColor:color[ '' /]index[ '' /],
103           data: sensorData,
104       };
105   }
106   //get canvas of graph
```

```
107     var ctx = document.getElementById(canvas).getContext('2d');
108     //generate the graph with the dataset
109     graph = new Chart(ctx, {
110         type: 'line',
111         data: {
112             datasets: sensorDataSet,
113         },
114         options: {
115             scales: {
116                 xAxes: ['/']{
117                     type: 'time',
118                 }[']']
119             }
120         }
121     });
122     return graph;
123 }
124 /**
125  * Start a Histogram
126  * @param {String} canvas - id of canvas element for graph
127  * @param {Array} sensors - array of sensors, each sensor if an array with
        data. example: sensors = ['/']accelerometer['/']2[']'/],straingauge['/']4[']
        '/][']'/]
128  * @param {Array} time - array of time data.
129  * @param {Number} td - diff of Local Date and UTC in minutes
130  * @param {Array} labels - labels for each sensor (optional) example: ['/']'ac
        .2','sg.4'[']'/]
131  */
132 function initHistogram(canvas,sensors,time,td,labels) {
133
134     var sensorDataSet = new Array();
135     //init each dataset in the graph
136     for (let index = 0; index < sensors.length; index++) {
137         var sensor = sensors['/']index[']'/];
138         var sensorData = new Array();
139         var absSensorData = new Array();
140         for (let index = 0; index < time.length; index++)
141             {
142                 var aData = parseFloat(sensor['/']index[']'/]);
143                 sensorData.push(aData);
144                 absSensorData.push(Math.abs(aData));
145             }
```

```
146         var min = Math.min(...absSensorData);
147         var max = Math.max(...absSensorData);
148         //Generate bins with d3.js
149         var histGenerator = d3.histogram()
150                 .domain([ '[' /]min,max[ ']' /])
151                 .thresholds(defaultBins)
152         var histo = histGenerator(absSensorData);
153         var c = new Array();
154         var histogramData = new Array();
155         //set bins and labels for each bin
156         for (let i = 0; i < histo.length; i++) {
157             histogramData.push(histo[ '[' /]i[ ']' /].length);
158             c.push('[ ['  ']'/]'+Math.min(...histo[ '[' /]i[ ']' /])+'  '+Math.max(...histo[ '[' /]i[ ']
        ' /])+'[ ] ']'/]');
159         }
160
161         var lb = 's '+(index+1);
162         if (labels!=null) {
163             lb = labels[ '[' /]index[ ']' /];
164         }
165         sensorDataSet[ '[' /]index[ ']' /] =
166         {
167             label: lb,
168             backgroundColor:color[ '[' /]index[ ']' /],
169             data: histogramData,
170         };
171     }
172     //get canvas of graph
173     var ctx = document.getElementById(canvas).getContext('2d');
174     //generate the graph with the dataset
175     graph = new Chart(ctx, {
176         type: 'bar',
177         data: {
178             labels: c,
179             datasets: sensorDataSet,
180         },
181         options: {
182             scales: {
183                 y:{
184                     beginAtZero: true,
185                 }
186             }
```

```
187         }
188     });
189     return graph;
190 }
191
192 /**
193  * Trigger function that update a graph
194  * @param {*} graph - graph to update
195  * @param {String} container - id of container of chart
196  * @param {Object} dataset - dataset (time, sensors)
197  * @param {Number} td - diff of Local Date and UTC in minutes
198  * @param {String} widget - type of widget that trigger update
199  * @param {String} triggerElement - id of widget that trigger update
200  * @param {String} type - class of type of chart
201  */
202 function updateGraph(graph,container,dataset,td,widget,triggerElement,type) {
203     switch (type) {
204         case 'LineChart':
205             updateChart(graph,container,dataset,td,widget,triggerElement);
206             break;
207         case 'AreaChart':
208             updateChart(graph,container,dataset,td,widget,triggerElement);
209             break;
210         case 'Histogram':
211             updateHistogram(graph,container,dataset,td,widget,triggerElement);
212             break;
213         default:
214             break;
215     }
216 }
217 /**
218  * update a Line Chart or Area Chart
219  * @param {*} graph - graph to update
220  * @param {String} container - id of container of chart
221  * @param {Object} dataset - dataset (time, sensors)
222  * @param {Number} td - diff of Local Date and UTC in minutes
223  * @param {String} widget - type of widget that trigger update
224  * @param {String} triggerElement - id of widget that trigger update
225  */
226 function updateChart(graph,container,dataset,td,widget,triggerElement) {
227
228     //get dropdwon
```

