XLIV Jornadas de Automática 2023

Desarrollo de componentes para aplicaciones basadas en microservicios

Cuadra, J.[a,*], Hurtado, E.[a], López A.[a], Estévez, E.[b], Casquero, O.[a], Armentia, A.[a]

[a] *Departamento de Ingeniería de Sistemas y Automática, Universidad del País Vasco (UPV/EHU), 48013, Bilbao, España*
[b] *Departamento de Ingeniería Electrónica y Automática, Universidad de Jaén, 23071, Jaén, España*

To cite this article: Cuadra, J., Hurtado, E., López, A., Estévez, E., Casquero, O., Armentia, A., 2023. development of components for microservice based applications
XLIV Jornadas de Automática, 789-794 https://doi.org/10.17979/spudc.9788497498609.789

Resumen

La utilización de la computación en la Niebla está ganando terreno en el dominio de la fabricación avanzada. Este paradigma de computación distribuida habilita el análisis de los datos producidos por los activos de fabricación en los procesos industriales, acercando los servicios que suele ofrecer la Nube a los dispositivos que producen los datos, mejorando la seguridad de estos y el rendimiento de las comunicaciones. Las aplicaciones desplegadas en la Niebla suelen diseñarse como conjuntos de componentes distribuidos que aprovechan el paradigma de microservicios, generalmente encapsulados en contenedores, para adaptar su despliegue en nodos con capacidades heterogéneas. Este artículo hace uso de la Ingeniería Conducida por Modelos para proporcionar a los programadores de componentes pautas y herramientas a la hora de diseñar y desarrollar dichos componentes de aplicación, preparándolos para su posterior despliegue en forma de microservicios.

Palabras clave: Industria 4.0, Computación en la Niebla, Microservicios, MDE, Modelado, Contenedores, CBSE

development of components for microservice based applications

Abstract

Fog computing is getting more widespread on the advanced manufacturing domain. This distributed computing paradigm enables analysing the data produced by the manufacturing assets at the industrial processes, offering Cloud-like services closer to the devices that produce the data, improving their data security and the performance of their communications. The applications deployed on the Fog layer are usually designed as sets of distributed components that leverage the advantages of the microservice paradigm, generally encapsulated in containers, to adapt their deployment to nodes with heterogeneous capabilities. This article makes use of Model Driven Engineering to provide component programmers with tools and steps when designing and developing said application components, preparing them for their latter deployment as microservices.

Keywords: Industry 4.0, Fog Computing, Microservices, MDE, Modelling, Containers, CBSE

1. Introducción

La adopción de las últimas tendencias tecnológicas en la industria ha dado lugar al paradigma de fabricación conocido como Industria 4.0. Este paradigma tiene como objetivo, entre otros, la optimización de los procesos industriales mediante el aprovechamiento de los datos producidos por los activos de planta, permitiendo el desarrollo de aplicaciones innovadoras en el ámbito de la planificación dinámica, el mantenimiento preventivo, etc. (Belman-Lopez et al., 2020). En este contexto, se ha propuesto que los sistemas de fabricación se estructuren en un modelo de interconectividad global basado en tres niveles: 1) la Planta o Edge, compuesta de activos físicos encargados de realizar las actividades productivas; 2) la Niebla o Fog, formada por sistemas distribuidos que ofrecen recursos computacionales y de almacenamiento ubicados cerca del origen de los datos; y 3) la Nube o Cloud, que ofrece servicios equiparables a los de la niebla, pero con recursos situados en servidores externos (Dobaj et al., 2018).

Al ser el nivel con mayores recursos, en primer lugar, el despliegue de estas novedosas aplicaciones se realizó en la Nube (Ali et al., 2019). En efecto, los activos de Planta no cuentan, generalmente, con suficiente capacidad de procesamiento y almacenamiento. Aun así, debido a problemas como la aparición de latencias o vulnerabilidad de los datos, la tendencia reciente es que estas aplicaciones se desplieguen en

la Niebla, ya que cuenta con las características adecuadas para ofrecer soluciones a la mayoría de los problemas presentes en los otros niveles.

Este nivel intermedio se deriva del concepto de Fog Computing, establecido por Cisco en 2012 (Bonomi et al., 2012). Estas plataformas deben cumplir una serie de características: portabilidad, adaptabilidad, escalabilidad, interoperabilidad y reconfigurabilidad, entre otras (Dintén et al., 2021). Debido a la alta cantidad de requisitos, se han realizado intentos de estandarización del paradigma Fog Computing como FORA (Pop et al., 2021) u OpenFog ("IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing," 2018). Este último, además, fue absorbido en 2019 por el Industry IoT Consortium, que está realizando una labor de integración de este estándar en su arquitectura de referencia IIRA, recientemente actualizada (Industry IoT Consortium, 2022).

