Un enfoque a la Generación Automática de Microservicios ColmenaTool

Alberto Cortez[1,2,3,4], Carlos Martinez[1,2,4], Claudia Naveda[1,2,3,4], Raúl Varela[1,2,4]

Joel Patrizio[3], Ana Garis[4], Alejandro Vazquez[1,3]

[1] Laboratorio de Auditoria y Seguridad de TI – UTN Facultad Regional Mendoza – Mendoza -Argentina

[2] Instituto de Informática de la Facultad de Ingeniería - Universidad de Mendoza - Mendoza- Argentina

[3]Licenciatura en Informática y Desarrollo de Software - Universidad del Aconcagua - Mendoza- Argentina

[4] Departamento de Informática - Facultad de Ciencias Físico-Matemáticas y Naturales - Universidad de San Luis – San Luis – Argentina

cortezalberto@gmail.com, carlos.martinez@frm.utn.edu.ar,

claudialaboral.gmail.com, raul.varela@frm.utn.edu.ar, jdpatrizio@hotmail.com

agaris@unsl.edu.ar, avazquez@frm.utn.edu.ar

Abstract. *Developing microservices-based architectures often involves tedious development tasks, such as establishing interface-based communications between services or configuring infrastructure components. Therefore, this article introduces the ColmenaTool tool, which is designed to facilitate the necessary development work through graphical modeling and code generation. It contains three main components: ColmenaML as a unit based on ColmenaMLI, ColmenaVisual that allows the modeling of microservices system diagrams, and ColmenaGenerator that produces automatic code generation and generates systems architecture.*

Keywords: MDE – Code Generators – MDA – Sirius - Microservices

Resumen. *El desarrollo de arquitecturas basadas en microservicios suele implicar tediosas tareas de desarrollo, como establecer comunicaciones basadas en interfaces entre servicios o configurar componentes de infraestructura. Por lo tanto, en este artículo se presenta la herramienta ColmenaTool, que está diseñada para facilitar el trabajo de desarrollo necesario mediante el modelado gráfico y la generación de código. Contiene tres componentes principales: ColmenaML como unidad basada en ColmenaMLI, ColmenaVisual que permite el modelado de diagramas de sistemas de microservicios, y ColmenaGenerador que produce la generación de código automática y genera arquitectura de sistemas.*

Palabras Claves: MDE – Generadores de Código – MDA – Sirius - Microservicios

1. Introducción

El principal objetivo de los sistemas basados en microservicios es descomponer grandes proyectos de software en pequeñas unidades desacopladas que se comunican entre sí mediante una interfaz sencilla. Constituye un diseño arquitectónico contrario al enfoque tradicional y monolítico sobre las aplicaciones, en el que todo se crea en una única pieza.

En los últimos años, muchas empresas están adoptando la arquitectura de microservicios para transformar sus modelos de software existentes. Generalmente, estos sistemas son divididos en múltiples componentes independientes, heterogéneos, que corren en su propia plataforma y además representan una unidad lógica de negocio. Los equipos se organizan alrededor de la funcionalidad, y son capaces de elegir su propia tecnología o lenguaje de programación. Estos equipos son multifuncionales porque se ocupan de cada fase del ciclo de vida, incluyendo el desarrollo, las pruebas y el despliegue en entornos de ejecución.

La Arquitectura de Microservicio (por sus siglas en inglés MSA) representa un estilo arquitectónico cada vez más popular para sistemas distribuidos y basados en servicios [Francesco et al. 2017]. Como tal, MSA se basa en el concepto de servicio como el bloque de construcción arquitectónico fundamental para la arquitectura de un sistema. Cada uno de los servicios de una MSA es altamente cohesivo y encapsula una capacidad empresarial única que, por lo tanto, da lugar al término microservicio. Desde un punto de vista técnico, un microservicio se realiza como un proceso independiente que se puede diseñar, desarrollar, implementar y operar de forma autónoma [Newman 2015].

Entre los nuevos conceptos que están ligados al término microservicios, cabe destacar la tecnología de contenedores y la metodología DevOps [Nadareishvili et al. 2016]. El despliegue y puesta en operación de cada microservicio se realiza mediante contenedores que facilitan la portabilidad, la independencia y la escalabilidad. También nacen nuevas metodologías que se ajustan a las características intrínsecas de una aplicación basada en microservicios. El término DevOps, que constituye un nuevo método más adecuado para el desarrollo y el despliegue de los microservicios, de forma ágil y eficiente. Esto también conlleva consecuencias organizativas para un proceso de desarrollo de MSA. Cada microservicio está alineado organizacionalmente con exactamente un equipo que generalmente practica DevOps. Para realizar capacidades comerciales más complejas, muchos microservicios pueden colaborar de manera coreográfica y formar cadenas de llamadas de servicio basándose en interfaces simples.

Si bien los microservicios son independientes, esta característica no significa que estén aislados. Deben proveer interfaces que sean conocidas por los clientes (ya sean otros servicios o usuarios finales) sin tener conocimiento de su ubicación ni de la tecnología subyacente. Por lo tanto, ofrecen API's que utilizan protocolos de comunicación estándares como HTTP, bajo las tecnologías REST e incluso SOAP (Web Services), así como los mecanismos de intercambio de datos como XML y JSON. Por lo tanto, la interacción del servicio es generalmente sin estado y utiliza protocolos como HTTP o MQTT1 [Newman 2015]. Las aplicaciones de MSA resultantes son, entre otras características, tanto vertical como horizontalmente escalables, flexiblemente extensibles y tienen ciclos de lanzamiento cortos, lo que las hace especialmente adecuadas para grandes aplicaciones en la nube como Spotify o Netflix . Como resultado del

acoplamiento flexible de MSA, las ventajas como escalabilidad, resiliencia y extensibilidad naturalmente tienen el costo de un mayor esfuerzo de desarrollo, especialmente en las primeras etapas del desarrollo de MSA en comparación con un enfoque monolítico [Newman 2015]. Por ejemplo, los MSA necesitan componentes de infraestructura adicionales [Wizenty et al. 2017], por ejemplo, para configurar, descubrir o monitorear los servicios de un sistema. Además, la comunicación exige una tediosa implementación de código adicional para proporcionar y consumir API o manejar errores. En este artículo, presentamos la herramienta de investigación prototípica ColmenaTool que tiene como objetivo facilitar el esfuerzo de la elaboración manual redundante y engorrosa de la implementación en el proceso de desarrollo de MSA aprovechando la Ingeniería basada en modelos (por sus siglas en inglés MDE). MDE denota el uso de modelos como entidades de primera clase en el proceso de desarrollo de software [France y Rumpe 2007]. ColmenaTool permite a los desarrolladores crear diagramas de su entorno de microservicios previsto y es capaz de generar bases de sistema preconfiguradas basadas en la información del diagrama. En consecuencia, ColmenaTool comprende un lenguaje de modelado específico de dominio gráfico ligero (por sus siglas en inglés DSL) [Wegeler et al. 2013], un editor basado en Eclipse y un generador para realizar transformaciones de modelo a código (M2C). El código fuente resultante se basa actualmente en Java y el marco de Spring Cloud Netflix [Spring 2021] y el FrontEnd en Angular[Angular 2021].