```
229   var dp = document.getElementById(container).getElementsByClassName('
        dropdown-sensor');
230   //get range slider
231   var rs = document.getElementById(container).getElementsByClassName('
        data-slider')[ '[' /]0[ ']' /];
232   //get range
233   const range = rs.value.split(';');
234   var from = range[ '[' /]0[ ']' /];
235   var to = range[ '[' /]1[ ']' /];
236
237   // Each widget has different action
238   switch (widget) {
239      //In case of dropdown update data for sensor selected
240      case 'dropdown-sensor':
241         var dropdown = document.getElementById(triggerElement);
242         var sensor = dropdown.value;
243         var aSensor;
244         if (dropdown.classList.contains('dropdown-accelerometer')) {
245            aSensor = dataset.accelerometer;
246         }else if (dropdown.classList.contains('dropdown-straingauge')) {
247            aSensor = dataset.straingauge;
248         }
249
250         var sensorData = new Array();
251         for (let index = from; index < to; index++) {
252            var sensordata = {x:moment.unix(parseInt(dataset.time[ '[' /]index[ ']'
          /])/1000).utc().add(td,'m'), y:parseFloat(aSensor[ '[' /]sensor[ ']' /][ '[' /]index[ ']'
           /])};
253            sensorData.push(sensordata);
254         }
255         var k = 0;
256         while (dp[ '[' /]k[ ']' /]!=dropdown) {
257            k++;
258         }
259         graph.data.datasets[ '[' /]k[ ']' /].data = sensorData;
260         graph.update();
261         break;
262      //In case of range slider update range of each sensor
263      case 'data-slider':
264
265         var datasets = new Array();
266         for (let index = 0; index < dp.length; index++) {
```

```
267             var sensor = dp['['/]index[']'/].value;
268             var aSensor;
269             if (dp['['/]index[']'/].classList.contains('dropdown-accelerometer')) {
270                 aSensor = dataset.accelerometer;
271             }else if (dp['['/]index[']'/].classList.contains('dropdown-straingauge')
    ) {
272                 aSensor = dataset.straingauge;
273             }
274             var sensorData = new Array();
275             for (let j = from; j < to; j++) {
276                 var sensordata = {x:moment.unix(parseInt(dataset.time['['/]j[']'/])
    /1000).utc().add(td,'m'), y:parseFloat(aSensor['['/]sensor[']'/]['['/]j[']'/])};
277                 sensorData.push(sensordata);
278             }
279             graph.data.datasets['['/]index[']'/].data = sensorData;
280
281         }
282         graph.update();
283         break;
284       default:
285         break;
286     }
287
288
289 }
290 /**
291  * update a Histogram
292  * @param {*} graph - graph to update
293  * @param {String} container - id of container of chart
294  * @param {Object} dataset - dataset (time, sensors)
295  * @param {Number} td - diff of Local Date and UTC in minutes
296  * @param {String} widget - type of widget that trigger update
297  * @param {String} triggerElement - id of widget that trigger update
298  */
299 function updateHistogram(graph,container,dataset,td,widget,triggerElement) {
300     var bins;
301     //get dropdwon
302     var dp = document.getElementById(container).getElementsByClassName('
      dropdown-sensor');
303     //get range slider
304     var rs = document.getElementById(container).getElementsByClassName('
      data-slider')['['/]0[']'/];
```

```
305     //get range
306     const range = rs.value.split(';');
307     var from = range[ '[' /]0[ ']' /];
308     var to = range[ '[' /]1[ ']' /];
309     //get bins
310     var bs = document.getElementById(container).getElementsByClassName('
          bins-slider')[ '[' /]0[ ']' /];
311     bins = bs.value;
312
313     // Each widget has different action
314     switch (widget) {
315         //In case of dropdown update data for sensor selected
316         case 'dropdown-sensor':
317             var dropdown = document.getElementById(triggerElement);
318             //var sensor =  dropdown.value;
319             var sensor;
320             if (dropdown.classList.contains('dropdown-accelerometer')) {
321                 sensor = dataset.accelerometer[ '[' /] dropdown.value[ ']' /] ;
322             }else if (dropdown.classList.contains('dropdown-straingauge')) {
323                 sensor = dataset.straingauge[ '[' /] dropdown.value[ ']' /] ;
324             }
325             var sensorData = new Array();
326             var absSensorData = new Array();
327             for (let j = from; j < to; j++) {
328                 var aData = parseFloat(sensor[ '[' /] j[ ']' /] );
329                 sensorData.push(aData);
330                 absSensorData.push(Math.abs(aData));
331             }
332             var min = Math.min(...absSensorData);
333             var max = Math.max(...absSensorData);
334             //generate histogram with charjs
335             var histGenerator = d3.histogram()
336             .domain([ '[' /] min,max[ ']' /] )
337             .thresholds(bins)
338             var histo = histGenerator(absSensorData);
339             var c = new Array();
340             var histogramData = new Array();
341             for (let i = 0; i < histo.length; i++) {
342                 histogramData.push(histo[ '[' /]i[']'/].length);
343                 c.push('[␣['␣/]'+Math.min(...histo[ '[' /]i[']'/])+'-'+Math.max(...histo[ '[' /]i['
          ]'/])+'[␣]'␣/]');
344             }
```