El estándar más extendido y aceptado es OpenFog, que define las aplicaciones Fog como "una colección de microservicios débilmente acoplados" ("IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing," 2018, p. 85). Este paradigma de diseño y composición de aplicaciones permite que la lógica de la aplicación se separe entre los diferentes componentes. Cada uno cumple con una función específica y para interconectarlos, poseen entradas y/o salidas con interfaces claramente definidas (Zimmermann, 2016). La definición de los límites de estos microservicios es un reto de diseño, ya que se conciben como componentes ligeros, además de escalables y desplegables de manera independiente (Zimmermann, 2016).

Este enfoque ofrece simplicidad, escalabilidad y flexibilidad en el desarrollo de aplicaciones Fog, ya que tanto el hardware como los protocolos y la funcionalidad se abstraen como nodos en el flujo, añadiendo simplicidad a la lógica de la aplicación. Algunos autores restringen este enfoque determinando que la lógica se lleva a cabo como un flujo de datos dirigido (workflow) (Giang et al., 2020).

En cuanto a su implementación, los contenedores se han convertido en el estándar de facto para empaquetar microservicios, ya que permiten una virtualización ligera a nivel de sistema operativo (Fazio et al., 2016; Vayghan et al., 2021). Aun teniendo las herramientas adecuadas, y a pesar de los intentos de normalización, hasta donde los autores conocen, el diseño y la composición de estos microservicios se aborda a menudo mediante el desarrollo de soluciones ad hoc (Taneja et al., 2019).

En este contexto, en el presente trabajo se propone un procedimiento para el diseño y desarrollo de los componentes necesarios para formar aplicaciones basadas en microservicios. El uso de la ingeniería conducida por modelos permite que se consiga la reutilización de los componentes en varias aplicaciones, automatizando parte de la implementación de estos componentes como microservicios contenerizados, para su posterior despliegue como parte de una aplicación gestionada por un orquestador.

El resto del documento está organizado de la siguiente forma: la sección 2 resume la visión general de los autores sobre el diseño y el desarrollo de aplicaciones Fog basadas en microservicios, que se basa en el uso de una herramienta gráfica previamente presentada en (Hurtado et al., 2021). En las secciones 3 y 4 se detalla cómo se aborda la propuesta del diseño y desarrollo de los componentes, respectivamente. La sección 5 se centra en la generación automatizada de la librería de componentes de la herramienta gráfica, que contiene los componentes diseñados y permite el desarrollo de los microservicios para su despliegue. Por último, la sección 6 presenta las conclusiones y el posible trabajo futuro de la propuesta.

2. Visión general del diseño y desarrollo de componentes y aplicaciones Fog

Cabe destacar que la propuesta presentada en este artículo parte del trabajo realizado en (Hurtado et al., 2021). En él, se propuso una aproximación para el diseño y desarrollo de aplicaciones Fog como un flujo de microservicios, los cuales se implementan mediante componentes contenerizados. Como herramienta gráfica, se hacía uso de Node-RED.

Node-RED es una herramienta de programación visual con el potencial gráfico para definir flujos de trabajo dirigidos, relacionando las entradas y las salidas de los nodos disponibles en su librería. Además, Node-RED proporciona mecanismos de extensión con los que añadir nodos personalizados y organizarlos dentro de la librería. Los autores consideran dos grupos de interés: programadores de componentes y diseñadores de aplicaciones. El trabajo de (Hurtado et al., 2021) se centró en dar soporte al segundo grupo, mientras que este trabajo ofrece un procedimiento para las tareas a realizar por el primero. Es decir, a pesar de proponer la automatización del diseño y desarrollo de las aplicaciones, no cubría la fase de diseño y desarrollo de los componentes que las forman. En efecto, antes de que el diseñador de aplicaciones realizara su trabajo era necesario que otro usuario generase la llamada librería de componentes Fog, añadiendo los componentes que conformarían las aplicaciones. Una vez que la aplicación era diseñada, Node-RED automatizaba su desarrollo, es decir la generación de todos sus microservicios, listos para su despliegue. Por esta razón, este trabajo supone un paso más en la dirección definida en (Hurtado et al., 2021), presentando una propuesta para el diseño y desarrollo metódico de componentes Fog, que permita la generación automática de la librería donde se almacenan, utilizando la Ingeniería Conducida por Modelos. Más concretamente, se hace uso de tecnologías XML (eXtensible Markup Language).