El documento está organizado de la siguiente manera. La sección 2 describe el Estado del arte y los trabajos relacionados, continuando la sección 3 se presenta la Arquitectura del modelado específico del dominio, posteriormente en la sección 4 se presenta la Herramienta ColmenaTool, en el capítulo 5 se establecen las conclusiones y los trabajos futuros.

2. Estado del Arte y Trabajos relacionados

Existen varias herramientas que permiten generación de MSA. Una de las más populares y maduras es la plataforma JHipster [Jhipster 2021]. Posibilita la generación de MSA basado en un formulario web y viene con un lenguaje específico de dominio textual para expresar las entidades de servicios y sus relaciones. El objetivo de JHipster radica en el aprovisionamiento de una arquitectura completamente funcional con FrontEnd y componentes de BackEnd. Tecnológicamente, genera el BackEnd en Spring Framewok, Micronaut, Node.js entre otros y permite varias tecnologías posibles para realización de FrontEnd como Angular.

En contraste con el formulario web de JHipster y el modelado de entidades textuales, ColmenaTool comprende un DSL gráfico completo dedicado a MSA lo que brinda la posibilidad de modelar una base de sistema completo. Por tanto, los modelos resultantes contienen más información y permiten la generación de la arquitectura sin demasiado conocimiento técnico y complicación modelando de manera textual. Finalmente, ColmenaTool tiene como objetivo proporcionar componentes de BackEnd y FrontEnd completamente funcionales y ayudar a un desarrollador a omitir engorrosos y redundantes procesos repetitivos de codificación al crear su propio MSA personalizado.

Otra herramienta que tiene como objetivo aliviar el costo de desarrollo de MSA es la herramienta MAGMA [Wizenty et al. 2017]. MAGMA es una herramienta de gestión de construcción que se basa en el mecanismo de arquetipo de Maven. Puede crear

bases de microservicios basadas en plantillas de servicios predefinidas. Sin embargo, como MAGMA se basa en Maven, no contiene capacidades de modelado y los microservicios generados individualmente no se pueden relacionar

Por último, encontramos a Ajil [Sorgaila et al. 2018] una herramienta que sigue nuestra línea de investigación, que tiene un enfoque metodológico y de lineamientos similares a los seguidos por nuestro grupo e incluye artefactos de software similares a ColmenaTool. Sin embargo, encontramos que sólo genera exitosamente el modelado y código exclusivamente sobre el ecosistema de Java Cloud Nexflix. ColmenaTool pertenece a un ecosistema de herramientas y procesos de Agilidad [Cortez et al. 2019] a la luz de MDE, que coopera con otras herramientas en desarrollo como ColmenaReverse que se encarga de realizar un análisis sintáctico de código y convertir código en Java a archivos XMI (en inglés XML Metadata Interchange) [XMI 2021], que son posteriormente leídos por ColmenaVisual en forma automática posibilitando la actualización del modelo de clases en forma dinámica.

3. Arquitectura de Solución de Modelado Específico de Dominio

Con la incorporación de conceptos específicos de dominio y experiencia de desarrollo de alta calidad en MDE, las tecnologías pueden mejorar significativamente la productividad de los expertos en el dominio y la calidad del sistema. Esta realización ha llevado a trabajar, a partir de finales de los noventa, en workbench de trabajos de lenguaje MDE que apoyan al modelado específico de dominio (por sus siglas en inglés DSM) [Tolvanen y Kelly 2005], y herramientas asociadas (por ejemplo, editores modelo, simuladores y generadores de código). Un DSM proporciona un puente entre el espacio (problema) en el que trabajan los expertos del dominio y el espacio de implementación (programación). En este contexto, los DSL se pueden utilizar para apoyar la coordinación sociotécnica proporcionando los medios para que las partes interesadas cierren la brecha entre cómo perciben un problema y su solución, y las tecnologías de programación utilizadas para implementarla.

En el presente trabajo se ha optado por una forma típica de creación de entornos DSM, donde se parte del desarrollo de modelos a partir de conceptos aplicados por expertos del dominio. Se utilizan los mismos conceptos como input para producir generadores de código según los principios fundamentales de MDE. Para el desarrollo de una solución DSM en un contexto MDE, se utiliza una arquitectura de 3 niveles sobre la plataforma de ejecución: un lenguaje específico de dominio, un generador de código y un framework específico de dominio. Siguiendo este enfoque, la Figura 1 muestra la arquitectura propuesta para el DSM.

3.1 Visión General de un DSM

En el marco de DSM, el lenguaje específico de dominio (por sus siglas en inglés DSL) es la herramienta que permite abstraer la complejidad de un dominio dado, a través de la provisión de conceptos y reglas expresados directamente en el dominio del problema. Un DSL se define como “un lenguaje de programación de limitada expresividad focalizado en un dominio en particular” [Tolvanen y Kelly 2005].