```
345
346         var k = 0;
347         while (dp[ '[' /]k[']'/]!=dropdown) {
348             k++;
349         }
350
351         graph.data.datasets[ '[' /]k[']'/].data = histogramData;
352         graph.update();
353         break;
354      //In case of range slider update range of each sensor
355      case 'data-slider':
356
357         var datasets = new Array();
358         for (let index = 0; index < dp.length; index++) {
359             //var sensor = dp[ '[' /]index[']'/].value;
360             var sensor;
361             if (dp[ '[' /]index[']'/].classList.contains('dropdown-accelerometer')) {
362                 sensor = dataset.accelerometer[ '[' /]dp[ '[' /]index[']'/].value[']'/];
363             }else if (dp[ '[' /]index[']'/].classList.contains('dropdown-straingauge'))
    {
364                 sensor = dataset.straingauge[ '[' /]dp[ '[' /]index[']'/].value[']'/];
365             }
366             var sensorData = new Array();
367             var absSensorData = new Array();
368             for (let j = from; j < to; j++) {
369                 var aData = parseFloat(sensor['['/]j[']'/]);
370                 sensorData.push(aData);
371                 absSensorData.push(Math.abs(aData));
372             }
373             var min = Math.min(...absSensorData);
374             var max = Math.max(...absSensorData);
375
376             //Se genera el histograma
377             var histGenerator = d3.histogram()
378             .domain([ '[' /] min,max[ ']' /] )
379             .thresholds(bins)
380             var histo = histGenerator(absSensorData);
381             var c = new Array();
382             var histogramData = new Array();
383             for (let i = 0; i < histo.length; i++) {
384                 histogramData.push(histo['['/]i[']'/].length);
385                 c.push('[␣['␣/]'+Math.min(...histo[ '[' /]i[']'/])+'-'+Math.max(...histo[ '['
```

```
    /]i[']']])+'[ ]'<br>/]');
386             }
387             graph.data.datasets[ '[' /]index[']'/].data = histogramData;
388
389         }
390         graph.data.labels = c;
391         graph.update();
392         break;
393       case 'bins-slider':
394         var datasets = new Array();
395         for (let index = 0; index < dp.length; index++) {
396             //var sensor = dp[ '[' /]index[']'/].value;
397             var sensor;
398             if (dp[ '[' /]index[']'/].classList.contains('dropdown-accelerometer')) {
399                 sensor = dataset.accelerometer[ '[' /]dp[ '[' /]index[']'/].value[']'/];
400             }else if (dp[ '[' /]index[']'/].classList.contains('dropdown-straingauge'))
    {
401                 sensor = dataset.straingauge[ '[' /]dp[ '[' /]index[']'/].value[']'/];
402             }
403             var sensorData = new Array();
404             var absSensorData = new Array();
405             for (let j = from; j < to; j++) {
406                 var aData = parseFloat(sensor[ '[' /]j[']'/]);
407                 sensorData.push(aData);
408                 absSensorData.push(Math.abs(aData));
409             }
410             var min = Math.min(...absSensorData);
411             var max = Math.max(...absSensorData);
412
413             //Se genera el histograma
414             var histGenerator = d3.histogram()
415             .domain([ '[' /]min,max[ ']' /] )
416             .thresholds(bins)
417             var histo = histGenerator(absSensorData);
418             var c = new Array();
419             var histogramData = new Array();
420             for (let i = 0; i < histo.length; i++) {
421                 histogramData.push(histo[ '[' /] i[']'/].length);
422                 c.push('[ ['<br>/]i[']'/])+'[ ]'<br>/]');
423             }
424             graph.data.labels = c;
```

```
425                 graph.data.datasets[ '[' /]index[']'/].data = histogramData;
426
427             }
428             graph.update();
429             break;
430         default:
431             break;
432     }
433
434
435 }
436 function updateCard(card,container,dataset,td,type) {
437     //get dropdwon
438     var dp = document.getElementById(container).getElementsByClassName('
          dropdown-card')[ '[' /]0[ ']' /];
439     //get content
440     var content = document.getElementById(container).
          getElementsByClassName('content')[ '[' /]0[ ']' /];
441     var sensor;
442     //get sensor
443     if (dp.options[ '[' /] dp.selectedIndex[ ']' /] .text.includes('ac')) {
444         sensor = dataset.accelerometer[ '[' /] dp.value[ ']' /] ;
445     } else if (dp.options[ '[' /] dp.selectedIndex[ ']' /] .text.includes('sg')){
446         sensor = dataset.straingauge[ '[' /] dp.value[ ']' /] ;
447     }
448
449     absSensor= new Array();
450     for (let index = 0; index < sensor.length; index++) {
451         var aData = parseFloat(sensor[ '[' /] index[ ']' /] );
452         absSensor.push(Math.abs(aData))
453     }
454     switch (type) {
455         case 'MIN':
456             var min = Math.min(...absSensor);
457             content.innerHTML = String(min);
458             break;
459         case 'MAX':
460             var max = Math.max(...absSensor);
461             content.innerHTML = String(max);
462             break;
463         case 'AVERAGE':
464             var sum=0;
```

```
465             for (let i = 0; i < absSensor.length; i++) {
466                 sum = sum+absSensor[ '[' /]i[ ']' /] ;
467             }
468             var average = sum/absSensor.length;
469             content.innerHTML = average.toFixed(4);
470             break;
471         default:
472             break;
473     }
474 }
475 function onChangeCardDropdown(select,td,dataset,card,type) {
476     //get container of card
477     var container = document.getElementById(select).parentElement.
          parentElement.parentElement.id;
478     // update card
479     $('#'+select).on('change', function (e) {
480         updateCard(card,container,dataset,td,type)
481       });
482 }
483 /**
484  * update graph on change dropdown (select)
485  * @param {String} select - id of dropdown
486  * @param {Number} td - diff of Local Date and UTC
487  * @param {Object} dataset - dataset (time, sensors)
488  * @param {*} graph - graph to update
489  */
490 function onChangeDropdown(select,td,dataset,graph,type) {
491     //get container of graph
492     var container = document.getElementById(select).parentElement.
          parentElement;
493     // update graph
494     $('#'+select).on('change', function (e) {
495         updateGraph(graph,container.id,dataset,td,'dropdown-sensor',select,type);
496       });
497 }
498 /**
499  * set a dropdown o group of dropdown
500  * @param {String} dropdown - id or class of dropdown
501  * @param {Number} max - number of sensors
502  * @param {*} label - short label
503  */
504 function setDropdown(dropdown,max,label){
```

