Revista Iberoamericana de Automática e Informática Industrial

Revista Iberoamericana de Automática e Informática Industrial 18 (2021) 300-311

www.revista-riai.org

Arquitectura de referencia para el diseño y desarrollo de aplicaciones para la Industria 4.0

Dintén, R.*, López Martínez, P., Zorrilla M.

Grupo de Ingeniería Software y Tiempo Real, Universidad de Cantabria, 39005, Santander, España..

To cite this article: Dintén, R., López Martínez, P., Zorrilla, M. 2021. Reference architecture for the design and development of applications for Industry 4.0. Revista Iberoamericana de Automática e Informática Industrial 18, 300-311. https://doi.org/10.4995/riai.2021.14532

Resumen

La implementación práctica de la Industria 4.0 requiere la reformulación y coordinación de los procesos industriales. Para ello se requiere disponer de una plataforma digital que integre y facilite la comunicación e interacción entre los elementos implicados en la cadena de valor. Actualmente no existe una arquitectura de referencia (modelo) que ayude a las organizaciones a concebir, diseñar e implantar esta plataforma digital. Este trabajo proporciona ese marco e incluye un metamodelo que recoge la descripción de todos los elementos involucrados en la plataforma digital (datos, recursos, aplicaciones y monitorización), así como la información necesaria para configurar, desplegar y ejecutar aplicaciones en ella. Asimismo, se proporciona una herramienta compatible con el metamodelo que automatiza la generación de archivos de configuración y lanzamiento y su correspondiente transferencia y ejecución en los nodos de la plataforma. Por último, se muestra la flexibilidad, extensibilidad y validez de la arquitectura y artefactos software construidos a través de su aplicación en un caso de estudio.

Palabras clave: Arquitectura centrada en el dato, netamodelo, desarrollo basado en modelos, aplicaciones industriales, industria 4.0.

Reference architecture for the design and development of applications for Industry 4.0

Abstract

The real implementation of Industry 4.0 requires the reformulation and coordination of industrial processes. This requires defining a digital platform that integrates and facilitates communication and interaction between all elements involved in the value chain. There is currently no reference architecture (model) that helps organizations conceive, design and build this digital platform. This work provides a framework and includes a metamodel that describes the main elements involved in the digital platform (data, resources, applications and monitoring), as well as the information needed to configure, deploy and run applications on it. In addition, a tool conformed with this metamodel is provided. This automates the generation of configuration and launch files and their corresponding sending and execution on the platform nodes. Finally, the flexibility, extensibility and validity of the architecture and software artifacts built are shown through its application on a case study.

Keywords: Data-centric architecture, metamodel, model-based development, industrial applications, industry 4.0

1. Introducción

El término Industria 4.0 (I4.0) surgió en 2015 con el fin de revolucionar los procesos industriales y de manufactura europeos para enfrentarse a los grandes desafíos actuales como son la globalización, la competencia, la volatilidad en la demanda o la necesidad de reducir los ciclos de vida durante la innovación, creación y personalización de nuevos productos y/o servicios. Este nuevo modelo industrial se basa en la ubicuidad y conectividad de datos, personas, procesos, servicios y sistemas ciberfísicos que intercambian y explotan la información generada en cada nivel de la arquitectura (ciberfísico, intermediación y niveles de aplicación), que debe ser optimizada de acuerdo con distintos criterios como costes, disponibilidad y eficacia, para conseguir una producción descentralizada y adaptable a cambios en tiempo real

*Autor para correspondencia: ricardo.dinten@unican.es

(Alcácer and Cruz Machado, 2019). Para ello, esta red debe ser dinámica, optimizada en tiempo real y autogestionada (Reference architectural model industrie 4.0, 2018).

Considerando la importancia de adoptar nuevas tecnologías en los métodos de producción industrial, en 2016 Alemania publicó el Reference Architecture Management Industrial 4.0 (RAMI4.0) (Reference architectural model industrie 4.0, 2018) y Estados Unidos, la Industrial Internet Reference Architecture (IIRA) (I.I. Consortium, 2019) con objeto de facilitar su adopción y evolución continua. Ambos proporcionan propuestas arquitecturales para orientar a los fabricantes en la construcción y operación de sistemas industriales pero ninguno de ellos ofrece herramientas para abordar su concepción, diseño y despliegue. Posteriormente, Salkin et al. (2018) señalaron la necesidad de disponer de un conjunto de casos de uso que permitieran visibilizar la aportación de la I4.0 junto con proyectos reales donde se pudiera observar la integración y conexión de las nuevas tecnologías para el desarrollo de productos y procesos inteligentes. Aunque no ofrecen una arquitectura de referencia que se pueda instanciar en un entorno industrial, sí señalan los principios de diseño que debe cumplir, que son: agilidad, interoperabilidad, virtualización, descentralización, gestión de datos en tiempo real, orientación al servicio y procesos de negocio integrados.

Actualmente el Big Data y la computación en la nube (*Cloud Computing*) se consideran habilitadores clave para construir el futuro ecosistema industrial al interconectar mundos cibernéticos y físicos siendo capaces de gestionar la variedad, velocidad, volumen y criticidad de los datos que el entorno industrial genera mediante arquitecturas altamente distribuidas y escalables que pueden ser dimensionadas dinámicamente para procesar cargas de trabajo en tiempo real (Velásquez et al., 2018). De hecho, los datos son el recurso fundamental para promover la I4.0 y conceptos como los “buses de datos" que conectan los entornos de fabricación ya se han identificado como uno de los elementos más importantes (Raptis et al., 2019; Sahal et al., 2020). Es por esto que nuestra propuesta se sustenta en una arquitectura flexible centrada en el dato que facilite la integración de las tecnologías disruptivas con las más tradicionales presentes en la industria. Se ha de señalar que los entornos industriales son complejos ya que disponen de una gran cantidad de recursos hardware heterogéneos y software específico que se ejecuta bajo estrictas restricciones de tiempo real que deben coexistir con estas nuevas tecnologías y que condicionarán su despliegue. Por ejemplo, los datos para la supervisión y el control de un robot deben procesarse por lo general en el nodo más cercano para cumplir con los requisitos de latencia (Cheng et al., 2018) y, por lo tanto, estos datos nunca deben almacenarse o procesarse en la nube.

Para contribuir al desarrollo de la I4.0, se propone:

- La descripción de una arquitectura de referencia para la I4.0, denominada RAI4.0, centrada en el dato y soportada por un conjunto de servicios que proporcionan la gestión y monitorización de la plataforma.
- La definición de un metamodelo que recopila la descripción de todos los elementos involucrados en una plataforma digital compatible con RAI4.0 (datos, recursos, cargas de trabajo y métricas), así como la información necesaria para configurar, desplegar y ejecutar aplicaciones (cargas de trabajo) en él.
- Una herramienta compatible con el metamodelo RAI4.0 que automatiza el despliegue de sistemas RAI4.0 mediante la generación de archivos de configuración y lanzamiento y su correspondiente transferencia y ejecución en los nodos de la plataforma.

El artículo está organizado del siguiente modo: la sección 2 recoge los requisitos de diseño que una plataforma digital I4.0 debe satisfacer a partir del análisis de la literatura y comenta trabajos próximos a nuestra propuesta. La sección 3 describe nuestra arquitectura de referencia RAI4.0. La sección 4 explica detalladamente el metamodelo RAI4.0, tanto a nivel independiente de las tecnologías, como su extensión específica para un conjunto de tecnologías Apache. La sección 5 describe la herramienta para la configuración y despliegue de la plataforma digital, exponiendo las ventajas que proporciona. La sección 6 presenta un caso de estudio para mostrar la validez de nuestra propuesta. Finalmente, la sección 7 resume las principales aportaciones del trabajo y esboza las líneas de trabajo futuro.

2. Arquitecturas Big Data para la I4.0

En esta sección se especifican los requisitos de diseño que una plataforma digital I4.0 debe satisfacer a partir del análisis de la literatura. Así mismo se relacionan y comentan otros trabajos próximos a la arquitectura propuesta en este trabajo.

2.1. Principios de diseño para la I4.0

El término I4.0 surgió en la Feria de Hannover (Alemania) en 2013, también se conoce entre otros como “Fábrica inteligente” o "Internet industrial" (Zhong et al., 2017). De Thoben et al. (2017), se extrae que la I4.0 se define como una nueva estrategia de organización y control de los procesos de producción industrial que utiliza la información que se genera en todo el ciclo de vida de sus productos. El proceso de producción se basa en una infraestructura informática que interconecta todos sus sistemas cibernéticos de producción, logística, gestión y supervisión, para que cada uno de ellos tenga acceso en tiempo real a la información que generan los otros, y a través de asociaciones autónomas descentralizadas y autoregulación, se optimice la producción y se incremente el valor añadido a los productos que se fabrican.