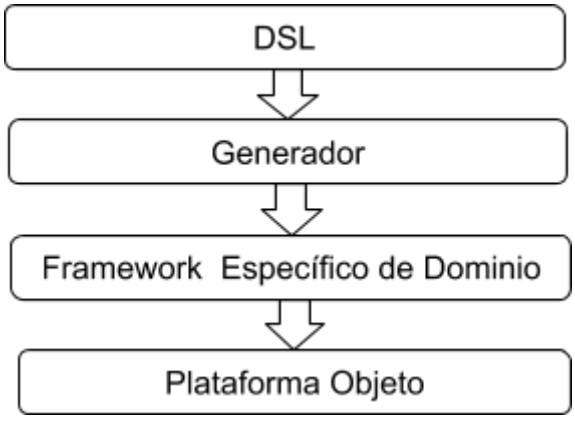


Figura 1. Arquitectura utilizada para el entorno DSML

Los conceptos principales suelen tener su representación en la notación del lenguaje (gráfica o textual), otros conceptos se representan mediante conexiones o propiedades. En este trabajo se propone una notación gráfica; el lenguaje se formalizó con un metamodelo que define una sintaxis abstracta. Sobre él se definió una sintaxis concreta con la cual el experto del dominio interactúa. El generador de código especifica cómo se extrae la información de los modelos y se transforma en código ejecutable que no necesita ser modificado para su funcionamiento. El Framework específico de dominio es una interfaz entre el código generado y la plataforma subyacente. Su objetivo es simplificar la tarea del generador, eliminando la duplicación de código y proveyendo de una capa de abstracción sobre la plataforma. Al proveer de componentes previamente validados, se permite un aseguramiento de la calidad del producto. La plataforma comprende el conjunto de sistema operativo, máquina virtual, lenguajes, librerías y frameworks de componentes.

3.2 Componentes de un DSL

3.2.1 Sintaxis abstracta

La sintaxis abstracta de un lenguaje representa la terminología de los conceptos del lenguaje, las relaciones entre ellos, y las reglas necesarias para construir las sentencias (programas, instrucciones, expresiones o modelos) válidas del lenguaje.

3.2.2 Sintaxis concreta

La sintaxis concreta de un lenguaje detalla la notación usada para representar los modelos. Hay dos tipos de sintaxis concretas: textuales y gráficas. La sintaxis gráfica (o visual) describe los modelos de forma diagramática, usando símbolos para representar sus elementos y las relaciones entre ellos. La sintaxis textual permite describir modelos usando sentencias compuestas por cadenas de caracteres, de una forma similar a como lo hacen la mayoría de los lenguajes de programación.

3.3 Semántica

En general, lo que parece representar un modelo no es lo mismo que lo que realmente significa. La semántica representa el significado [Combemale et al. 2016].

3.4 Relaciones entre sintaxis abstracta, sintaxis concreta y semántica

Mientras que la sintaxis abstracta especifica cuáles son los modelos válidos, la sintaxis concreta permite representar dichos modelos. Por su parte, la semántica les asigna un significado preciso y no ambiguo. La forma natural de definir la sintaxis abstracta de los DSL es mediante metamodelos. La sintaxis concreta de un lenguaje se suele definir a partir de su sintaxis abstracta, mediante una relación (mapeo de sintaxis concreta) que asigna un símbolo de la sintaxis concreta a cada concepto de la sintaxis abstracta. Dicha asignación debe respetar la semántica que suelen tener los símbolos.

3.5 Metamodelado

Los metamodelos sirven para representar los conceptos de un lenguaje, las relaciones entre ellos y las reglas estructurales que restringen los posibles elementos de los modelos válidos.

3.6 Sintaxis concreta para metamodelos

La sintaxis concreta proporciona la notación que permite a los diseñadores describir los modelos. La sintaxis concreta puede definirse desde cero, o bien haciendo uso de una notación existente, extendiéndola al agregar los conceptos de nuestro lenguaje.

3.7 Validación de Metamodelos

OCL (por sus siglas en inglés, Object Constraint Language) [OCL 2021], es un lenguaje que utiliza la lógica de primer orden con el fin de otorgar formalidad y precisión tanto a modelos como a metamodelos. Un metamodelo se expresa con notación gráfica y con un lenguaje formal, como OCL, para aumentar su precisión y eliminar ambigüedades. OCL es utilizado para definir la semántica de UML, especificando reglas bien formadas sobre el metamodelo. Una de sus principales características es su fácil comprensión. Las reglas bien formadas son definidas como invariantes sobre las metaclases en la sintaxis abstracta.

4. Herramienta de modelado ColmenaTool

4.1 Lenguaje de modelado ColmenaModel

Como DSL, ColmenaML se compone de tres componentes fundacionales: (i) sintaxis abstracta, (ii) sintaxis concreta y (iii) semántica.

4.1.1 Sintaxis abstracta.

Eclipse Modeling Framework (por sus siglas en inglés EMF) [Eclipse Modeling Framework 2021], es un conjunto de complementos de Eclipse que se pueden utilizar para modelar un modelo de datos y generar código u otro resultado basado en este modo. EMF tiene una distinción entre el metamodelo y el modelo real. El metamodelo describe la estructura del modelo. Un modelo es una instancia concreta de este metamodelo.

EMF permite al desarrollador crear el metamodelo a través de diferentes medios, por ejemplo, XMI, anotaciones Java, UML o un esquema XML. También permite

conservar los datos del modelo; la implementación predeterminada utiliza un formato de datos llamado Intercambio de metadatos XML.

Para diseñar un metamodelo se selecciona el IDE Eclipse, dentro de este se utiliza el framework EMF. El proceso de diseño del metamodelo es el siguiente: 1) crear un proyecto EMF, para lo cual se selecciona la opción Empty EMF Project; 2) colocar un nombre al proyecto; 3) en la carpeta model se debe crear un modelo Ecore; 4) colocar un nombre al metamodelo; 5) diseñar el metamodelo con los diferentes elementos constitutivos, como EClass, EReferences, EAttributes, etc.

EMF permite ampliar los modelos existentes mediante herencia. La sintaxis abstracta se derivó en metamodelo base y una extensión basada en este metamodelo base. También se puede trabajar con modelos EMF ecore directamente sin usar las herramientas ecore. La sintaxis se representa como un metamodelo en la Figura 2, que representa el modelo detallado de funcionalidad de la lógica de negocios del microservicios usando diagramas de clases de ecore. En la Figura 3 se representa el metamodelo del ecosistema de cooperación de microservicios (Nextflix) que extiende del anterior, donde se describen los conceptos de microservicios funcionales y de infraestructura, expresando de este modo la arquitectura.