La Figura 1 muestra el nuevo escenario propuesto en este trabajo, donde las tareas del diseñador de aplicaciones se marcan en color verde (estas tareas son las mismas que en el trabajo previo), las tareas del programador de componentes se resaltan en color naranja (tareas en las que se centra este trabajo), y, por último, en color rojo las tareas que Node-RED automatiza. Para poder lograr la separación de intereses entre el diseñador de aplicación y el programador de componentes es necesario definir dos conceptos diferenciados: componente Fog y aplicación Fog. Estos siguen los preceptos de la Ingeniería de Software Basada en Componentes (CBSE) (Solis y Hurtado, 2020), que propone la construcción de sistemas complejos (en nuestro caso, aplicaciones Fog) mediante la composición de componentes sencillos (componentes Fog), los cuales se han desarrollado independientemente de estos sistemas. Este trabajo se centra en el concepto de componente Fog y en el primer grupo de interés mencionado.

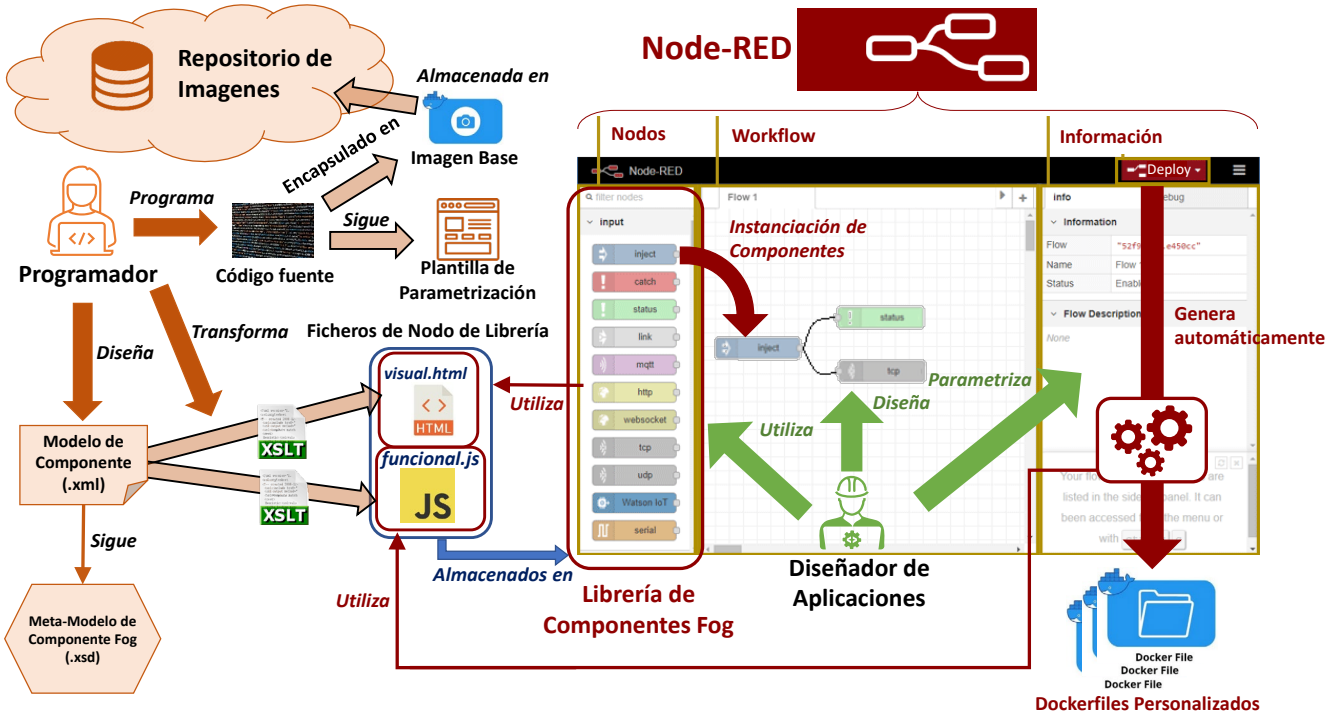


Figura 1 Escenario de diseño y desarrollo de componentes Fog integrada en el editor basado en Node-RED. En color verde se reflejan las responsabilidades del Diseñador de Aplicaciones; en color naranja, las tareas del Programador de Componentes y en color rojo las tareas que automatiza Node-RED.