De un análisis de la literatura más actual extrajimos los requisitos esenciales que la plataforma informática que soporte aplicaciones para la I4.0 debe satisfacer y los complementamos con los entresacados de la definición recogida en el párrafo anterior.

En Hermann et al. (2016), se identifican seis principios de diseño que las empresas deben tener en cuenta a la hora de implementar soluciones para la I4.0: interoperabilidad, virtualización, descentralización, capacidad en tiempo real, orientación al servicio y modularidad. Estos principios proceden de los siguientes componentes de la I4.0: sistemas ciberfísicos, Internet de las cosas, Internet de los servicios y Smart Factory. Esta lista es ampliada en Ghobakhloo (2016) añadiendo otros principios no tan vinculados a una solución implementable sino a aspectos generales y muy relevantes a tener en cuenta como son la integración horizontal y vertical,

la personalización del producto o la responsabilidad social corporativa. En Belman-López et al. (2020), realizan un análisis del significado y las implicaciones de la I4.0 y exponen de forma detallada 17 principios de diseño fundamentales obtenidos a través de un estudio de mapeo sistemático: eficiencia, integración, flexibilidad, descentralización, personalización, virtualización, seguridad, solución holística, orientada a servicios, ubicua, colaborativa, modular, robusta, que utiliza información en tiempo real, sirve para la toma de decisiones optimizadas por los datos, equilibra la vida laboral, es autónoma e inteligente.

Estos principios son adoptados por nuestra propuesta arquitectónica. A continuación, se describen de forma más particularizada y bajo una óptica de solución implementable las premisas sobre las que se sustenta:

- Diversidad de la información: El entorno industrial es habitualmente extenso y diverso, por lo que la plataforma computacional debe tener capacidad para gestionar grandes volúmenes de datos, de diversa naturaleza, que se generan a gran velocidad y que además, pueden cambiar dinámicamente en el tiempo.
- Tiempo real y persistencia: Los requisitos de accesibilidad a la información generada en el entorno es muy dispar. Hay información que requiere ser accedida en tiempo real con latencias acotadas, y otra que debe ser persistida para poder ser accedida por aplicaciones que no existían cuando se generó la información.
- Flujos de procesamiento dinámicos: La información que se gestiona se genera dinámicamente en el entorno (máquinas, procesos, sistemas y personas), bien en base a fuentes de datos que existen en él, o bien en base al resultado de cadenas de procesamiento que se generan sobre ellos.
- Interconectividad universal: Las aplicaciones y componentes software deben tener capacidad de acceder a la información que genera cualquier otro componente del entorno. Las limitaciones de acceso no deben ser consecuencia de la arquitectura, sino establecidas por criterios dinámicos de funcionalidad y seguridad.
- Heterogeneidad de los recursos: La plataforma computacional I4.0 es heterogénea. No se construye de forma planificada, sino que resulta de la integración dinámica de recursos requeridos o embebidos en equipos integrados en el entorno, servidores computacionales específicos agregados al entorno como recursos computacionales y recursos virtuales contratados ocasionalmente en la nube.
- Escalabilidad de los recursos: Los recursos computacionales que constituyen la plataforma se reclutan o abandonan en fase operativa en base a la carga de trabajo que se produzca. Ocurrirá cuando el sistema requiera escalar horizontalmente, esto es, modificar su capacidad, o verticalmente, cambiar su funcionalidad.
- Organización autónoma y descentralizada: Para satisfacer el requisito de escalabilidad frente a la carga de trabajo, los componentes deben tener capacidad de establecer y cancelar interacciones con otros componentes de forma autónoma y descentralizada. Los componentes y aplicaciones deben poder organizarse en base a estrategias de búsqueda, negociación y asociación autónoma y descentralizada de grupos funcionales que asuman las cargas y responsabilidades específicas que se produzcan.
- Fiabilidad del sistema: El entorno industrial tiene muchos aspectos críticos desde el punto de vista vital o económico, por lo que debe operar con estrategias que permitan establecer niveles de inmunidad frente a fallos de los elementos hardware y software críticos.
- Seguridad de la información: La plataforma garantiza la seguridad (confidencialidad, integridad y disponibilidad) de la información que proporciona o a la que acceden los recursos, así como de la información de configuración de la propia plataforma (incluida la que define la propia seguridad).

En resumen, la arquitectura propuesta es distribuida y escalable, combina el uso de los recursos disponibles en la computación en la nube (cloud) con los existentes en la infraestructura más cercana a los dispositivos que generan los datos (fog) y la de los propios dispositivos (edge) para cumplir con las restricciones de latencia. Asimismo, facilita la colaboración entre máquinas, procesos, sistemas y personas a través del intercambio de información en tiempo real, lo que permite incorporar componentes que actúen de forma reactiva basados en decisiones dirigidas por los datos presentes en el entorno. Esto conduce a la adopción de una solución técnica centrada en el dato, flexible y dinámica, que permite incorporar o reducir fuentes de datos y/o aplicaciones que procesan estos datos y actúan de forma reactiva y autónoma. Estas aplicaciones se deben construir siguiendo un esquema modular y orientado al servicio (Angulo et al, 2017) para así favorecer el establecimiento de nuevos modelos de negocio colaborativos (Wiesner and Thoben, 2016).

Wingerath et al. (2016) propusieron kappa y lambda como soluciones arquitecturales centradas en el dato y basadas en buses de datos para hacer frente al procesamiento de flujos de datos en tiempo real. Estas han sido utilizadas en el ámbito de la I4.0 por Arantes et al. (2018) para desarrollar una herramienta de análisis de datos para la I4.0; por Zorrilla et al. (2019) para sensibilizar a los usuarios domésticos sobre el consumo de energía sostenible y por Ahmad et al. (2018) para construir un modelo para minimizar la ocurrencia de fallos en las máquinas de una cadena de producción, entre otros.

Asimismo, Raptis et al. (2019) señalaron que los buses de datos (*data buses*) son una de las soluciones técnicas más adecuadas como consecuencia del papel crucial que desempeñan los datos en la integración de los dos mundos, el físico y el cibernético. Finalmente, IIRA incluye el concepto de buses de datos en capas en su arquitectura de referencia para permitir el intercambio de datos entre sistemas IIoT ya que facilitan una comunicación de datos *peer-to-peer*, segura y de baja latencia a través de las capas lógicas de las aplicaciones. En general, esta comunicación se establece mediante un modelo basado en publicación-suscripción. Las aplicaciones simplemente se "suscriben" a los datos que necesitan para operar y que se encuentran en el bus y "publican" la información que producen para que otras hagan a su vez uso de ella.

Por ello, y como se describe en la sección 3, nuestra arquitectura de referencia se concibe bajo un modelo de publicación-subscripción sobre un bus de datos.

2.2 Trabajos relacionados

Desde el inicio de la I4.0, los investigadores y el sector productivo han trabajado con el fin de proponer enfoques metodológicos y herramientas para abordar el diseño y desarrollo de sistemas complejos. El diseño basado en modelos es una disciplina bien establecida para apoyar la concepción, el diseño, la evaluación y el desarrollo de software. En cuanto a su uso y aplicación en la I4.0 o sus tecnologías habilitadoras, encontramos particularmente interesante el estudio realizado por Wortmann et al. (2017) en el que identifican la ingeniería de sistemas basada en modelos (MBSE) como un habilitador clave para el desarrollo de sistemas complejos, caso de la I4.0. Los autores realizan una revisión sistemática de la aplicación de MBSE en la fábrica inteligente y concluyen que la mayoría de los artículos analizados aportan métodos y conceptos para resolver desafíos concretos de la I4.0. Los trabajos se clasifican de acuerdo a la etapa del ciclo de vida que abordan: la fase de especificación de requisitos o la etapa de desarrollo. Por ejemplo, en el primer grupo, Petrasch y Hentschke (2016) definen un lenguaje de modelado de procesos para la I4.0 (I4PML) que extiende el estándar BPMN (Business Process Model and Notation) y describe un método para la especificación de nuevos componentes, por ejemplo, dispositivos IoT o tareas de activación y detección que están presentes en las aplicaciones de I4.0. En el segundo, destacamos el trabajo de Pérez-Palacín et al. (2019) y Guerreiro et al. (2016) que proponen un perfil UML específico para soportar el diseño, la evaluación y el despliegue continuo de aplicaciones de uso intensivo de datos tanto en nubes públicas como privadas, por ser próximo a nuestra propuesta y que podría integrarse en nuestra solución mediante transformaciones de modelos.