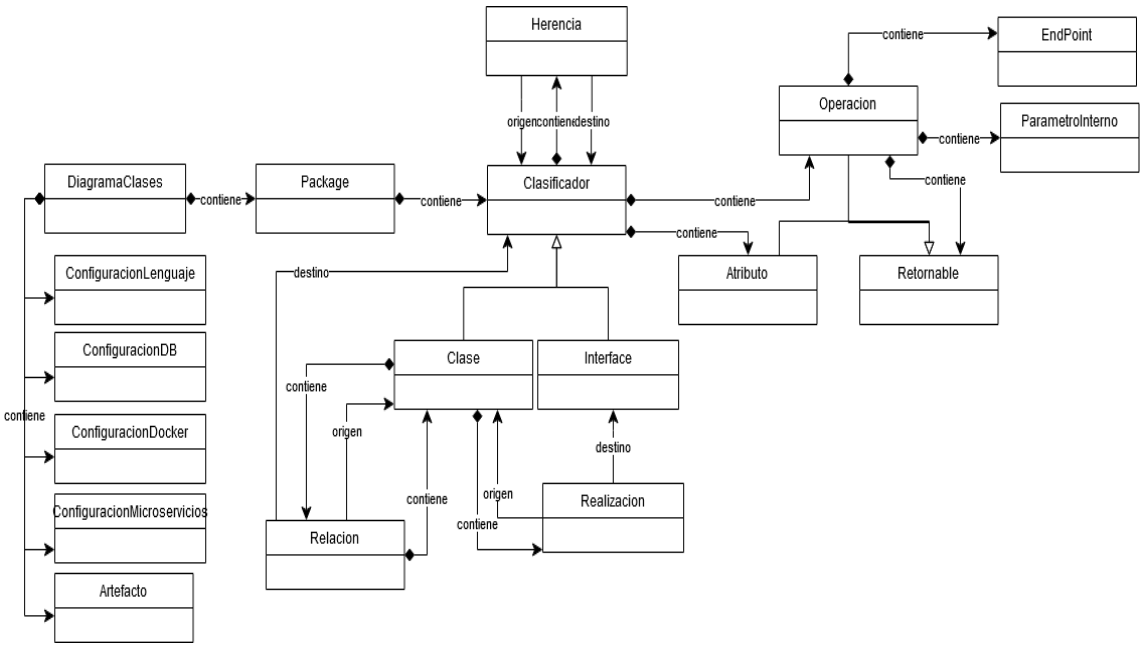


Figura 2. Metamodelo simplificado de funcionalidad de negocios

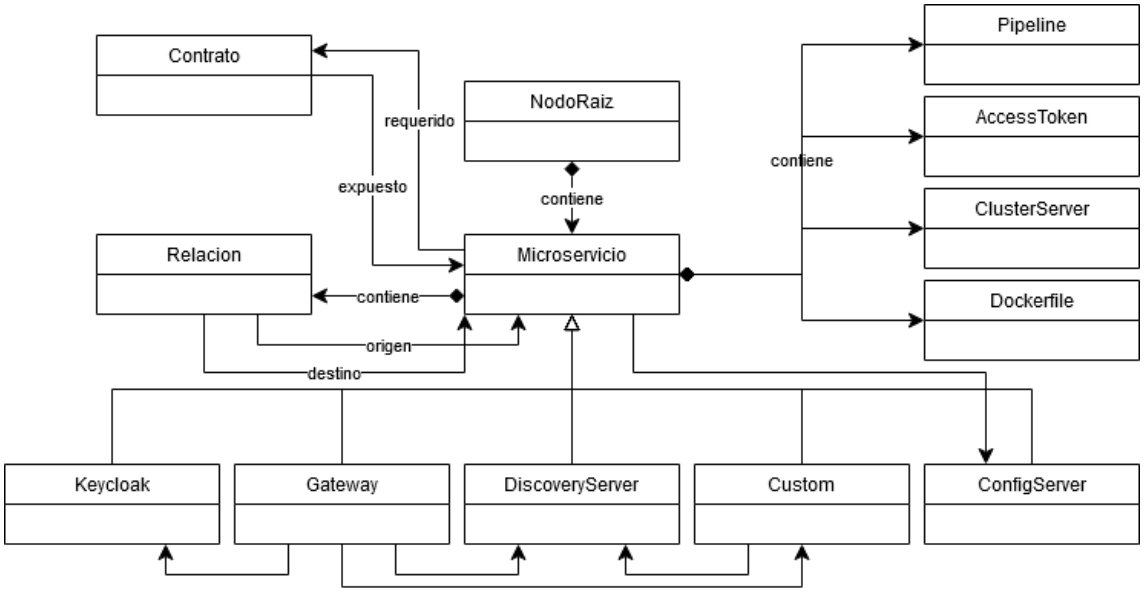


Figura 3. Metamodelo de infraestructura

Aunque creamos ColmenaTool dirigido a la plataforma Spring Cloud, es factible por el diseño de estos metamodelos agregar en un futuro generadores para otras plataformas y tecnologías de BackEnd y FrontEnd. Por ejemplo, Node:js, Micronaut, Go y tecnologías del FrontEnd como React.js y Vue.

En el paquete superior que representa al lenguaje ColmenaML en la Figura 2 se representan los conceptos básicos de clases, atributos, operaciones, relaciones y configuraciones especiales. A partir del mismo se pueden generar instancias de modelado del dominio de negocios que representan la lógica de funcionalidad del microservicio que se pretende diseñar y exponer la Api. Por otra parte, el paquete que extiende se denomina ColmenaMLI, que permite crear instancias del dominio del ecosistema de infraestructura, es decir los modelos que de él se desprendan podrán representar la arquitectura del ecosistema de Netflix, expresando así las relaciones entre los microservicios funcionales y de infraestructura (Gateway, Discovery, Config Server y Keycloak).

4.1.2 Sintaxis concreta.

Tomando como base los metamodelos anteriores, se diseña la forma gráfica que tendrá cada uno de los elementos del metamodelo. Se utilizó Sirius [Sirius 2021], que permite la especificación breve y simple de un workbench de modelado en términos de “vistas”, y que puede tener la forma de editores gráficos, tablas o árboles, con reglas de validación y acciones usando descripciones declarativas.

El proceso de creación de un proyecto Sirius, considera las siguientes actividades: 1) creación de un proyecto Viewpoint Specification Project, 2) especificar un Viewpoint, 3) especificar el tipo de Representación (Diagram, Edition Table, Cross Table,Tree, Sequence Diagram), 3) mapeo entre los elementos gráficos del diagrama con los elementos del metamodelo, 4) especificar las acciones que se van a realizar cuando se produzca un evento, y 5) especificar los elementos de la barra de herramientas del editor (paleta). La figura 4 muestra el entorno de configuración de un punto de vista.