Los componentes Fog se diseñan en documentos XML (modelo de componente) que siguen un meta-modelo asociado, implementado como un fichero XSD (XML Schema Definition). Partiendo de los modelos de componente Fog, los autores proponen el uso de hojas de estilo XSLT (eXtensible Stylesheet Language Transformations), las cuales permiten transformar documentos XML en otros tipos de formato, con el objetivo de generar los archivos necesarios que conforman los nodos Node-RED que representan los componentes Fog en la librería. En definitiva, para generar, de forma automática, la librería de componentes Fog.

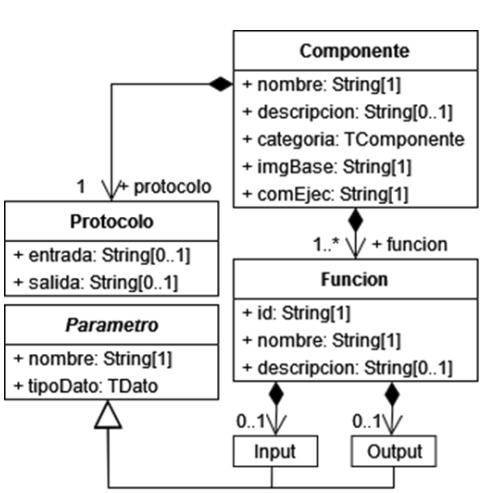
En lo que se refiere a su implementación, en este trabajo los componentes Fog son módulos software, desarrollados por el programador de componentes, previamente y de forma independiente al diseño de la aplicación Fog. Estos componentes encapsulan diferentes funcionalidades que cubren ciertos aspectos de las necesidades de negocio que se identifiquen en el dominio de aplicación.

De forma totalmente alineada con el estándar OpenFog y con su definición de aplicación Fog, las aplicaciones estarán compuestas por microservicios. De acuerdo con la Figura 1 y como parte del trabajo presentado en (Hurtado et al., 2021), el Diseñador de Aplicaciones es el encargado de diseñar las aplicaciones utilizando los componentes disponibles en la librería. Así, la instancia de componente Fog corresponderá al concepto de microservicio. Los componentes se programan de forma parametrizable en una imagen que contiene todas las funcionalidades conocidas como imagen base. Por su parte, el diseñador selecciona una de las funcionalidades del componente, determinando la función que va a ejecutar dentro de la aplicación. Por ejemplo, las funcionalidades relacionadas con la gestión de una base de datos podrán encapsularse en un mismo componente Fog, y este se podrá reutilizar en diferentes aplicaciones Fog como diferentes microservicios. Una vez diseñada la aplicación, el diseñador solicita el despliegue de la aplicación, lo que conlleva la generación automatizada de las imágenes personalizadas de los componentes.

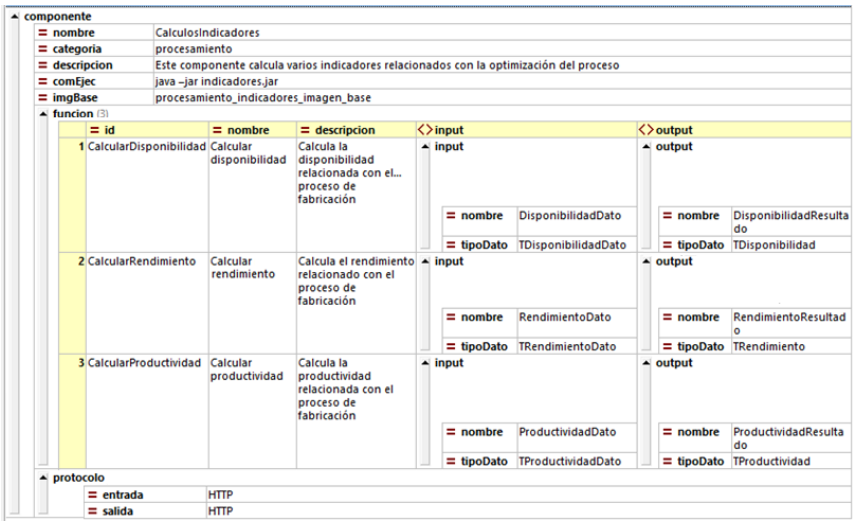
Las siguientes secciones detallan cómo debe proceder el programador de componentes para: diseñar y modelar los componentes Fog, desarrollar dichos componentes y generar los ficheros necesarios para incluir el componente en la Librería de Componentes Fog.