En nuestra búsqueda bibliográfica no hemos encontrado un modelo conceptual como el propuesto para describir los elementos que constituyen una solución arquitectural basada en la compartición de datos sobre la que se orquestre un conjunto de servicios globales, heterogéneos y descentralizados. Otras propuestas arquitecturales se basan en el patrón de tres capas (borde, plataforma y negocio) o en el patrón de gestión y conectividad de borde mediado (I. I. Consortium, 2019). En el primero, el nivel de borde recopila datos de los dispositivos ciberfísicos para transferirlos al nivel de plataforma que, a su vez, proporciona funciones de gestión para activos de datos y ofrece servicios relacionados con su análisis, siendo el nivel de negocio el que proporciona interfaces a los usuarios finales e implementa aplicaciones y sistemas de apoyo a la toma de decisiones. Esto es, divide la solución funcionalmente y no existe de facto una compartición e integración global de los datos del entorno. Bajo esta arquitectura se encuentra el modelo conceptual propuesto por Ungurean et al., 2020. El segundo patrón persigue la conectividad local de los dispositivos en el edge y su aislamiento salvo por una puerta de enlace que conecta con una red de área amplia. Su principal beneficio es reducir la complejidad de los sistemas IIoT, de modo que puedan escalar tanto en número de dispositivos administrados como en redes. A diferencia de nuestra propuesta, persigue la computación en el borde y no mixta (Marino et al., 2019).

3. Arquitectura de referencia RAI4.0

La arquitectura RAI4.0 se propone con objeto de delimitar y jerarquizar las estrategias con las que se organiza, cualifica y asocia la información que se gestiona en el entorno, las tareas que la procesan, los recursos de la plataforma que dan soporte a su transferencia, almacenamiento y procesamiento y los agentes de monitorización que permiten configurar y gestionar el sistema en su conjunto.

A fin de satisfacer los requisitos ya expuestos, se propone una arquitectura de referencia basada en tres principios claves:

• DaaS (Los datos como servicio). El entorno industrial se define al más alto nivel en base a la descripción y caracterización de la información que gestiona. Como se muestra en la Figura 1, el sistema digital que da soporte al entorno se construye situando a los datos y su gobierno en el centro de la arquitectura. Los componentes software y los recursos hardware que constituyen la plataforma son sólo medios que se reclutan y escalan a fin de que generen, procesen, den soporte o consuman los datos gestionados.

La gobernanza de los datos es el reconocimiento de que en la I4.0, los datos constituyen un activo estratégico, una ventaja competitiva y que su gestión es esencial para la plataforma informática. Del análisis y la toma de decisiones establecidas en la gobernanza de los datos resultan los requisitos para su gestión y constituyen la base del diseño y de la configuración de la plataforma digital. De la gobernanza emanan los metadatos que en fase productiva están asociados a los datos y que sirven de base para que la plataforma opere de forma autónoma y descentralizada.

Para gestionar la variabilidad de la información del entorno industrial, se organizan los datos por tópicos, que representan flujos de instancias de datos con un mismo tipo de información y que se gestionan con unos mismos criterios de persistencia, durabilidad, disponibilidad, seguridad, integridad, etc. Los tópicos se registran en la plataforma de producción, y los metadatos asociados a ellos constituyen una información global que es consultada por todos los agentes que operan con sus instancias. Su disponibilidad para ser consultada y la capacidad de notificar sus actualizaciones son las que hacen posible la reconfiguración dinámica de los componentes y servicios de forma descentralizada. El conjunto de tópicos registrados en el sistema representa su dominio de operación.

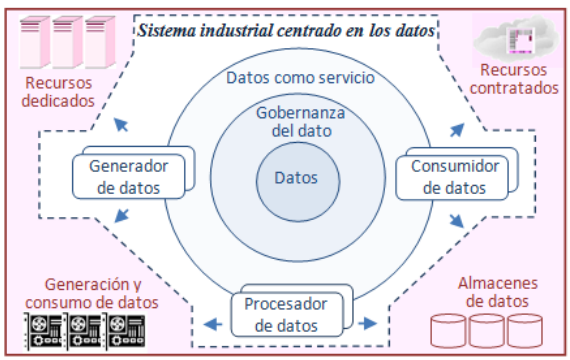

Figura 1: Sistema industrial centrado en los datos.

Las instancias de datos de un tópico se registran en la plataforma como series temporales inmutables, que algún productor escribe una única vez, y son leídas tantas veces como requieran los consumidores que se subscriban a ellas. El ciclo de vida de las instancias está definido en el tópico, y son eliminadas por el entorno en base a criterios de volumen o antigüedad. Los datos se distribuyen en base a particiones, para escalar su almacenamiento por su volumen o facilitar el nivel de concurrencia en su procesamiento y conseguir mayor productividad. El número de particiones, su ubicación y los criterios de distribución de las instancias, se realiza en base a metadatos asignados al tópico y utilizando criterios de distribución equilibrados o por campos claves (*keys*).

Las aplicaciones que incorporan la funcionalidad del entorno se conciben como *workflows* de tareas de procesamiento que operan en base a una estrategia reactiva. Un workflow se ejecuta en base a la ocurrencia de un determinado patrón de instancias de tópicos declarados en el entorno. El conjunto de tareas que constituyen un workflow se organizan en base a una relación de flujo de control con estructura de grafo sin bucles (Figura 2(a)). La ejecución del workflow se inicia en una tarea raíz activada con la ocurrencia del patrón de activación. Las restantes tareas del workflow se activan en base a las dependencias de sincronización (pipeline, branch, fork, merge y join) establecidas en el workflow. Para implementar los mecanismos de transferencia de flujo de control entre tareas, el workflow puede crear y registrar tópicos privados cuyo ciclo de vida coincide con el del workflow que los crea.

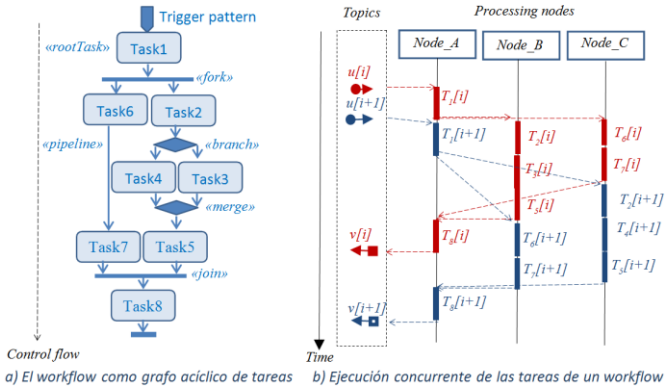

Figura 2: Los workflows como gráficos acíclicos de tareas.

Las tareas de procesamiento se implementan como componentes software desacoplados que se declaran subscriptores de los tópicos que les activan y publicadores de los tópicos que generan. Cada tarea de un workflow se puede instanciar de forma replicada en múltiples nodos de procesamiento, bajo el criterio de que el código de la tarea se transfiere al nodo en que están los datos, y no son los datos lo que se mueven a donde se instancia el código de la tarea. Como muestra la Figura 2(b), las tareas de un workflow que corresponden a la ejecución de una ocurrencia del patrón de disparo se pueden ejecutar distribuidas por diferentes nodos de la plataforma, y así mismo, las ejecuciones correspondientes a la ocurrencia de diferentes patrones de disparo se pueden ejecutar concurrentemente entre sí. El número de réplicas del código de cada tarea de procesamiento que se distribuye en la plataforma, y el nivel de concurrencia con el que se ejecutan las réplicas, es función de la estrategia de particionado asignada al tópico y del mecanismo de planificación distribuida que se asigne.

• PaaS (La plataforma como servicio). Las características claves de las plataformas computacionales de los entornos industriales son su heterogeneidad, su naturaleza distribuida y su necesidad de escalar. Una parte relevante de la capacidad computacional corresponde a sistemas embarcados (dew computing) en los que su conectividad al hardware y/o las condiciones operativas requieren utilizar recursos específicos establecidos por los fabricantes de los equipos. Otra gran parte de la capacidad computacional está compuesta por los recursos destinados específicamente al entorno industrial como CPDs, almacenes de datos, etc. (fog computing) y, por último, cada día es más frecuente contratar recursos públicos (cloud computing) que suplan bajo demanda la capacidad de computación que se necesita. En las plataformas digitales I4.0 no sólo se necesita que los recursos estén interconectados y puedan intercambiar información, sino ofrecer capacidad de reconfiguración dinámica que permita disponer de potencia computacional en base a las necesidades de cada momento.