```
505     //default label
506     var lb = "S. ";
507     if (label!=null) {
508         lb = label+" ";
509     }
510     //get all dropdown with id or class 'dropdown'
511     const dropdown_sensor = document.querySelectorAll(dropdown);
512     //put options (sensors) in each dropdown
513     dropdown_sensor.forEach(element => {
514         for (var i = 1; i < max; i++) {
515             var opt = document.createElement('option');
516             opt.value = i;
517             opt.innerHTML = String(lb+i);
518             element.appendChild(opt);
519         }
520     })
521 }
522 /**
523  * set a range slider and trigger events of slider
524  * @param {String} slider - id of range slider
525  * @param {*} time - data with time
526  * @param {*} td - diff of Local Date and UTC
527  */
528 function setRangeSlider(slider,time,td,dataset,graph,type) {
529     //function for show time in dropdown
530     function ptf(data) {
531         return moment.unix(parseInt(time[ '[' /]data[ ']' /])/1000).utc().format("HH:
          mm:ss a");
532     }
533     //new range slider
534     var $myrange = $('#'+slider).ionRangeSlider({
535         type: "double",
536         grid: true,
537         min: 0,
538         max: time.length-1,
539         from: 0,
540         to: time.length-1,
541         prefix: "",
542         prettify: ptf,
543         onStart: function (data) {
544             // fired then range slider is ready
545         },
```

```
546         onChange: function (data) {
547           //  fired then range slider change
548         },
549         onFinish: function (data) {
550             // on finish update a graph
551             var container = document.getElementById(slider).parentElement.
        parentElement;
552             updateGraph(graph,container.id,dataset,td,'data-slider','data-slider',
        type);
553
554         },
555         onUpdate: function (data) {
556             // fired on changing slider with Update method
557         }
558     });
559 }
560
561 /**
562  * set a range slider and trigger events of slider
563  * @param {String} slider - id of range slider
564  * @param {*} time - data with time
565  * @param {*} td - diff of Local Date and UTC
566  */
567 function setBinsSlider(slider,time,td,dataset,graph,type) {
568
569     //function for show time in dropdown
570     function ptf(data) {
571         return data;
572     }
573     //new range slider
574     var $myrange = $('#'+slider).ionRangeSlider({
575         min: 1,
576         max: 40,
577         from: defaultBins,
578         grid: true,
579         prefix: 'threshold: ',
580         block: true,
581         step: 10,
582         onStart: function (data) {
583             // fired then range slider is ready
584         },
585         onChange: function (data) {
```

```
586             //  fired then range slider change
587           },
588           onFinish: function (data) {
589               // on finish update a graph
590               var container = document.getElementById(slider).parentElement.
          parentElement;
591               updateHistogram(graph,container.id,dataset,td,'bins-slider','bins-slider',
          data.from);
592           },
593           onUpdate: function (data) {
594               // fired on changing slider with Update method
595           }
596       });
597   }
598   /**
599    * update graph on change checkbox
600    * @param {String} checkbox - id of dropdown
601    * @param {Number} td - diff of Local Date and UTC
602    * @param {Object} dataset - dataset (time, sensors)
603    * @param {*} graph - graph to update
604    */
605   function onChangeCheckbox(checkbox,td,dataset,graph) {
606
607       var checkbox = document.getElementById(checkbox);
608       var slider = checkbox.parentElement.getElementsByClassName('bins-slider')
          [ '[' /]0[ ']' /];
609       var $range = $('#'+slider.id);
610       var instance = $range.data("ionRangeSlider");
611       var container = checkbox.parentElement.parentElement;
612       checkbox.addEventListener( 'change', function() {
613           if(this.checked) {
614               instance.update({
615                   block: true,
616                   from:defaultBins,
617               });
618               updateHistogram(graph,container.id,dataset,td,'bins-slider','bins-slider',
          defaultBins);
619
620           } else {
621               instance.update({
622                   block: false
623               });
```

```
624         }
625     });
626 }
627 [/file]
628 [/template]
```

C.2.3. Paquete vis4bridge.acceleo.js.files.html

Módulo generateView

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateView.
4   */]
5  [module generateView('https://gitlab.com/braulioqh/vis4bridge.git')]
6  [import braulioqh::vis4bridge::acceleo::js::files::html::generateCardInView /]
7  [import braulioqh::vis4bridge::acceleo::js::files::html::generateGraphInView /]
8
9  [**
10  * The documentation of the template generateView.
11  * @param aView
12  */]
13 [template public generateView(aView : View)]
14         <div class="view" id="view1">
15           <h2>[aView.name/]</h2>
16           <div class="view-card row" id="[aView.id/]_cards">
17 [if (aView.card->notEmpty())]
18     [for (itCard : Card | card)]
19         [generateCardInView(itCard)/]
20     [/for]
21 [/if]
22           </div>
23           <div class="view-graph" id="[aView.id/]_graph">
24 [if (aView.graph->notEmpty())]
25     [for (itGraph : Graph | graph)]
26         [generateGraphInView(itGraph)/]
27     [/for]
28 [/if]
29           </div>
30         </div>
31 [/template]
```