3. Modelado de componentes Fog

El Programador de Componentes debe generar un modelo de componente representado en un fichero XML. Este modelo debe seguir las normas y reglas de composición definidas en el meta-modelo de Componente Fog presentado en la Figura 2.a.



a) Meta-modelo de componentes Fog.



b) Modelo del componente Fog *CalculosIndicadores* implementado como fichero XML.

Figura 2 Modelado de componentes Fog: a) Meta-modelo de componentes Fog; b) Modelo de un componente Fog llamado *CalculosIndicadores*.

Cada componente Fog dispone de un nombre y de una descripción detallada. El atributo *categoria* se utiliza para definir agrupaciones dentro de la librería según el criterio del desarrollador de componentes, para facilitar su localización e instanciación. Este se determina entre un tipo numerado *TComponente* definido previamente. Los atributos *imgBase* y *comEjec* se utilizan para almacenar información relativa al desarrollo de los componentes y su posterior implementación. Cada componente puede comunicarse mediante un protocolo, determinado en el elemento *Protocolo*.

Cada componente puede ofrecer múltiples funcionalidades, (representadas en el elemento *Funcion)* y cada funcionalidad está determinada por un identificador único (atributo *id*), un nombre y una descripción detallada. Por otro lado, las funcionalidades disponen de parámetros de entrada y/o parámetros de salida, representadas por los elementos *Input* y *Output*. Estos elementos representan estructuras de datos y generalizan el elemento *Parametro,* caracterizadas por un nombre y un tipo de dato (atributo *tipoDato*) de entre un tipo enumerado *TDato* definido previamente por el programador de componentes.

Como prueba de concepto, en la Figura 2.b se muestra un componente para el cálculo de diversos indicadores de calidad. Pertenece a la categoría de componentes "procesamiento", y ofrece diferentes funcionalidades para el cómputo de varios indicadores. En este ejemplo, cada funcionalidad recibe una estructura de datos con los datos requeridos para el cálculo, y genera, como resultado, el indicador deseado, comunicándose con el resto de componentes mediante el protocolo HTTP.

La existencia de tres funcionalidades diferenciadas evidencia la reusabilidad del componente. En este caso, distintos diseñadores de aplicaciones podrían utilizar el mismo componente al instanciarlo desde la Librería de Componentes Fog, seleccionando distintas funcionalidades dependiendo del indicador a calcular. Además, en caso de necesitar calcular un nuevo indicador, el desarrollador de componentes podría tanto añadirlo a este componente como una cuarta funcionalidad, como desarrollar un componente independiente con otro conjunto de funcionalidades.

Finalmente, el desarrollador de componentes comprueba que el modelo XML generado se ajusta al meta-modelo de componentes Fog. El uso de modelos, meta-modelos y un analizador XML asegura que los componentes diseñados son correctos.

4. Desarrollo de componentes Fog

Cabe señalar que el desarrollo de estos componentes Fog está ligado con su despliegue como microservicios. En este caso, los componentes Fog se van a desplegar en contenedores, siendo Docker el estándar de facto para su generación.

Una de las primeras tareas en el desarrollo de este tipo de componentes, es generar el código fuente que guía la ejecución del programa. El código fuente del componente Fog debe disponer de la lógica para ejecutar todas las funcionalidades establecidas en la fase de diseño en el modelo del componente, siendo capaz, además, de ser personalizable por el usuario antes de ejecutarse como microservicio. Es por eso que este trabajo propone desarrollar los componentes Fog siguiendo una plantilla de parametrización. Además, se propone encapsular cada componente una vez programado en una imagen de contenedor, la cual contiene todas las funcionalidades diseñadas. Por ello, una vez finalizado el desarrollo del código, el programador debe generar la imagen base para el componente Fog, encapsulando el código parametrizado y sus dependencias.

Una vez que la aplicación está diseñada, es necesario generar una imagen personalizada relativa a cada microservicio de la aplicación, partiendo de la imagen base del componente Fog desarrollado. Así el microservicio queda listo para su despliegue. Node-RED será el encargado de generar automáticamente el fichero Dockerfile correspondiente a la imagen personalizada. No obstante, el desarrollador de componentes debe preparar la imagen base para que sea fácilmente personalizable con información relativa a la funcionalidad seleccionada durante la fase de diseño de aplicación, al instanciar el componente Fog de la librería.