En nuestra arquitectura de referencia, los recursos computacionales se organizan en nodos de procesamiento físicos o virtuales, que se consideran las unidades de distribución de los datos y de despliegue de las tareas de procesamiento; y, en redes de comunicación, que transfieren información entre ellos. Sin embargo, los recursos que constituyen la plataforma digital del entorno industrial cambian en el tiempo, bien por fallo de los equipos o por el escalado de la plataforma en base a la carga de trabajo, y por ello se definen servicios que gestionan y hacen referencia a agrupaciones de recursos (clusters).

Como se muestra en la Figura 3, los agentes que gestionan los flujos de datos acceden a los recursos computacionales a través de las interfaces de los servicios que definen la plataforma de ejecución como un servicio. Así mismo, la configuración del servicio establece los modos de operación y la forma de uso de los recursos, y a través de ellos el comportamiento de la plataforma.

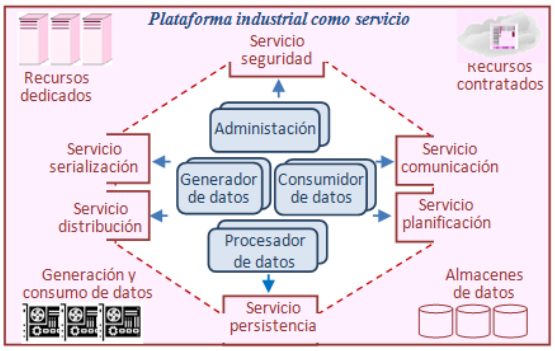

Figura 3: Plataforma computacional como conjunto de servicios.

Todos los servicios que se definen son replicables, redundantes, distribuidos y escalables, esto es, se pueden definir versiones del servicio por ámbitos de tópicos, mantienen un número de réplicas de la información que gestionan para garantizar su robustez frente a la caída de los recursos en que se almacenan, permiten instalar múltiples *brokers* de acceso al servicio y definir por configuración los

recursos que le dan soporte. En la arquitectura de referencia se definen la funcionalidad genérica y la responsabilidad que asumen en el acceso a la plataforma, pero aspectos específicos como la interfaz de acceso y los parámetros de configuración serán dependientes de la implementación del servicio que se utilice. Los servicios que definen la plataforma son:

- Servicio de serialización: Resuelve la serialización de las instancias de los tópicos que se necesita para intercambiar información a través de una red o de un recurso de persistencia, con independencia del lenguaje de programación con el que se han codificado los agentes que acceden a él.
- Servicio de distribución: Resuelve el acceso seguro a una información global compartida en un sistema distribuido. Proporciona dos servicios claves: es el medio de registro y consulta de los metadatos que cualifican los tópicos y de los datos de configuración y coordinación de los servicios distribuidos.
- Servicio de comunicación: Facilita el registro de las instancias de los tópicos de la plataforma y la transferencia de instancias entre agentes del entorno en base al paradigma publicador/subscritor. Este servicio gestiona el ciclo de vida de las instancias de los tópicos, escala la capacidad de gestión del volumen de datos en base a particiones y garantiza la distribución de los datos en base a la declaración de grupos de subscriptores. Todos estos aspectos se configuran en base a metadatos que se asocian a las descripciones de los tópicos.
- Servicio de planificación: Planifica las ejecuciones de las tareas de procesamiento en los nodos en que están almacenados los datos, bien por transferencia e invocación de los códigos de procesamiento, o por activación de los componentes software que están instanciados en los correspondientes nodos.
- Servicio de persistencia: Almacena persistentemente las instancias de datos seleccionadas para que su ciclo de vida supere el nivel de persistencia establecido en el servicio de comunicación general para las instancias del tópico. En una plataforma Big Data hay que balancear las características de consistencia, disponibilidad y particionado de los datos por lo que hay que tener en consideración los diferentes paradigmas NoSQL.
- Servicio de seguridad: Garantiza la autenticación de los agentes y la integridad, confidencialidad y disponibilidad de la información. En una plataforma Big Data las reglas de seguridad deben ser dinámicas y estar orientadas al dato, esto es, los metadatos asociados a cada tópico deben incluir los criterios sobre quién y cómo puede acceder al dato y bajo qué limitaciones.

• MaaS (La monitorización como un servicio). En una plataforma descentralizada se necesita un mecanismo que mantenga actualizada la información sobre el estado global del sistema, la disponibilidad de los datos y los niveles de utilización de los recursos. El servicio de monitorización mantiene una información global del sistema, que facilita la instanciación de los componentes software y de los servicios, así como la notificación de los cambios significativos del estado y de la utilización que los recursos de la plataforma para que se adapten dinámicamente.

A la monitorización de la plataforma contribuyen dos elementos: un servicio de monitorización distribuido que es mantenido por la plataforma y que está constituido por un conjunto de recursos activos instalados en sus nodos y un conjunto de agentes de monitorización que son asociados a los componentes software y a los servicios que dinámicamente se instancien. Los recursos del servicio de monitorización se configuran e instancian cuando los nodos en los que se instalan se incorporan a la plataforma digital. La estrategia de configuración y despliegue de los recursos de monitorización permite balancear el impacto de la monitorización sobre el tráfico de datos en la red frente al requerimiento de memoria para el almacenamiento de la información en los nodos. Los agentes de monitorización se configuran e instancian como parte de la instanciación del componente o del servicio al que estén asociados.

4. Metamodelo RAI4.0

Esta sección detalla el metamodelo RAI4.0, que define los elementos de modelado necesarios para concebir, diseñar y configurar sistemas diseñados de acuerdo con la arquitectura de referencia propuesta.

Cada modelo conforme al metamodelo RAI4.0 está organizado en torno a un elemento raíz, instancia de la clase *ComputationalSystem* (ver Figura 4). Este actúa como contenedor de las tres secciones que componen el modelo:

- Modelo de la plataforma, formado por el conjunto de recursos físicos y virtuales y los servicios de middleware que conforman la plataforma digital, todos ellos modelados a través de clases que heredan de la clase abstracta *PlatformResource*.
- Modelo de workload, formado por el conjunto de data streams (o tópicos) que fluyen en el entorno, modelados a través de la clase *WorkloadStreamData*, junto con los workflows (conjunto de tareas de procesamiento) que producen y/o consumen estos tópicos (clase *Workflow*).
- Modelo de monitorización, esto es, las métricas que deben medirse durante la ejecución del sistema. La clase raíz de la jerarquía de elementos empleados con este propósito se denomina *Metric*.

A continuación, se detallan los elementos de modelado que conforman cada una de estas secciones, y que se definen por herencia de las clases raíces mostradas en la Figura 4.

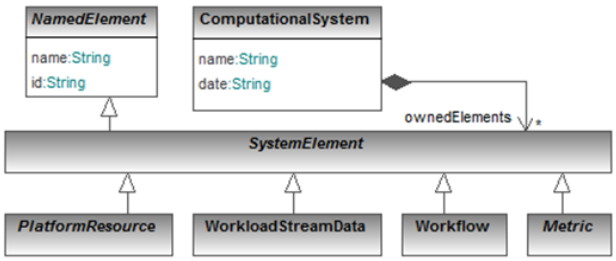

Figura 4. Elementos raíces del metamodelo RAI4.0.

4.1. Modelo de plataforma

La Figura 5 define las principales clases que heredan de *PlatformResource* y que, por tanto, son utilizadas para describir los diferentes tipos de recursos disponibles en la plataforma. *ProcessingNode* representa los nodos de cómputo

que participan en el sistema, tanto los nodos disponibles en el entorno (dew/edge computing) (*PhysicalProcessingNode*) como los recursos virtuales aprovisionados en la nube (*VirtualProcessingNode*). Se identifica cada nodo con su dirección IP y propiedades como el número de procesadores, la memoria disponible o la dirección IP externa en el caso de los nodos virtuales. Los elementos de tipo *PlatformResource* pueden agruparse en elementos *ResourceCluster*.