Módulo generateGraphInView

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateGraphInView.
4   */]
5  [module generateGraphInView('https://gitlab.com/braulioqh/vis4bridge.git')]
6  [import braulioqh::vis4bridge::acceleo::js::files::html::graphs::generateLineChart
       /]
7  [import braulioqh::vis4bridge::acceleo::js::files::html::graphs::generateAreaChart
       /]
8  [import braulioqh::vis4bridge::acceleo::js::files::html::graphs::generateHistogram
       /]
9
10 [**
11  * The documentation of the template generateGraphInView.
12  * @param aGraph
13  */]
14 [template public generateGraphInView(aGraph : Graph)]
15 [let aGraphType : String = aGraph.eClass().name]
16           <div class="[aGraphType/]_graph" id="[aGraph.id/]_container">
17 [if (aGraph.eClass().name.equalsIgnoreCase('LineChart'))]
18    [generateLineChart(aGraph)/]
19 [elseif(aGraph.eClass().name.equalsIgnoreCase('AreaChart'))]
20    [generateAreaChart(aGraph)/]
21 [elseif(aGraph.eClass().name.equalsIgnoreCase('Histogram'))]
22    [generateHistogram(aGraph)/]
23 [/if]
24           </div>
25 [/let]
26 [/template]
```

Módulo generateCardInView

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateCardInView.
4   */]
5  [module generateCardInView('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7
```

```
8  [**
9   * The documentation of the template generateCardInView.
10  * @param aCard
11  */]
12 [template public generateCardInView(aCard : Card)]
13 [let aCardType : String = aCard.Type.toString()]
14         <div class="column " id="container_[aCard.id/]">
15           <div class="[aCard.eClass().name/]">
16             <div class="sensor" id="[aCard.id/]_sensor">
17               <select class="selectpicker dropdown-card" id="dropdown_[aCard.id
       /]">
18               </select>
19             </div>
20             <div class="card">
21               <h3>[aCardType.toLower().toUpperFirst()/] Card</h3>
22               <p id="content_[aCard.id/]" class="content">content</p>
23             </div>
24           </div>
25         </div>
26 [/let]
27 [/template]
```

C.2.4. Paquete vis4bridge.acceleo.js.files.html.graphs

Módulo generateSelectorsForSensors

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateSelectorsForSensors.
4   */]
5  [module generateSelectorsForSensors('https://gitlab.com/braulioqh/vis4bridge.
       git')]
6
7
8  [**
9   * The documentation of the template generateElement.
10  * @param anUserGroup
11  */]
12 [template public generateSelectorsForSensors(anUserGroup : UserGroup)]
13 [if (anUserGroup.basicsensor->notEmpty())]
```

```
14   <div class="select-group">
15 [for (itSensor : BasicSensor | self.basicsensor)]
16 [if (itSensor.oclIsTypeOf(TriAxisAccelerometer))]
17     <select class="dropdown-sensor selectpicker dropdown-accelerometer" id=
       "dropdown_[itSensor.id/]"></select>
18 [elseif (itSensor.oclIsTypeOf(StrainGauge))]
19     <select class="dropdown-sensor selectpicker dropdown-straingauge" id="
       dropdown_[itSensor.id/]"></select>
20 [/if]
21 [/for]
22   </div>
23 [/if]
24 [/template]
```

Módulo generateLineChart

```
 1 [comment encoding = UTF-8 /]
 2 [**
 3  * The documentation of the module generateLineChart.
 4  */]
 5 [module generateLineChart('https://gitlab.com/braulioqh/vis4bridge.git')]
 6 [import braulioqh::vis4bridge::acceleo::js::files::html::graphs::
       generateSelectorsForSensors /]
 7
 8 [**
 9  * The documentation of the template generateLineChart.
10  * @param aGraph
11  */]
12 [template public generateLineChart(aGraph : Graph)]
13 [if (aGraph.usergroup->notEmpty())]
14     [for (itUsergroup : UserGroup | self.usergroup)]
15         [generateSelectorsForSensors(itUsergroup)/]
16     [/for]
17 [/if]
18             <h3>[aGraph.name/]</h3>
19             <div class="graph" id="[aGraph.id/]">
20               <canvas id="[aGraph.id/]_canvas"></canvas>
21             </div>
22             <div class="rangeSlider_container">
23               <input type="text" class="js-range-slider_data-slider" id="slider_[
       aGraph.id/]"/>
```