La manera más sencilla e inmediata de parametrizar el código dentro de un contenedor es utilizar variables de entorno. De esta forma, se puede determinar la personalización de la ejecución de este código durante la creación de una imagen personalizada. Con este enfoque, se permite la parametrización de la ejecución del microservicio por parte del diseñador de aplicaciones.

Para ello, se han determinado, por una parte, tres variables de entorno asociadas a la instanciación del componente, directamente relacionadas con la plantilla de parametrización antes citada. La variable *Función* hace referencia a la funcionalidad seleccionada por el diseñador, mientras que los tipos de datos que maneja dicha funcionalidad durante la ejecución se determinan con las variables *tipoDatoEntrada* y *tipoDatoSalida*.

Por otra parte, también es necesaria información relativa al desarrollo del componente, independiente de su instanciación. En cada Dockerfile se debe determinar la imagen de la cual parte el contenedor, que será la imagen base desarrollada por el programador de componentes (atributo *imgBase* del elemento *Componente*). Además, es necesario determinar el comando de ejecución del código fuente (atributo *comEjec* del elemento *Componente*). Estos datos se recogen del modelo de componentes propuesto, mientras que las variables de entorno mencionadas son parte de la instanciación del componente.

A modo de ejemplo, la Figura 3 muestra el fichero correspondiente a la imagen personalizada del componente Fog diseñado en la Figura 2.b, con la selección de la función *CalcularDisponibilidad*, lo cual fija los parámetros de tipoDatoEntrada y tipoDatoSalida a *TDisponibilidadDato* y *TDisponibilidad* como las variables asociadas a la instanciación. Además, en color rojo se muestran las variables obtenidas del modelo de componente.

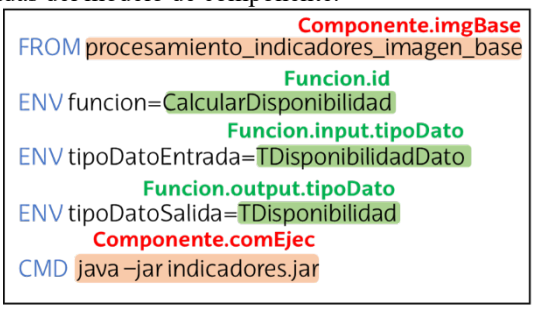


Figura 3 Ejemplo de Dockerfile personalizado resaltando el origen de cada uno de los parámetros del fichero.

5. Generación automática de la librería de componentes Fog

La Librería de Componentes Fog juega un papel importante en el diseño y desarrollo de aplicaciones Fog, no solo en el diseño gráfico de las aplicaciones Fog (los nodos instanciados permiten el diseño gráfico del workflow de microservicios) sino también en la generación automática de los microservicios pertenecientes a aplicaciones Fog (los nodos pueden ser personalizados con funcionalidades a ejecutar una vez instanciados). Por tanto, es la librería la que habilita una separación de intereses entre los programadores de componentes y los diseñadores de aplicaciones.

Para poder llevar a cabo este proceso, se ha identificado la necesidad de introducir dos herramientas clave: una interfaz para habilitar la parametrización del componente Fog, y un programa capaz de ejecutar la generación automática mencionada. Siendo Node-RED la plataforma seleccionada para realizar estas tareas, es importante destacar que es una herramienta de desarrollo construida sobre Node.js y que proporciona un editor basado en navegador web, y, por tanto, los scripts resultantes para los nodos deben estar programados en HTML y JavaScript. Como se indica en la Figura 1, por cada componente Fog, se generan dos ficheros para cubrir la mencionada necesidad:

1. visual.html: Node-RED utiliza una interfaz web, por tanto, este fichero se estructura en HTML. Contiene la información relativa a la representación visual del nodo (vista web del componente) tanto en la librería (color, nombre, salidas o entradas del nodo) como una vez instanciado, y en el apartado de *información* (parte derecha de la Figura 1).

2. funcional.js: Contiene la lógica para desarrollar de forma automatizada los microservicios. Estos ficheros se almacenan en la librería y se ejecutan cuando el diseñador de aplicaciones solicita su despliegue desde el editor. Este fichero es capaz de recoger los valores de las variables definidas en la parte visual, las cuales serán parametrizadas por el usuario durante la instanciación.