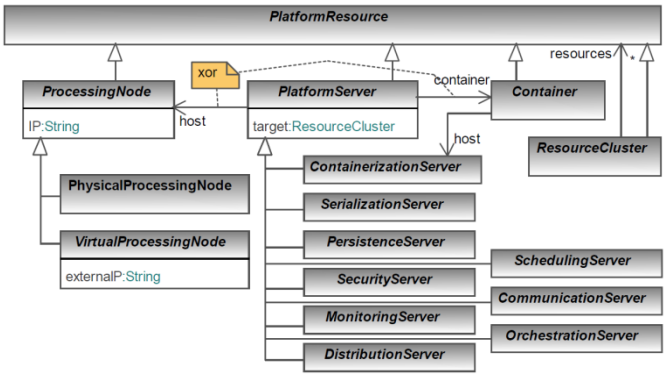

Figura 5. Elementos del modelo de la plataforma.

PlatformServer es una clase abstracta que contiene información genérica sobre los brokers que proporcionan acceso a los servicios distribuidos disponibles en la plataforma. A este nivel de abstracción, solamente están definidos como atributos el nodo donde se aloja el broker y el clúster al que pertenece. Como se muestra en la Figura 5, se define una clase hija por cada tipo de servicio descrito en la sección 3: *SerializationServer*, *SecurityServer*, *DistributionServer*, *CommunicationServer*, *SchedulingServer*, *MonitoringServer* y *PersistenceServer*. Además, se definen dos clases más, para dar soporte al despliegue en contenedores: *ContainerizationServer*, que representa el servicio empleado en la gestión de los contenedores; y *OrchestrationServer*, que representa el servicio encargado de orquestar el clúster de contenedores. El metamodelo se define de manera que se permitan despliegues híbridos, esto es, un *PlatformServer* puede desplegarse directamente sobre un nodo físico o virtual (atributo *host*) o sobre un contenedor (atributo *container*), para lo cual se define la clase abstracta *Container*.

Todas estas clases están definidas como clases abstractas, de manera que deben especializarse para incorporar la semántica y la información específica de las tecnologías escogidas para su implementación. Para la implementación principalmente basada en Apache ("The Apache Sofware Foundation", s.f.) propuesta en el artículo, se definen en el metamodelo un servicio de comunicación basado en Kafka ("Apache Kafka Project", s.f.) (*KafkaServer*) que implementa el bus de datos basado en el modelo publicador-subscriptor; un servicio de distribución basado en Zookeeper ("Apache zookeeper", s.f.; Junqueira & Reed, 2014) (*ZookeeperServer*) responsable de la coordinación del entorno distribuido; un servicio de planificación basado en Apache Storm ("Apache Strom", s.f.) (*StormServer*) responsable de la planificación de las tareas de procesamiento de los datos en tiempo real y un servicio de persistencia basado en el gestor de bases de datos NoSQL Cassandra ("Apache Cassandra", s.f.) (*CassandraServer*). Asimismo, se ha añadido un servicio de monitorización basado en Prometheus ("Prometheus overview", s.f.) (*PrometheusServer*), software que extrae medidas y eventos a través de agentes instalados en los elementos monitorizados y en base a ellos genera métricas o alarmas. Por último, se incorporan las clases necesarias para dar soporte a despliegues basados en contenedores usando tecnología Docker ("Empowering app development for developers", s.f) (*DockerServer*, *DockerContainer* y *SwarmServer*). La tecnología de contenedores es muy utilizada actualmente en la computación en la nube como alternativa al despliegue directo de las tecnologías en nodos físicos, pues permite a los desarrolladores crear entornos predecibles y aislados de otras aplicaciones, facilitando su movimiento de un nodo de computación a otro.

La Figura 6 muestra algunas de estas clases concretas (no se muestran todas por razones de espacio). Cada clase define los atributos necesarios para dar soporte a la información de configuración específica de los servicios que modelan, lo cual requiere una gran cantidad de atributos debido a la complejidad de cada elemento. Sin embargo, la mayor parte de los atributos definen valores por defecto, de manera que el usuario que quiera declarar una instancia de estas clases en un modelo solo tiene que sobrescribir aquellos que necesiten ser ajustados al escenario de despliegue correspondiente.

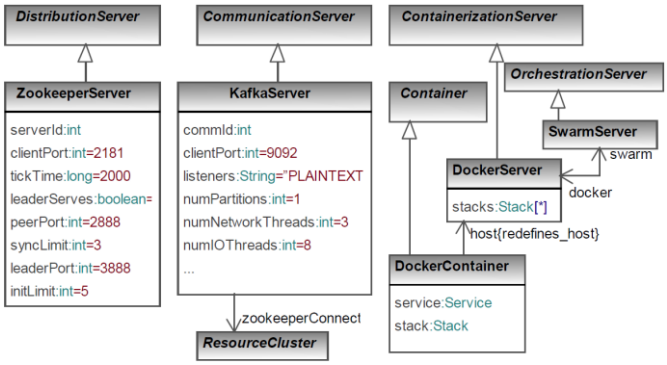

Figura 6. Vista parcial de la extensión de *PlatformServer* para la implementación basada en Apache.

4.2. Modelo de Workload

Como se mencionó anteriormente, dos elementos contribuyen a la definición del workload del sistema: el conjunto de tópicos que fluyen en el entorno y el conjunto de workflows de procesamiento que usan esos tópicos.

La clase raíz que representa los tópicos globales es *WorkloadStreamData*. Esta clase hereda de una clase abstracta denominada *StreamData*, que define atributos comunes a cualquier tópico, como el tamaño (*messageSize)* o el número de copias disponibles de cada instancia del tópico (*numReplication*). La clase *WorkloadStreamData* puede necesitar alguna extensión para incluir características dependientes de las tecnologías, como es el caso de la clase *KafkaWorkloadStramData* que se muestra en la Figura 7 y que incluye aspectos específicos de la forma en que Kafka gestiona los tópicos.

Hay otro tipo de tópicos que pueden ser definidos en el modelo, cuyo propósito es apoyar el flujo de control de las tareas ejecutadas por un mismo workflow. Estos otros tópicos se modelan con la clase *WorkflowStreamData*, que también

hereda de *StreamData* y a su vez está especializada en dos clases: i) *FlowStreamData* que describe el flujo de datos generado por un workflow y es usado para gestionar el flujo de control de las tareas internas por medio de los atributos *source* y *predecessor*; ii) *DerivedStreamData* que representa un flujo de datos virtuales que resultan de aplicar una función de filtro a uno o más flujos de datos.

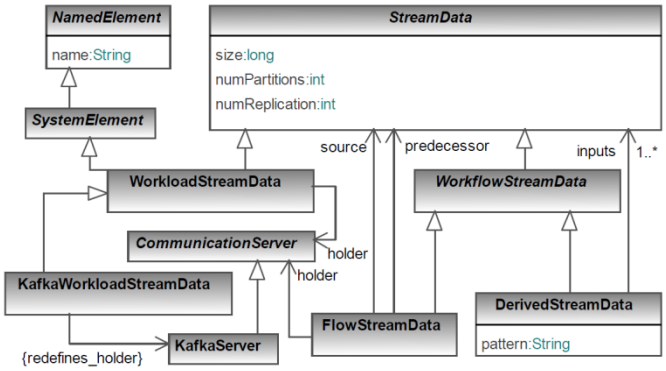

Figura 7. Elementos para el modelado de tópicos.

Por otro lado, las tareas de procesado que explotan los tópicos disponibles en el entorno se organizan en elementos de tipo *Workflow*, cuya estructura principal se muestra en la Figura 8. La ejecución de un workflow es provocada en respuesta a la ocurrencia de un tópico (*rootTask.trigger*), que puede ser de tipo *WorkloadStreamData* o *DerivedStreamData*. Después, un conjunto de tareas de procesado (*ownedTasks*) relacionadas por el flujo de control se ejecutan en el sistema. El control de flujo entre estas tareas se implementa por medio de un conjunto privado de tópicos (*ownedStreamData*) definidos en el workflow. Cuando una tarea finaliza su ejecución, un flujo de datos (*returnedTopic*) es retornado, el cual puede provocar la ejecución de otra(s) tarea(s) del workflow. El tópico que dispara una tarea puede ser un tópico derivado, por tanto, se deben gestionar flujos de control complejos dentro del workflow que pueden estar formados por patrones de branching, forking, joining, etc. El servicio de planificación responsable de asignar y lanzar la ejecución de las tareas se referencia a través del atributo *scheduler* de la clase Workflow. Por último, el elemento *TaskExecutor* representa un artefacto software que encapsula el código de un conjunto de tareas (*executedTasks*) del workflow. Solo se usa en el caso de usar una opción de planificación estática, en la que el despliegue y la instanciación de las tareas debe realizarse explícitamente durante la instanciación del workflow.