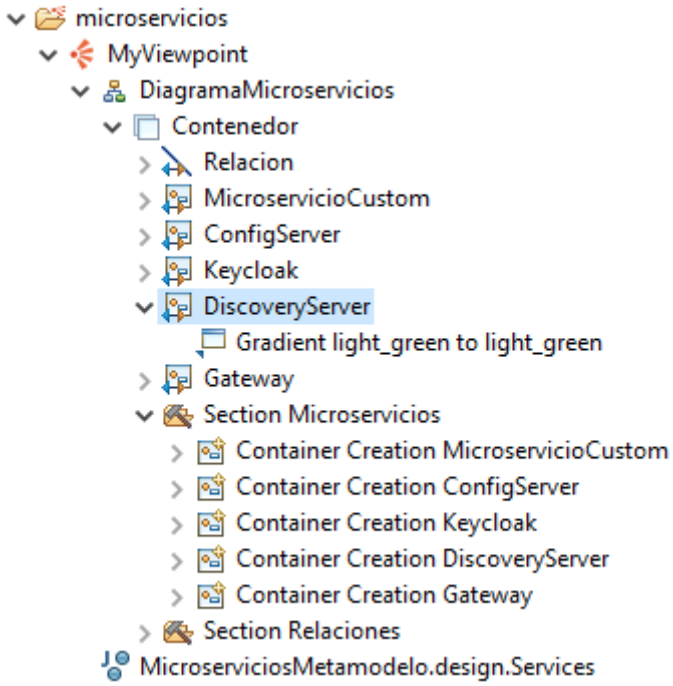


Figura 4. Punto de vista

La notación gráfica se divide en dos diferentes tipos de diagramas que colaboran juntos: el diagrama de infraestructura del modelado que se muestra en la Figura 5 y el diagrama de funcionalidad de modelado de cada microservicio que se puede ver en la Figura 6.

Los diagramas de infraestructura representan las interacciones entre los servicios y, por lo tanto, sólo visualiza las partes externas, es decir, los servicios y sus puntos finales representados como rectángulos y elipses con colores específicos que expresa el lenguaje, así como el servicio relaciones representadas a través de los flechas. El tipo de cada servicio se distingue por su color y forma. Por ejemplo, el diagrama de arquitectura de microservicios que se muestra en la Figura 5 contiene dos servicios funcionales marcados como rectángulos de color lila, un Discovery Service marcado en verde y una elipse; un API Gateway marcado con una elipse y color naranja tenue; Keycloak para la seguridad con un rectángulo celeste y por último ConfigServer con un rectángulo marrón claro. De esta forma se puede visualizar la MSA de una forma ágil y clara, la cual otorga grandes beneficios al modelador.

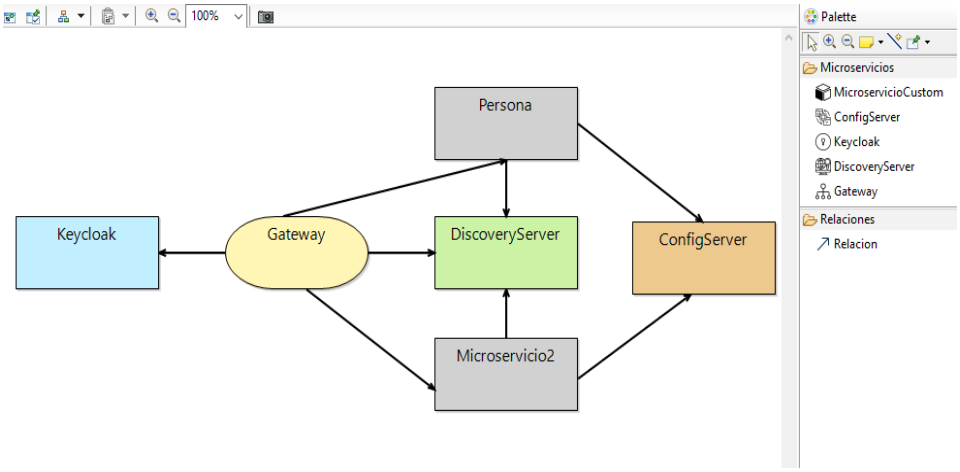


Figura 5. Diagrama de Infraestructura de microservicios

A diferencia de los diagramas de infraestructura, los diagramas de funcionalidad visualizan partes internas de un determinado servicio funcional, por ejemplo, la Figura 5 muestra las partes internas de la ApiPersona presentadas en la descripción general del diagrama. Su modelo de dominio ApiPersona consta de cuatro entidades de las cuales la entidad Persona, gráficamente se puede modelar cambiando de color, esto significa que la Api persona será expuesta como un controlador con los métodos necesarios para consumir ese endpoint. Las demás clases del dominio cooperan en la funcionalidad del diseño del microservicio.

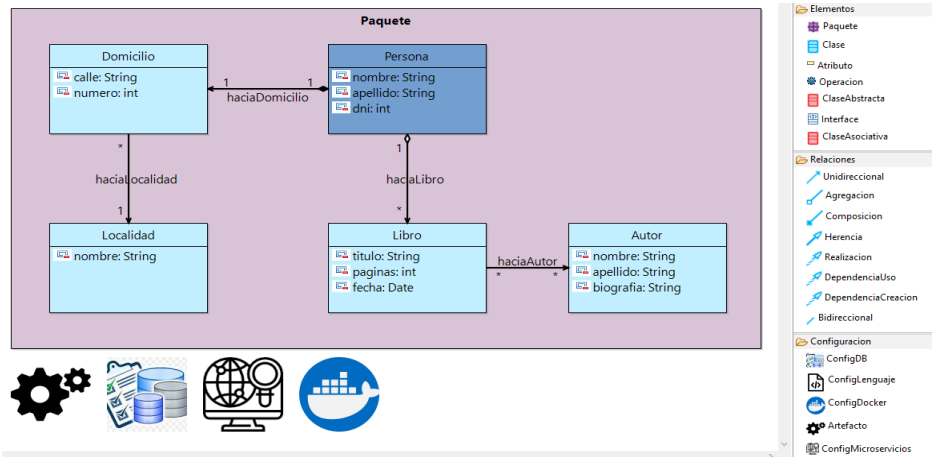


Figura 6. Diagrama de funcionalidad de microservicios

Por lo tanto, un modelo MSA anotado en ColmenaML siempre consta de un diagrama de infraestructura y varios diagramas de funcionalidad dependiendo de la cantidad de microservicios funcionales que se modelen.