Una vez instanciado el componente en el workflow que representa la aplicación, el diseñador de aplicaciones debe seleccionar la funcionalidad a ejecutar por el componente, ayudándose de la información mostrada por el fichero *visual.html*. Con el workflow diseñado, el diseñador de aplicaciones solicita el desarrollo de la aplicación y la herramienta será la encargada de generar automáticamente todos los archivos Dockerfile personalizados (imágenes personalizadas) asociados a cada componente. Para ello, hace uso de los ficheros funcional.js de los nodos de librería instanciados. Concretamente, el fichero *funcional.js* genera las imágenes personalizadas de los componentes Fog con la información extraída del modelo de componente Fog y la parametrización del componente realizada por el diseñador de aplicaciones.

La creación de la librería, es decir la creación de estos ficheros por cada componente Fog, es una tarea repetitiva y propensa a errores. Además, toda la información necesaria, tanto para la parte visual como para la funcional, ha sido previamente recogida por el desarrollador de componentes en el modelo del componente. Por ello, los autores proponen el uso de la tecnología XSLT para implementar transformaciones modelo-a-texto (M2T) que generen los scripts mencionados a partir de la información del modelo de componentes en XML. Por lo tanto, el programador de componentes recibe una hoja de estilo XSLT para cada uno de los archivos que debe generar.

En la Figura 4 se muestra la parte del proceso de desarrollo relacionada con las diferentes funcionalidades de los componentes Fog, definidas por el programador en el modelo en XML del componente de la Figura 2. En la mencionada figura, se muestra el fragmento del código XSLT que genera el elemento HTML para la parametrización de la funcionalidad por parte del diseñador de aplicaciones, siendo el resultado un menú desplegable de opciones web. Por otro lado, gracias a la otra transformación M2T, que genera la parte funcional, el componente es capaz de interpretar la información de selección de ese menú.

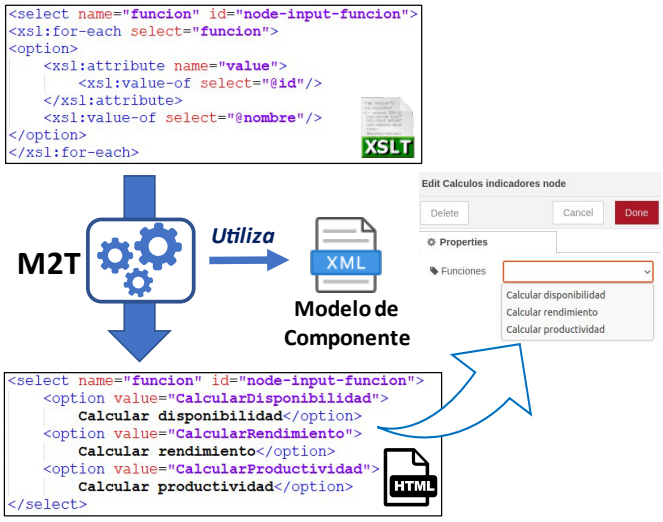


Figura 4 Transformación M2T aplicando el XSLT correspondiente al modelo del componente de la Figura 2.b para generar su fichero *visual.html* y vista del desplegable desde Node-RED.

6. Conclusiones

Teniendo en cuenta que el diseño y desarrollo de los microservicios de aplicaciones Fog se realiza principalmente de forma ad-hoc, en este trabajo se presenta una propuesta metódica para el diseño y desarrollo de componentes Fog que pueden formar parte de diferentes aplicaciones basadas en microservicios.

Con el fin de definir formalmente estos componentes se ha propuesto un meta-modelo de componentes Fog. Así, se separa la definición de los componentes de la lógica de la aplicación, al mismo tiempo que se favorece la reutilización de los componentes. La implementación de este meta-modelo utilizando tecnologías basadas en XML y transformaciones M2T permite validar la coherencia del modelo a la vez que la automatización de la generación de la Librería de Componentes Fog.

El editor propuesto permite utilizar los componentes desarrollados y los prepara para su despliegue como microservicios, aprovechando el potencial gráfico y la capacidad de personalización de la herramienta Node-RED. Todo esto posibilita la separación de intereses entre los programadores de componentes y diseñadores de aplicaciones Fog.

Una limitación del trabajo afecta principalmente a los programadores de componentes, que se ven forzados a parametrizar el código fuente de los componentes. Sin embargo, esta restricción también ofrece ventajas, ya que los programadores son libres de organizar el código como necesiten y, al desarrollar varias funcionalidades dentro del mismo código. se habilita la reutilización de código.