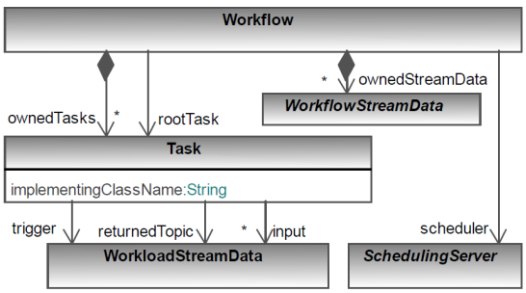

Figura 8. Elementos para el modelado de Workflows.

4.3. Modelo de monitorización

La Figura 9 muestra las clases principales definidas en el metamodelo RAI4.0 para dar soporte a la estrategia de monitorización. Las métricas a medir se modelan a través de clases que heredan de la clase *Metric* y son parte del *ComputationalSystem*. A este nivel de abstracción, los atributos que pueden ser definidos son dos: el elemento al que se le aplica la métrica (*target*) y el agente encargado de proporcionar las medidas (*meter*).

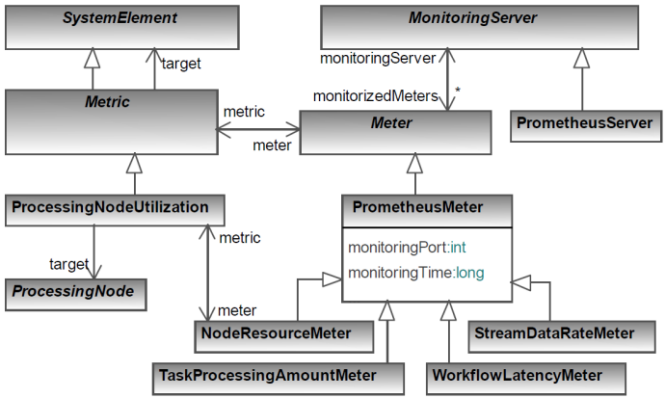

Figura 9. Elementos del modelo de monitorización.

La clase *Meter* describe la información necesaria para la configuración y despliegue de los agentes encargados de obtener las medidas necesarias para evaluar las métricas. A este nivel de abstracción solamente debe definirse la asociación con el correspondiente *MonitoringServer*. Se definirán extensiones de las clases *MonitoringServer* y *Meter* para cada tecnología específica empleada para la tarea de monitorización. En nuestro caso, basado en *Prometheus*, se han añadido las siguientes clases:

- *PrometheusServer*, que modela al propio servidor.
- *PrometheusMeter*, clase raíz para todos los tipos de agentes de medida que se pueden usar. Como Prometheus emplea una estrategia basada en muestreo para la comunicación entre servidor y agentes, los atributos añadidos definen el puerto a través del cual el agente proporciona medidas al servidor (*monitoringPort*) y la frecuencia de muestreo (*monitoringTime*).
- Un conjunto de clases que extienden *PrometheusMeter* con el propósito de modelar los diferentes tipos de agentes de monitorización soportados en la plataforma. Por ejemplo, la clase *NodeResourceMeter* representa un agente de monitorización basado en el *NodeExporter* (Prometheus exporters, s.f.) que permite extraer medidas sobre la utilización y el consumo de memoria de un servidor basado en Linux.

Como consecuencia de que existe una amplia, casi infinita, variedad de métricas que pueden ser potencialmente definidas para un sistema de cómputo, cada nuevo tipo de métrica requiere la definición de una extensión de la clase *Metric*. En la versión actual del metamodelo las métricas definidas están restringidas a aspectos relacionados con el comportamiento temporal y la utilización de recursos. Otras métricas relacionadas con el consumo energético, la fiabilidad, la disponibilidad, etc. podrían ser incluidas si hay servicios de monitorización que proporcionen las medidas necesarias.

Para cada especialización de *Metric*, se debe redefinir sus atributos *target* y *meter* para referenciar a los tipos correctos de acuerdo a la naturaleza de la métrica.

4.4. Modelo de instanciación

Uno de los principios que guía la arquitectura de referencia RAI4.0 es que todos sus elementos, servicios y workflows pueden ser desplegados e instanciados de manera independiente siendo cada componente responsable de adquirir la información requerida para ser instalado en el entorno. Desde ese momento, operan de manera reactiva, esto es, permanecen suspendidos hasta que ocurre el correspondiente evento de activación (la recepción de un mensaje a través de un puerto en el caso de los servicios de la plataforma o una instancia de un tópico en caso de un workflow) y después, ejecutan código ofreciendo como resultado nuevos eventos que transmiten nuevos datos o tópicos del flujo de control. Por tanto, todos los componentes desplegables son distribuidos como artefactos software autocontenidos que encapsulan su código y que están disponibles en los nodos donde pueden ser potencialmente instanciados.

La información necesaria para configurar y lanzar estos artefactos es dependiente de la tecnología y de la naturaleza de cada uno, pero un conjunto de propiedades y métodos comunes se han recogido en el metamodelo bajo la clase *SystemComponent*, de la cual heredan todos aquellos elementos que pueden desplegarse de forma independiente en el sistema: *Workflow* y todas las clases hijas de *StreamData*, *PlatformServer* y *Meter*. Entre sus atributos se encuentran, entre otros, la localización y nombre del artefacto (*artifactLocator* y *artifactName*), los argumentos que puede requerir el tiempo de instanciación (*arguments*) o el directorio donde almacenar los ficheros de configuración que se han de utilizar durante su instanciación (*configDir*).

5. Herramienta de despliegue

El metamodelo propuesto no sólo apoya al proceso de diseño de una aplicación conforme a la arquitectura propuesta, sino que además proporciona toda la información necesaria para poder automatizar su despliegue. A continuación, se resumen los pasos que típicamente se deberían llevar a cabo para desplegar un nuevo componente (servicio o workflow) en un sistema ya establecido:

1. Crear y/o modificar los ficheros de configuración del propio componente así como de otros posibles elementos requeridos.
2. Registrar nuevos tópicos.
3. Definir las métricas a monitorizar una vez lanzado el componente y generar toda la infraestructura necesaria para llevar a cabo dicha monitorización.
4. Lanzar los artefactos software que implementan el componente y los elementos de monitorización.

En todo este proceso hay muchos elementos a configurar y lanzar, cada uno con su propia naturaleza y propiedades, además de las posibles dependencias que pueden surgir entre ellos. Por ello, disponer de una herramienta que automatice el proceso de configuración y despliegue en base a un modelo del sistema es muy deseable y conveniente pues ayuda a reducir errores y a ahorrar tiempo y dinero.

La figura 10 representa los principales elementos involucrados en la herramienta implementada. Esta toma como entrada el modelo del sistema conforme al metamodelo RAI4.0, y lo procesa, generando como resultado el conjunto de ficheros de configuración y scripts necesarios para el lanzamiento del sistema. A continuación, transfiere dichos ficheros a sus correspondientes nodos, usando para ello comandos SCP, y una vez enviados, ejecuta de manera remota, vía SSH, los scripts de lanzamiento de cada componente. Este último paso se realiza de manera ordenada, teniendo en cuenta las posibles dependencias entre componentes del sistema a nivel global. De este modo, la herramienta puede ser ejecutada desde una consola en uno de los nodos que forman parte del sistema o desde cualquier otro nodo conectado por red.

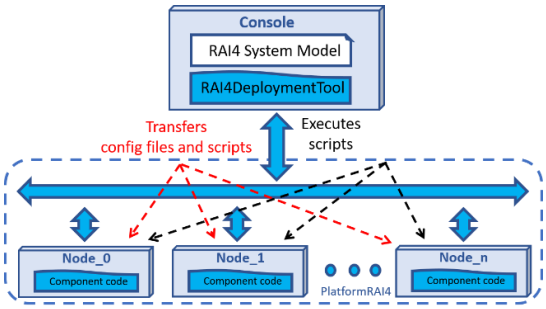

Figura 10. Esquema del proceso de configuración y despliegue.

La herramienta de despliegue se sustenta en un conjunto de atributos y métodos definidos en las clases del metamodelo y que se muestran en la Figura 11. La clase raíz del sistema, *ComputationalSystem* (ver Figura 4), define el método *deployAndLaunch*, cuya invocación lanza el proceso de despliegue y configuración.