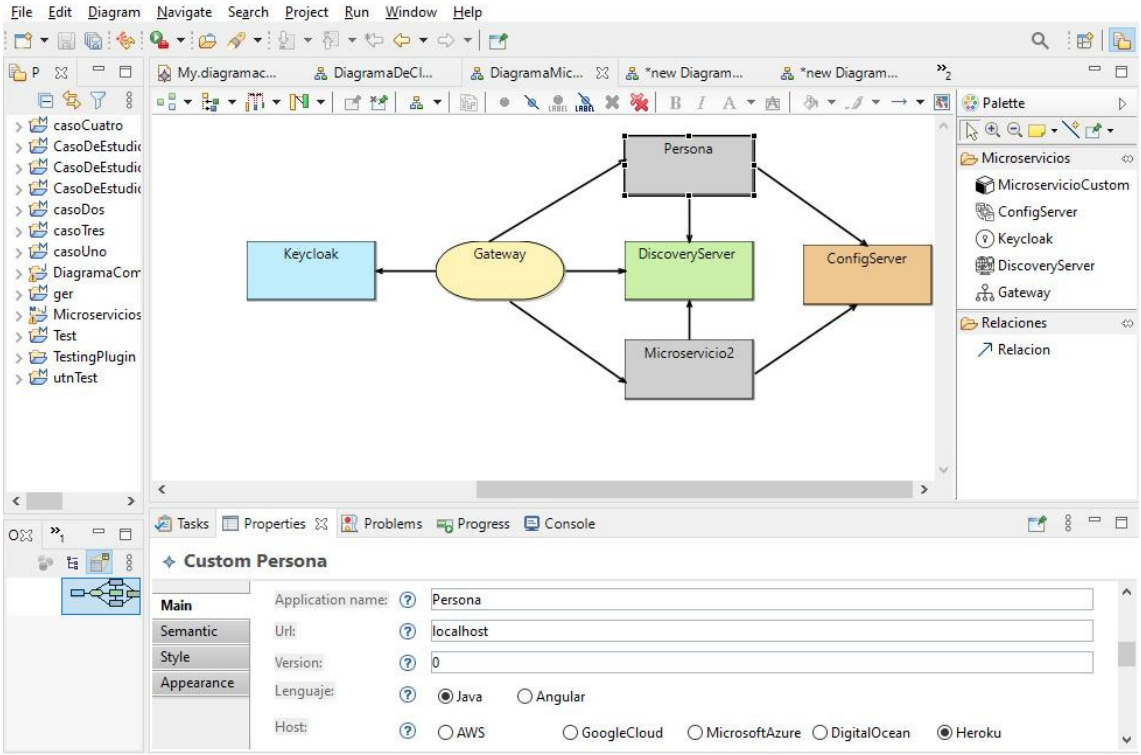
4.1.3 Semántica.

Además de la composición de la sintaxis abstracta de ColmenaML y ColmenaMLI que se muestran en la Figura 5 y Figura 6 respectivamente, el metamodelo está enriquecido con varias invariantes que describen las restricciones que deben obedecer las instancias bien formadas del metamodelo, es decir, los modelos ColmenaML. Por ejemplo, las relaciones entre entidades solo se permiten dentro de los límites del mismo servicio funcional. Las restricciones forman la sintaxis sintáctica de la semántica y están formulados en el lenguaje OCL.

4.1.4 ColmenaVisual

ColmenaVisual le brinda al diseñador la capacidad de crear diagramas ColmenaMLI a través de un workbeanch de Eclipse Sirius, que se basa en el estándar de Eclipse Modeling Framework). En el contexto de EMF, cada paquete de sintaxis abstracta se realiza como un modelo Ecore separado. Para proporcionar la capacidad de modelado gráfico, ColmenaVisual utiliza el marco Eclipse Sirius. Como se muestra en la Figura 4, el workbench de trabajo ColmenaVisual contiene un área para representar los diagramas, una ventana de propiedades para manipular atributos, por ejemplo, el nombre del microservicio, tipo de microservicio o el puerto de un microservicio, y una barra de herramientas para agregar nuevos elementos al modelo, por ejemplo, servicios o relaciones entre ellos. Además, es posible navegar desde diagramas de infraestructura a diagramas de funcionalidad de microservicios, así como comprobar si hay restricciones de OCL violadas. Por ejemplo, se visualizan con círculos rojos de alerta que indican que se viola alguna restricción en el diseño, por lo que la información detallada es visible en una descripción emergente y el proyecto se marca como defectuoso.

El presente caso de estudio muestra la primera instancia de modelado de un proyecto de software. En este caso se presenta la funcionalidad de devolución de un libro a una biblioteca. En principio se modelará la funcionalidad que representa al ecosistema de infraestructura Figura 7 y luego se modelará el diagrama de clases de este microservicio funcional. Para modelar dicha API Rest se utiliza la herramienta ColmenaVisual generada en Sirius como puede observarse en la Figura 8.



.