```
24              </div>
25  [/template]
```

Módulo generateAreaChart

```
1   [comment encoding = UTF-8 /]
2   [**
3    * The documentation of the module generateAreaChart.
4    */]
5   [module generateAreaChart('https://gitlab.com/braulioqh/vis4bridge.git')]
6   [import braulioqh::vis4bridge::acceleo::js::files::html::graphs::
        generateSelectorsForSensors /]
7
8   [**
9    * The documentation of the template generateAreaChart.
10   * @param aGraph
11   */]
12  [template public generateAreaChart(aGraph : Graph)]
13  [if (aGraph.usergroup->notEmpty())]
14      [for (itUsergroup : UserGroup | self.usergroup)]
15          [generateSelectorsForSensors(itUsergroup)/]
16      [/for]
17  [/if]
18              <h3>[aGraph.name/]</h3>
19              <div class="graph" id="[aGraph.id/]">
20                <canvas id="[aGraph.id/]_canvas"></canvas>
21              </div>
22              <div class="rangeSlider_container">
23                <input type="text" class="js-range-slider data-slider" id="slider_[
        aGraph.id/]"/>
24              </div>
25  [/template]
```

Módulo generateHistogram

```
1   [comment encoding = UTF-8 /]
2   [**
3    * The documentation of the module generateHistogram.
4    */]
5   [module generateHistogram('https://gitlab.com/braulioqh/vis4bridge.git')]
```

```
6  [import braulioqh::vis4bridge::acceleo::js::files::html::graphs::
       generateSelectorsForSensors /]
7
8  [**
9   * The documentation of the template generateHistogram.
10  * @param aGraph
11  */]
12 [template public generateHistogram(aGraph : Graph)]
13 [if (aGraph.usergroup->notEmpty())]
14    [for (itUsergroup : UserGroup | self.usergroup)]
15       [generateSelectorsForSensors(itUsergroup)/]
16    [/for]
17 [/if]
18             <h3>[aGraph.name/]</h3>
19             <div class="graph" id="[aGraph.id/]">
20               <canvas id="[aGraph.id/]_canvas"></canvas>
21             </div>
22             <div class="rangeSlider_container">
23               <input type="text" class="js-range-slider data-slider" id="slider_[
       aGraph.id/]"/>
24             </div>
25             <div class="bins_container">
26               <label for="bins_[aGraph.id/]">Bins (Auto):</label>
27               <input id="bins_[aGraph.id/]" type="checkbox" checked=true>
28               <input type="text" class="js-range-slider bins-slider" id="binsSlider_
       [aGraph.id/]" value="" />
29             </div>
30 [/template]
```

C.2.5. Paquete vis4bridge.acceleo.js.files.javascript

Módulo generateGraphsInView

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateGraphsInView.
4   */]
5  [module generateGraphsInView('https://gitlab.com/braulioqh/vis4bridge.git')]
6  [import braulioqh::vis4bridge::acceleo::js::files::javascript::generateLineChartJs
       /]
```

```
7  [import braulioqh::vis4bridge::acceleo::js::files::javascript::generateAreaChartJs
       /]
8  [import braulioqh::vis4bridge::acceleo::js::files::javascript::generateHistogramJs
       /]
9
10 [**
11  * The documentation of the template generateGraphsInView.
12  * @param aView
13  */]
14 [template public generateGraphsInView(aView : View)]
15 [if (aView.graph->notEmpty())]
16    [for (itGraph : Graph | graph)]
17 [let aGraphType : String = itGraph.eClass().name]
18       [if (itGraph.eClass().name.equalsIgnoreCase('LineChart'))]
19          [generateLineChartJs(itGraph)/]
20       [elseif(itGraph.eClass().name.equalsIgnoreCase('AreaChart'))]
21          [generateAreaChartJs(itGraph)/]
22       [elseif(itGraph.eClass().name.equalsIgnoreCase('Histogram'))]
23          [generateHistogramJs(itGraph)/]
24       [/if]
25 [/let]
26    [/for]
27 [/if]
28 [/template]
```

Módulo generateCardsInView

```
1  [comment encoding = UTF-8 /]
2  [**
3   * The documentation of the module generateCardsInView.
4   */]
5  [module generateCardsInView('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7
8  [**
9   * The documentation of the template generateCardsInView.
10  * @param aView
11  */]
12 [template public generateCardsInView(aView : View)]
13 [if (aView.card->notEmpty())]
14       [for (itCard : Card | card)]
```

```
15 [let aCardType : String = itCard.Type.toString()]
16 updateCard('container_[itCard.id/]','container_[itCard.id/]',data,td,'[aCardType/]');
17 onChangeCardDropdown('dropdown_[itCard.id/]',td,data,'container_[itCard.id/]','[
       aCardType/]');
18 [/let]
19       [/for]
20 [/if]
21 [/template]
```