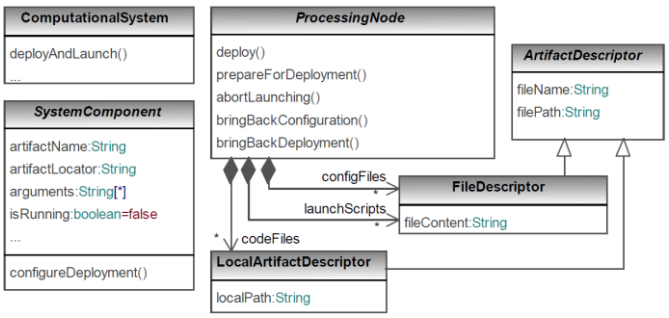

Figura 11. Atributos y métodos introducidos en el metamodelo para dar soporte a la herramienta de despliegue.

La implementación de este método comprende los siguientes pasos:

1. Invocación del método *configureDeployment*, heredado de *SystemComponent*, en cada instancia desplegable. Este método se encarga de generar el contenido de los ficheros de configuración y scripts requeridos para el despliegue de la instancia y añadirlos a los atributos *configFiles* y *launchScripts* de la instancia de *ProcessingNode* que representa el nodo físico en que serán desplegados. A modo de ejemplo, la instanciación en un servidor Kafka de un

tópico denominado *PollutionT*, que será utilizado en el caso de estudio de la sección 6, requeriría la ejecución del siguiente comando en el servidor:

home/apache/servers/kafka/bin/kafka-topics.sh --create –bootstrap-server XXX.XX.XX.46:9092 –partitions 8 –replication-factor 1 –topic PollutionT

El método *configureDeployment* de la correspondiente instancia de *KafkaWorkloadStreamData* que modela dicho tópico generará el citado comando, para lo cual usará la información contenida o accesible desde la propia instancia (ver Figura 7), como el nombre del tópico (*name*), el número de particiones (*numPartitions*) o el puerto de la instancia de servidor Kafka asociado (*holder.clientPort*).

2. Invocación del método *deploy* en cada instancia de *ProcessingNode*. Este método se encarga de transferir, usando SCP, los ficheros almacenados en los atributos *configFiles* y *launchScripts* a los nodos físicos que les corresponden. Si se detecta algún error durante la transferencia, el método lanza la excepción *DeploymentException* y el proceso de despliegue de ficheros se interrumpe, informando al operador y devolviendo el sistema al estado inicial (es decir, eliminando de los nodos todos los ficheros transferidos).
3. Ejecución remota, en base a SSH, de los scripts de lanzamiento transferidos a los nodos en el paso previo, haciendo uso en muchos de los casos de los ficheros de configuración igualmente transferidos anteriormente. Este método puede lanzar la excepción *LaunchingException*, en cuyo caso la herramienta debe retornar también el sistema a su estado inicial e informar al operador. Como se mencionó previamente, estos scripts se ejecutan siguiendo un orden a modo global, teniendo en cuenta las posibles dependencias de elementos que se ejecutan tanto en el mismo nodo como en nodos distintos. Por ejemplo, la ejecución de cualquier servidor Kafka requiere la ejecución previa de su correspondiente servidor Zookeeper.

La implementación actual de la herramienta de despliegue se ha realizado utilizando un entorno Eclipse/EMF como base. El metamodelo RAI4.0 se ha formalizado utilizando Ecore, mientras que la herramienta ha sido escrita en Java. Tanto los metamodelos como la implementación de la herramienta y el modelo correspondiente al caso de estudio se encuentran disponibles en (“RAI4 deployment tool and metamodel”, 2020). La versión actual del metamodelo da soporte a las tecnologías Apache descritas anteriormente. La extensión del mismo a nuevas tecnologías requeriría únicamente la definición de nuevas clases concretas heredadas de *PlaformServer*, *StreamData* y/o *Meter*. Para permitir además la ejecución de la herramienta de despliegue, sería necesaria también la sobreescritura de los métodos *configureDeployment* de dichas clases nuevas. El código de la herramienta no necesitaría ningún tipo de modificación. No obstante, el interés del proyecto es extender de forma gradual el modelo para cubrir las tecnologías más utilizadas con objeto de que el usuario final la utilice como una herramienta CASE.

6. Caso de estudio

Esta sección describe un caso de estudio que pretende mostrar la aplicabilidad de nuestra propuesta. Se trata de una aplicación de tiempo real que analiza y procesa datos sobre polución recogidos por un conjunto de sensores instalados en vehículos de transporte público de una ciudad inteligente (smart city). Los datos son recogidos y publicados a través de Kafka, posteriormente procesados con Storm y persistidos en Cassandra para poder ser consultados y, en un futuro, empleados en la generación de modelos de predicción (Reza Delavar et al., 2019; Díaz et al., 2020). A continuación, se describe el modelo definido para la aplicación de acuerdo con nuestro metamodelo, considerando un escenario de despliegue en el que Kafka y Zookeeper son desplegados en nodos físicos (edge computing), mientras que Storm y Cassandra son desplegados sobre contenedores Docker alojados en la nube.

El primer paso para elaborar el modelo consiste en definir y caracterizar los tópicos que fluyen en el entorno. En este caso, hay un único tópico, denominado *PollutionT*, que recoge los datos de polución leídos de los sensores.

El segundo paso consiste en describir los workflows encargados de producir, consumir y transformar las instancias de los tópicos. La información necesaria para la configuración y despliegue de un workflow incluye tres características: i) la reactividad i.e. los tópicos que desencadenan su ejecución, los que consume y los que genera como resultado de su ejecución; ii) la actividad, i. e., las tareas que son ejecutadas y el control de flujo entre ellas y; iii) la planificación, i. e., el planificador que define la concurrencia y la estrategia de multiplicidad aplicada a la ejecución de las tareas en los recursos y servicios disponibles. En este caso, se ejecuta un único workflow, denominado *PollutionProcessing* y desencadenado por el tópico *PollutionT*, cuyo esquema de comportamiento se muestra en la Figura 12.

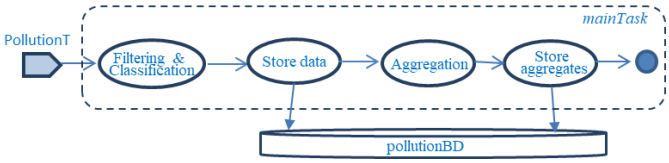

Figura 12. Comportamiento del workflow PollutionT.

El primer paso del workflow tiene como objetivo filtrar los datos para deshacerse de los valores fuera de rango y clasificarlos por región y ventana temporal (10 segundos). Después, el resultado de este paso se almacena en Cassandra. Tras esto, cada parámetro ambiental incluido en el tópico es promediado por región y ventana temporal y, finalmente, estos valores medios se almacenan en Cassandra. En este caso, todas estas funcionalidades se ejecutan dentro de una única tarea, denominada *mainTask* en el modelo.

El tercer paso aborda la definición de los recursos de la plataforma. Los servicios de middleware utilizados son en este caso, Kafka como servicio de comunicación; Zookeeper como servicio de distribución; Storm como servicio de planificación; y, Cassandra como servicio de persistencia. Sus correspondientes instancias deben ser, por tanto, añadidas al

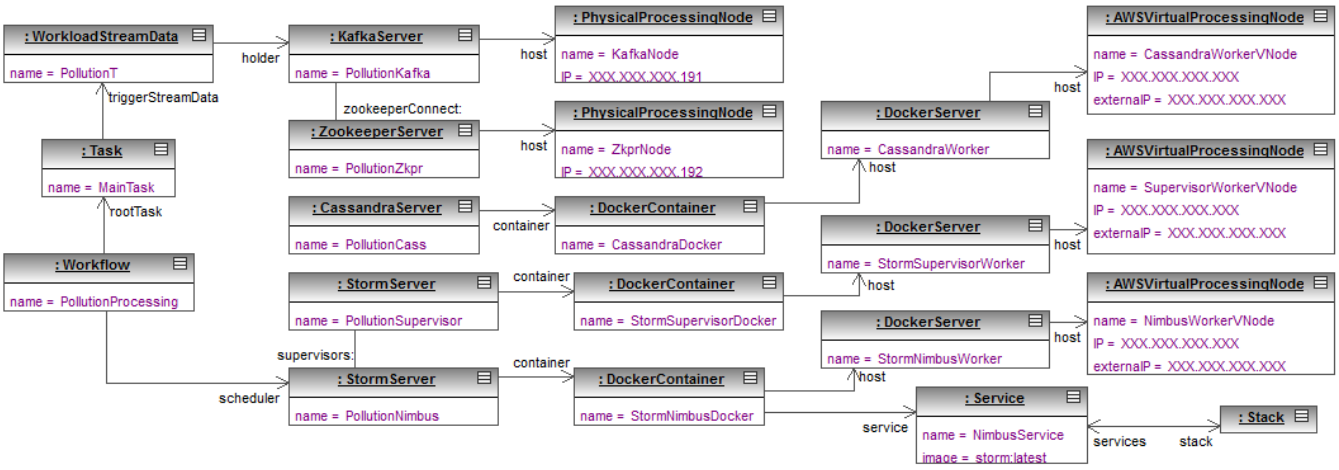

Figura 13. Instancias del modelo del caso de estudio para un entorno de despliegue distribuido y parcialmente virtualizado.