Figura 7. Diagrama infraestructura del modelo

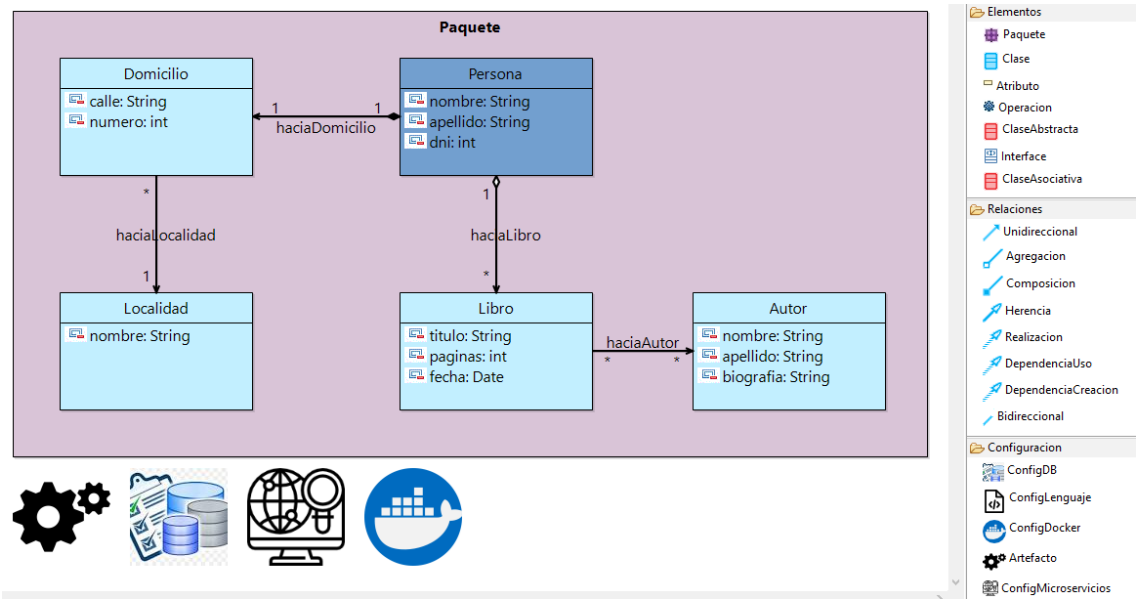


Figura 8. Diagrama de funcionalidad de Api de Biblioteca

4.1.5 ColmenaGenerador

ColmenaGenerador es responsable de transformar los modelos ColmenaML en código fuente. El generador se basa en plantillas y utiliza el framework de Acceleo [Acceleo 2021] para la transformación de modelo a texto. Acceleo es una implementación del

estándar Model-to-Text Language (por sus siglas en inglés MTL), definido por Object Management Group (por sus siglas en inglés OMG) [OMG 2021]. La base del sistema de microservicio generado se basa en el ecosistema de tecnología Spring Cloud y utiliza Java como lenguaje de programación. Como herramienta de gestión de compilación se utiliza Maven. A cada servicio se le asigna un archivo pom separado, de modo que las dependencias se pueden administrar específicamente para cada servicio. Los servicios de infraestructura modelable se realizan actualmente con una estructura estática que proporciona las funciones específicas del rol, por ejemplo, el tipo de servicio de descubrimiento da como resultado un servicio Spring Cloud Eureka. Los servicios funcionales se generan dinámicamente según su diagrama de funcionalidad, de iguales principios que en [Sorgaila et.al.13]. Tienen una estructura de archivos estandarizada, que se basa en el principio de tres capas, incluida la capa de acceso a datos, la capa de lógica empresarial y la capa de presentación, como se muestra en la Figura 9 para el servicio al cliente. La ilustración muestra a modo de ejemplo la estructura de archivo típica de un servicio funcional generado con ColmenaGenerador. Todas las interfaces generadas del servicio se realizan como controladores REST y se encuentran en el paquete Controlador. Las entidades generadas se asignan al paquete de Entidades. Las clases que realizan la lógica empresarial o el acceso a la base de datos se encuentran en el servicio de paquetes o repositorio respectivamente. La configuración del microservicio se realiza a través del archivo properties de Spring Boot. Por otro lado, la Figura 10 expresa la estructura de archivos generada para el FrontEnd en el lenguaje Angular.