Módulo generateLineChartJs

```
 1 [comment encoding = UTF-8 /]
 2 [**
 3  * The documentation of the module generateLineChartJs.
 4  */]
 5 [module generateLineChartJs('https://gitlab.com/braulioqh/vis4bridge.git')]
 6
 7
 8 [**
 9  * The documentation of the template generateLineChartJs.
10  * @param aGraph
11  */]
12 [template public generateLineChartJs(aGraph : Graph)]
13 [let graph : String = aGraph.id.replaceAll('-', '_')]
14    var [graph/];
15    sensors = new Array();
16    [for (itUserGroup : UserGroup | self.usergroup)]
17       [for (itSensor : BasicSensor | self.basicsensor)]
18          [if (itSensor.oclIsTypeOf(TriAxisAccelerometer))]
19             sensors.push(accelerometer[ '[' /]1[ ']' /]);
20          [elseif (itSensor.oclIsTypeOf(StrainGauge))]
21             sensors.push(straingauge[ '[' /]1[ ']' /]);
22          [/if]
23       [/for]
24    [/for]
25    [graph/] = initLineGraph('[aGraph.id/]_canvas',sensors,time,td);
26    setRangeSlider('slider_[aGraph.id/]',time,td,data,[graph/],'AreaChart');
27    [for (itUserGroup : UserGroup | self.usergroup)]
28       [for (itSensor : BasicSensor | self.basicsensor)]
29    onChangeDropdown('dropdown_[itSensor.id/]',td,data,[graph/],'AreaChart');
30       [/for]
```

```
31     [/for]
32 [/let]
33 [/template]
```

Módulo generateAreaChartJs

```
 1 [comment encoding = UTF-8 /]
 2 [**
 3  * The documentation of the module generateAreaChartJs.
 4  */]
 5 [module generateAreaChartJs('https://gitlab.com/braulioqh/vis4bridge.git')]
 6
 7
 8 [**
 9  * The documentation of the template generateAreaChartJs.
10  * @param aGraph
11  */]
12 [template public generateAreaChartJs(aGraph : Graph)]
13 [let graph : String = aGraph.id.replaceAll('-', '_')]
14     var [graph/];
15     sensors = new Array();
16     [for (itUserGroup : UserGroup | self.usergroup)]
17         [for (itSensor : BasicSensor | self.basicsensor)]
18             [if (itSensor.oclIsTypeOf(TriAxisAccelerometer))]
19                 sensors.push(accelerometer[ '[' /]1[ ']' /]);
20             [elseif (itSensor.oclIsTypeOf(StrainGauge))]
21                 sensors.push(straingauge[ '[' /]1[ ']' /]);
22             [/if]
23         [/for]
24     [/for]
25     [graph/] = initAreaGraph('[aGraph.id/]_canvas',sensors,time,td);
26     setRangeSlider('slider_[aGraph.id/]',time,td,data,[graph/],'AreaChart');
27     [for (itUserGroup : UserGroup | self.usergroup)]
28         [for (itSensor : BasicSensor | self.basicsensor)]
29     onChangeDropdown('dropdown_[itSensor.id/]',td,data,[graph/],'AreaChart');
30         [/for]
31     [/for]
32 [/let]
33 [/template]
```

Módulo generateHistogramJs

```
1  [comment encoding = UTF−8 /]
2  [**
3   * The documentation of the module generateHistogramJs.
4   */]
5  [module generateHistogramJs('https://gitlab.com/braulioqh/vis4bridge.git')]
6
7
8  [**
9   * The documentation of the template generateHistogramJs.
10  * @param aGraph
11  */]
12 [template public generateHistogramJs(aGraph : Graph)]
13 [let graph : String = aGraph.id.replaceAll('−', '_')]
14    var [graph/];
15    sensors = new Array();
16    [for (itUserGroup : UserGroup | self.usergroup)]
17       [for (itSensor : BasicSensor | self.basicsensor)]
18          [if (itSensor.oclIsTypeOf(TriAxisAccelerometer))]
19             sensors.push(accelerometer[ '[' /]1[ ']' /]);
20          [elseif (itSensor.oclIsTypeOf(StrainGauge))]
21             sensors.push(straingauge[ '[' /]1[ ']' /]);
22          [/if]
23       [/for]
24    [/for]
25    [graph/] = initHistogram('[aGraph.id/]_canvas',sensors,time,td);
26    setRangeSlider('slider_[aGraph.id/]',time,td,data,[graph/],'Histogram');
27    setBinsSlider('binsSlider_[aGraph.id/]',time,td,data,[graph/],'Histogram')
28    onChangeCheckbox('bins_[aGraph.id/]',td,data,[graph/],'Histogram');
29    [for (itUserGroup : UserGroup | self.usergroup)]
30       [for (itSensor : BasicSensor | self.basicsensor)]
31    onChangeDropdown('dropdown_[itSensor.id/]',td,data,[graph/],'Histogram');
32       [/for]
33    [/for]
34 [/let]
35 [/template]
```

D. Cuestionario de Usabilidad

D.1. Cuestionario

Vis4Bridge Usability Questionnaire

*Required

For the following questions indicate the degree of agreement, where 7 is Totally agree and 1 Totally disagree.