A pesar de poder validar la correcta definición de los componentes, y de automatizar la generación de la librería de componentes Fog, estas tareas tienen que realizarse de forma manual por el desarrollador de componentes. Es por ello que el trabajo futuro estará orientado a extender Node-RED para permitir el modelado gráfico de los componentes, automatizando también la generación de la librería, incorporando las hojas de estilo dentro de Node-RED.

Agradecimientos

Este trabajo ha sido financiado por el proyecto RTI2018-096116-B-I00 financiado por MCIN/ AEI /10.13039/501100011033/, por FEDER Una manera de hacer Europa, financiado por el proyecto PES18/48 financiado por la Universidad del País Vasco (UPV/EHU), y por la beca concedida en el marco de la convocatoria PIF 2022 de la Universidad del País Vasco (UPV/EHU).

Referencias

Ali, S., Kumar, V., Laghari, A., Karim, S., Brohi, A., 2019. Comparison of Fog Computing & Cloud Computing. International Journal of Mathematical Sciences and Computing 31–41.

Belman-Lopez, C.E., Jiménez-García, J.A., Hernández-González, S., 2020. Análisis exhaustivo de los principios de diseño en el contexto de Industria 4.0. Revista Iberoamericana de Automática e Informática industrial 17, 432–447. https://doi.org/10.4995/riai.2020.12579

Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12. Association for Computing Machinery, New York, NY, USA, pp. 13–16. https://doi.org/10.1145/2342509.2342513

Dintén, R., Martínez, P.L., Zorrilla, M., 2021. Arquitectura de referencia para el diseño y desarrollo de aplicaciones para la Industria 4.0. Revista Iberoamericana de Automática e Informática industrial 18, 300–311. https://doi.org/10.4995/riai.2021.14532

Dobaj, J., Iber, J., Krisper, M., Kreiner, C., 2018. A Microservice Architecture for the Industrial Internet-Of-Things, in: Proceedings of the 23rd European Conference on Pattern Languages of Programs. Presented at the EuroPLoP '18: 23rd European Conference on Pattern Languages of Programs, ACM, Irsee Germany, pp. 1–15. https://doi.org/10.1145/3282308.3282320

Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L., Villari, M., 2016. Open Issues in Scheduling Microservices in the Cloud. IEEE Cloud Computing 3, 81–88. https://doi.org/10.1109/MCC.2016.112

Giang, N.K., Lea, R., Leung, V.C.M., 2020. Developing applications in large scale, dynamic fog computing: A case study. Softw: Pract Exper 50, 519–532. https://doi.org/10.1002/spe.2695

Hurtado, E., López, A., Sarachaga, M.I., Armentia, A., Estévez Estévez, E., Marcos, M., 2021. Diseño basado en modelos de aplicaciones FOG como workflow de microservicios, in: XLII JORNADAS DE AUTOMÁTICA : LIBRO DE ACTAS. Servizo de Publicacións da UDC, pp. 695–700. https://doi.org/10.17979/spudc.9788497498043.701

IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing, 2018. . IEEE Std 1934-2018 1–176. https://doi.org/10.1109/IEEESTD.2018.8423800

Industry IoT Consortium, 2022. The Industrial Internet Reference Architecture.

Pop, P., Zarrin, B., Barzegaran, M., Schulte, S., Punnekkat, S., Ruh, J., Steiner, W., 2021. The FORA Fog Computing Platform for Industrial IoT. Information Systems 98, 101727. https://doi.org/10.1016/j.is.2021.101727

Solis, A., Hurtado, J., 2020. Reutilización de software en la robótica industrial: un mapeo sistemático. Revista Iberoamericana de Automática e Informática industrial 17, 354–367. https://doi.org/10.4995/riai.2020.13335

Taneja, M., Jalodia, N., Byabazaire, J., Davy, A., Olariu, C., 2019. SmartHerd management: A microservices-based fog computing–assisted IoT platform towards data-driven smart dairy farming. Software: Practice and Experience 49, 1055–1078. https://doi.org/10.1002/spe.2704

Vayghan, L.A., Saied, M.A., Toeroe, M., Khendek, F., 2021. A Kubernetes controller for managing the availability of elastic microservice based stateful applications. Journal of Systems and Software 175, 110924. https://doi.org/10.1016/j.jss.2021.110924

Zimmermann, O., 2016. Microservices tenets: Agile approach to service development and deployment. Computer Science - Research and Development 32. https://doi.org/10.1007/s00450-016-0337-0