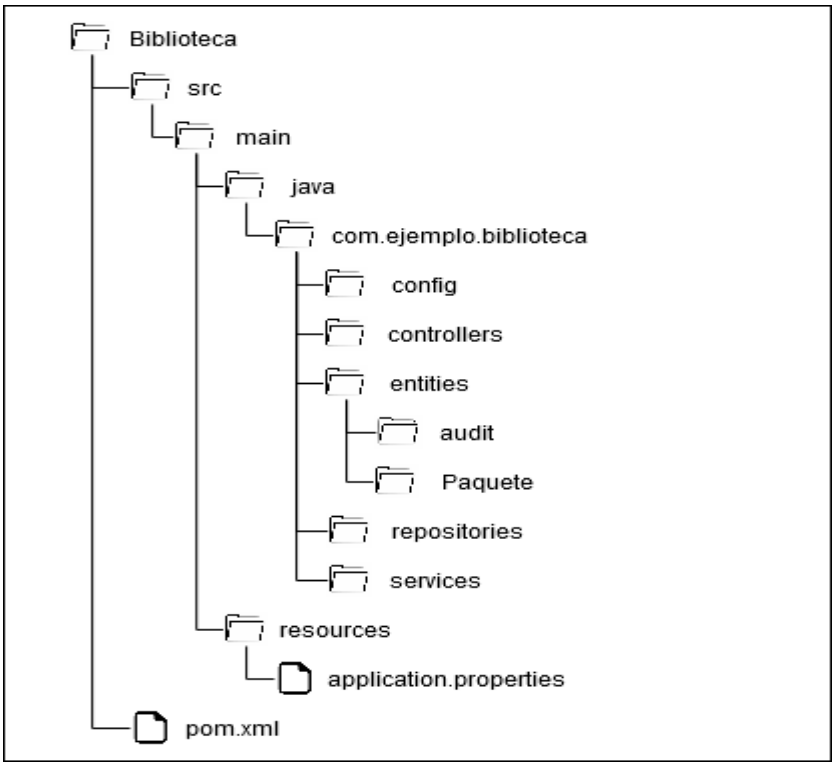


Figura 9. Diagrama de estructura generada de archivos para el BackEnd

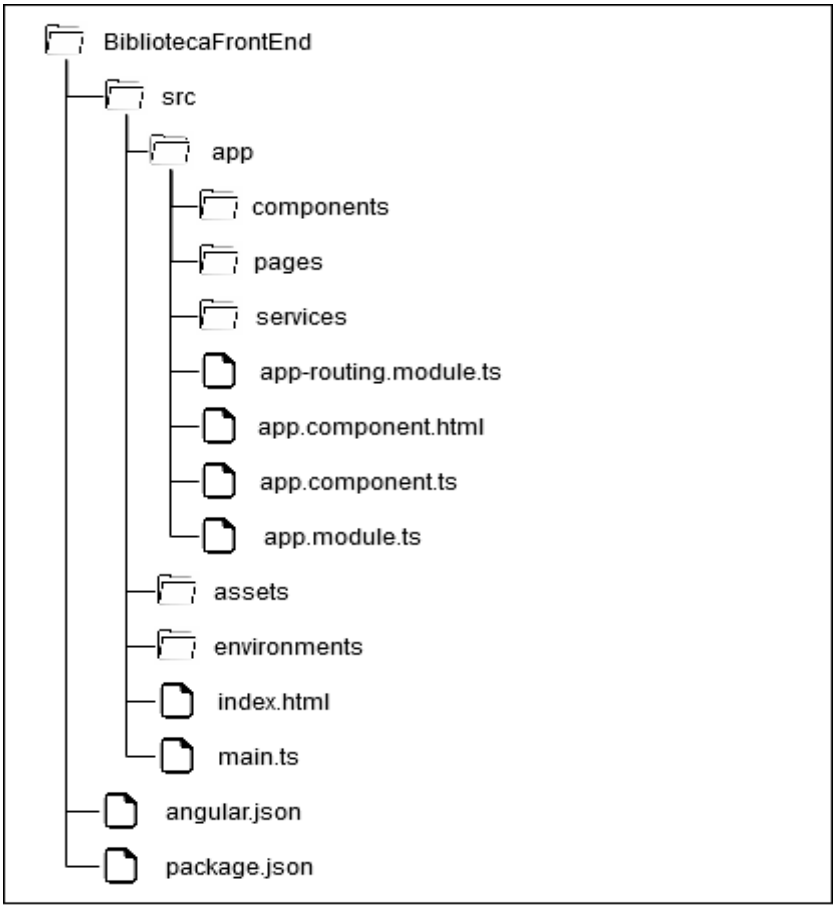


Figura 10. Diagrama de estructura generada de archivos para el FrontEnd

5. Conclusiones y trabajos Futuros

Este artículo presenta la herramienta Colmena Tool que integra el ecosistema de herramientas propuestas a la luz de MDE basada en modelos, que pertenece a la línea de investigación de "Desarrollo Dirigido por Modelos en entornos Ágiles", se desarrolla en el marco del proyecto Interinstitucional conformado por el Instituto de Informática de la Universidad de Mendoza, el GRUPO AuSegTIC (Grupo de Auditoría y Seguridad de TIC), de la Facultad Regional Mendoza de la Universidad Tecnológica Nacional y la Universidad del Aconcagua. Dicho acuerdo es parte de un proceso de incentivación para el desarrollo de actividades I/D.

ColmenaTool permite a los desarrolladores crear una arquitectura basada en microservicios y crear la generación de código automático para las instancias de BackEnd y FrontEnd respectivamente, con el fin de evitar la complicada elaboración manual redundante de la implementación. ColmenaTool se encuentra en GitHub [Sitio de GitHub 2021]. En dicho repositorio se detalla en el archivo readme una descripción general del proyecto, también se proporcionan en nuestro grupo de investigación un Canal de YouTube ColmenaTec [Sitio de ColmenaTec 2021]. Utilizamos ColmenaTool con éxito en proyectos de investigación, así como conferencias académicas. Basados en nuestras experiencias, creemos que el desarrollo de microservicios impulsado por modelos tiene un gran potencial para aumentar aún más la productividad y facilitar el esfuerzo de

desarrollo de MSA. Sin embargo, como herramienta de investigación, ColmenaTool aún no ha alcanzado la madurez de herramientas relacionadas como JHipster. Nuestro objetivo es abordar este problema en el futuro proporcionando esta herramienta a un grupo reducido de empresas productoras de Software de nuestra provincia. Con el objeto de obtener el feedback correspondiente de su uso. Por otro lado, el desafío, será generar nuevas plantillas para el BackEnd y FrontEnd, utilizando tecnologías como Node.js, .Net, Go , Vue y React en un mediano plazo.

6. Referencias

Acceleo. Disponible en: https://www.eclipse.org/acceleo/ (Último acceso 1/06/2021).

Angular. Disponible en: https://angular.io/ (Último acceso 1/06/2021).

ColmenaTec. Disponible en: https://github.com/mdd-utn?tab=repositories. (Último acceso 1/06/2021).

Combemale, B., France, R., Jézéquel, J., Rumpe, B., Steel, J. and Vojtisek , D. (2016) Engineering modeling languages. Taylor & Francis, CRC Press, Boca Raton.

Cortez, A., Martínez, C., Naveda, C., Luna, M, Vazquez, A. (2019). "Un proceso para desarrollo dirigido por modelos en entornos ágiles". XXI Workshop de Investigadores en Ciencias de la Computación. Facultad de Ciencias Exactas, Físicas y Naturales, Universidad Nacional de San Juan.

Eclipse Modeling Framework. Disponible en: https://www.eclipse.org/modeling/emf/ .(Último acceso 1/06/2021)

France, R. and Rumpe, R. (2007) Model-driven Development of Complex Software: A Research Roadmap. In 2007 Future of Software Engineering.

Francesco, P., Lago, P. and Malavolta,I. (2017) Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In 2017 IEEE Int. Conf. on Software Architecture.

GitHub. Disponible en: https://github.com/mdd-utn?tab=repositories. (Último acceso 8/03/2021).

Jhipster. Disponible en: https://www.jhipster.tech/(Último acceso 1/06/2021).

Newman, S. (2015) Building Microservices. O'Reilly Media, Inc.

OCL. Disponible en: https://www.omg.org/spec/OCL/About-OCL/. (Último acceso 8/02/2021).

OMG. Disponible en: https://www.omg.org/ (Último acceso 1/06/2021).

Singleton, A. (2016). The Economics of Microservices. IEEE Cloud Computing 3, 5.

Sirius. Disponible en: https://www.eclipse.org/sirius/. (Último acceso 1/06/2021).

Spring.io. Disponible en: https://spring.io/cloud (Último acceso 1/06/2021)

XMI. Disponible en: https://www.omg.org/spec/XMI/About-XMI/. (Último acceso 1/06/2021).

Sorgaila, J., Wizenty, P., Rademacher, F., Sachweh, S., & Zündorf, A. (2018). AjiL - enabling model-driven microservice development Jonas. In Proceedings of the 12th European Conference on Software Architecture Companion Proceedings –

Tolvanen, J., Kelly, S. (2005) Defining Domain-Specific Modeling Languages to Automate Product Derivation. Software Product Lines Vol. 3714, pp 198-209. Springer

Nadareishvili, I., Mitra, R., McLarty, M. and Amundsen, M. (2016) Microservice Architecture: Aligning Principles, Practices, and Culture. O'Reilly Media, Inc.

Newman, S. (2015). Building Microservices. O'Reilly Media, Inc.

Wegeler, T., Gutzeit, F., Destailleur, A., Dock, B. (2013) Evaluating the benefits of using DomainSpecific Modeling Languajes – an Experience Report. Proceedings of the 2013 ACM workshop on Domain-specific modeling, pp 7-12. ACM. New York, USA

Wizenty, P., Sorgalla, J., Rademacher, F., and Sachweh, S. (2017) MAGMA: Build Management-based Generation of Microservice Infrastructures. In Proc. of ECSA.ECSA '18, (pp. 1–4). New York, USA: ACM Press. https://doi.org/10.1145/3241403.3241406.

.