7: Strongly Agree
6: Agree
5: Somewhat Agree
4: Neither Agree nor Disagree
3: Somewhat Disagree
2: Disagree
1: Strongly Disagree

Overall, I am satisfied with the ease of completing this task. *

	1	2	3	4	5	6	7
Place items on the canvas	○	○	○	○	○	○	○
Connect objects	○	○	○	○	○	○	○
Edit object properties	○	○	○	○	○	○	○
Generate the visualization code	○	○	○	○	○	○	○

Overall, I am satisfied with the amount of time it took to complete this task. *

	1	2	3	4	5	6	7
Place items on the canvas	○	○	○	○	○	○	○
Connect objects	○	○	○	○	○	○	○
Edit object properties	○	○	○	○	○	○	○
Generate the visualization code	○	○	○	○	○	○	○

Overall, I am satisfied with the support information (on-line help, messages,documentation) when completing this task. *

	1	2	3	4	5	6	7
Place items on the canvas	○	○	○	○	○	○	○
Connect objects	○	○	○	○	○	○	○
Edit object properties	○	○	○	○	○	○	○
Generate the visualization code	○	○	○	○	○	○	○

Overall, I am satisfied with how easy it is to use this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

It was simple to use this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

I could effectively complete the tasks and scenarios using this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

I was able to complete the tasks and scenarios quickly using this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

I was able to efficiently complete the tasks and scenarios using this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

I felt comfortable using this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

It was easy to learn to use this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

I believe I could become productive quickly using this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

The system gave error messages that clearly told me how to fix problems. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

Whenever I made a mistake using the system, I could recover easily and quickly. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

The information (such as on-line help, on-screen messages and other documentation)provided with this system was clear. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

It was easy to find the information I needed. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

The information provided for the system was easy to understand. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

The information was effective in helping me complete the tasks and scenarios. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

The organization of information on the system screens was clear. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

The interface of this system was pleasant. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

I liked using the interface of this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

This system has all the functions and capabilities I expect it to have. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

Overall, I am satisfied with this system. *

	1	2	3	4	5	6	7	
Strongly Disagree	○	○	○	○	○	○	○	Strongly Agree

If you want, you can add additional comments here.

Your answer

Enter your contact email here (optional).

Your answer

Submit

Never submit passwords through Google Forms.

D.2. Comentarios adicionales de los encuestados

A continuación, se muestran los comentarios adicionales de los encuestados:

'Al crear un grupo de sensores tengo que seleccionar de nuevo la accion para hacer otro grupo. no es nada grabe pero seria mas comodo que se mantenga seleccionado'.

'No puedo comprobar si era fácil o no recuperar sobre errores debido a que no tuve ninguno durante el transcurso de las pruebas, muy buenos tutoriales de texto se entiende perfectamente que es lo que hay que hacer en cada paso. Todo el sistema funcionó de manera muy rápida, sencilla y los resultados se pueden ver rápidamente'.

'Fue muy sencillo y super bien explicado el procedimiento para utilizar el sistema, me siento conforme con lo que realicé y muy de acuerdo en que el sistema funciona correctamente. Dado que no tuve ningún error no puedo responder con sinceridad si el sistema entrega mensajes de error que me indiquen claramente como solucionar los problemas'.

'Al principio me enredó el cómo conectar los elementos, y mientras exploraba la aplicación me di cuenta de que los gráficos son en base a los grupos, y las cards a todos los dispositivos. Hubiese sido deseable seleccionar que en una card sólo un tipo de sensor (acelerómetros, strain gauges, etc). pero en si fue cómodo de utilizar, sencillo de instalar y es perfecto para construir un MVP de visualización de datos'.

'Falta un ejemplo con datos reales de sensores que podriamos seleccionar desde un archivo cualquiera, por ejemplo de aceleración y jugar además de con los periodos de muestreo con la frecuencia que es visualizable. La opción o forma de guardar esta un poco extraña, normalmente hay una opción save y ahi damos un nombre y se genera una carpeta especifica para el proyecto en el work space... aqui (y nose si fue problema mio) todos los archivos de visualización se creaban sobre el anterior con el nombre index por defecto'.

'Las preguntas con error las deje con un 4 porque nunca me surgio esa ventana, no tuve problemas en el desarrollo de las tareas'.

'Gran trabajo, felicidades'.

'Luego de realizar el tutorial y enfrentarme a los escenarios, fue muy sencillo implementarlos. Encontré que fue un proceso muy ágil ya que no me tomo casi nada

de tiempo y los tutoriales son bastante completos aun siendo cortos'.

'Dentro de la sección de errores, marque 2 opciones con 4 dado que no se me presentó ningún error durante la realización de las pruebas. Además, me gustaría mencionar 2 cosas, la primera es un detalle en realidad, sobre la redacción en la parte de los enunciados, podría mejorarse un poco y la segunda corresponde al caso en el que deseo eliminar un connector dentro de la vista, al parecer no me permite hacerlo sin tener que necesariamente eliminar el gráfico al que lo conecte. Aparte de estos detalles, muy buena herramienta y fácil de usar'.

'Como observación, creo que hay un error en el escenario 2, punto 4 dice que conecte el grupo se sensores con el grafico de líneas, pero en este escenario no se utiliza gráfico de lineas. Además los grupos ya estan conectados como se indican en los puntos anteriores'.

'Podría modificar la documentación y mejorar la redacción tanto de tutoriales como de instalación'.

'Una herramienta sencilla pero funcional'.

'Como acotación, a veces cuando arrastraba algún elemento al lienzo, el Obeo se caía por un par de segundos, y esto sucedió sólo después de la instalación de la herramienta'.

'Algo que me parece que se debe mejorar es que los objetos al soltarlos suelen quedar desordenados en las vistas y grupos de sensores, haciendo que se vea enredado visualmente. Más allá de eso, el sistema me ha resultado fácil de usar'.