modelo, configuradas y relacionadas con los elementos ya presentes en él, e. g. la instancia de *KafkaServer* se asigna al atributo *holder* del tópico *PollutionT*, ya que Kafka es responsable de su gestión, y la instancia de *StormServer* se asigna al atributo *scheduler* del workflow *PollutionProcessing*, ya que su *mainTask* debe ser desplegada en Storm. La figura 13 representa estas instancias del modelo del caso de estudio y como se relacionan.

El siguiente paso consiste en definir la plataforma de despliegue. Como se ha dicho previamente, en el escenario modelado Kafka y Zookeeper son desplegados en nodos físicos, así que como muestra la Figura 13, sus instancias correspondientes se relacionan, a través de sus correspondientes atributos *host*, con instancias de la clase *PhysicalProcessingNode*. Por otro lado, Storm y Cassandra son desplegados sobre contenedores Docker alojados en la nube, por tanto, sus instancias correspondientes se relacionan con instancias de *DockerContainer* a través del atributo *container*. Cada contenedor referencia, a través del atributo *host*, la instancia de *DockerServer* donde están alojado, cuyo propio atributo host, a su vez, referencia al nodo virtual en el que está desplegado Docker. Asimismo, la instancia *DockerContainer* referencia una instancia de tipo *Service*, que contiene la información necesaria para realizar su despliegue.

Este modelo se usa como entrada de la herramienta que genera el conjunto de ficheros de configuración y scripts necesarios para desplegar y lanzar el sistema en la plataforma. En este caso de estudio, el número de ficheros generados suma un total de siete ficheros de configuración más un fichero stack de Docker y seis scripts, uno por cada una de las siguientes tareas: lanzar Kafka y Zookeeper, crear el tópico en Kafka, desplegar el stack en Docker, crear la base de datos Cassandra y lanzar la topología correspondiente al workflow en Storm.

Con objeto de evidenciar la capacidad de reutilización que proporciona el metamodelo y los beneficios de utilizar la herramienta de despliegue para automatizar el proceso, planteamos dos modificaciones del escenario de despliegue inicial. Si se quisiera desplegar el sistema completo sobre nodos físicos (con un nodo por servicio), sería necesario realizar una ligera modificación del modelo: añadir tres nuevas instancias de *PhysicalProcessingNode* y referenciarlas desde el atributo *host* de las instancias ya existentes de *CassandraServer* y *StormServer*. En este caso se generarían siete ficheros de configuración y ocho scripts (se añaden los de lanzamiento de Cassandra y Storm y desaparecen los relacionados con Docker). Si, sobre este escenario, se quisiera realizar un escalado del servicio Cassandra, pasando a contar con tres instancias del servicio en lugar de una, bastaría con añadir dos nuevas instancias de *CassandraServer* y otras dos de *PhysicalProcessingNode* que representen los nuevos nodos para el despliegue. En este caso, el número de ficheros se aumentaría en seis (dos ficheros de configuración y un script para lanzar Cassandra por cada nuevo nodo). En ambos casos la modificación del modelo y posterior ejecución de la herramienta resultan acciones mucho menos costosas en términos de tiempo y menos propensas a errores que su alternativa basada en reescribir/actualizar los scripts y ficheros de configuración en cada nodo y ejecutarlos manualmente. Los tres modelos se pueden consultar en (“RAI4 deployment tool and metamodel”, 2020).

7. Conclusiones y líneas futuras

La transformación de la industria hacia la I4.0 supone un cambio tecnológico, organizativo y humano revolucionario. En particular, los procesos industriales son muy complejos con requisitos de confidencialidad, integridad y latencias diferentes e integrando tecnologías disruptivas y legadas. La I4.0 aún no está muy instaurada, la decisión de la plataforma digital que soporte las aplicaciones, la monitorización y autogestión de todos los elementos que interaccionan en el entorno no es trivial. Por ello, en este trabajo se ofrecen una arquitectura de referencia y un metamodelo que permite a los analistas y desarrolladores el diseño de un entorno computacional flexible, fácilmente ampliable, escalable y distribuido. Así mismo se ofrece una herramienta que facilita la instalación, configuración y despliegue de aplicaciones virtualizadas o no en nodos presentes tanto en el entorno (dew, edge o fog) como en la nube (cloud).

El trabajo en curso se centra principalmente en el desarrollo de la seguridad (autentificación, confidencialidad, integridad y disponibilidad) de la plataforma en el que se ha

de tener presente las normativas y certificaciones que se exijan a las aplicaciones que se desarrollen sobre ella. Así mismo, se está trabajando en la propuesta de un marco para la gobernanza de datos específica para I4.0 (Yebenes & Zorrilla, 2019) y en la extensión del metamodelo a otras tecnologías de soporte, como por ejemplo Kubernetes (Kubernetes, sf) para la gestion de contenedores o Spark (Apache Spark, sf) para la planificación de tareas, con objeto de ampliar la aplicabilidad de la propuesta.

Agradecimientos

Este trabajo ha sido financiado en parte por el Gobierno de España y los fondos FEDER (AEI/FEDER, UE) en el proyecto TIN2017-86520-C3-3-R (PRECON-I4).

Referencias

Ahmad, S., Badwelan, A., Ghaleb, A. M., Qamhan, A., Sharaf, M. Analyzing critical failures in a production process: is industrial iot the solution?, Wireless Communications and Mobile Computing (2018). doi: 10.1155/2018/6951318.

Alcácer, V., Cruz-Machado, V. Scanning the industry 4.0: A literature review on technologies for manufacturing systems, Engineering Science and Technology, an International Journal 22 (3) (2019) 899 – 919. doi:https://doi.org/10.1016/j.jestch.2019.01.006.

Angulo, P., Guzmán, C. C., Jiménez, G., Romero, D. A service-oriented architecture and its ict-infrastructure to support eco-efficiency performance monitoring in manufacturing enterprises, International Journal of Computer Integrated Manufacturing 30 (1) (2017) 202–214. doi:10.1080/0951192X.2016.1145810.

Arantes, M., Bonnard, R., Mattei, A. P., Saqui-Sannes, P. de. General architecture for data analysis in industry 4.0 using sysml and model based system engineering, in: 2018 Annual IEEE International Systems Conference, SysCon 2018, Vancouver, BC, Canada, April 23-26, 2018, 2018, pp.1–6. doi:10.1109/SYSCON.2018.8369574.

Apache Cassandra., http://cassandra.apache.org/ (accessed 30 April 2019).

Apache Kafka project: A distributed streaming platform, http://kafka.apache.org/ (accessed 30 April 2019).

The Apache Software Foundation, http://www.apache.org/ (accessed 30 April 2019).

Apache Spark: A fast and general engine for large-scale data processing, http://spark.apache.org/ (accessed 30 Dec 2019).

Apache Storm: A fast and general engine for large-scale data processing, https://storm.apache.org/ (accessed 30 Dec 2019)

Apache Zookeeper., https://zookeeper.apache.org/ (accessed 30 April 2019).

Belman-López, C., Jiménez-García, J., & 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(4), 432-447. doi:https://doi.org/10.4995/riai.2020.12579

Chen, Y., Feng, Q., Shi, W. An industrial robot system based on edge computing: An early experience, in: USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), USENIX Association, Boston, MA, 2018.

Díaz, G., Macià, H., Valero, V., Boubeta-Puig, J., Cuartero, F. An Intelligent Transportation System to control air pollution and road traffic in cities integrating CEP and Colored Petri Nets, Neural Computing and Applications 32(2): 405-426 (2020).

Empowering app development for developers | Docker, https://www.docker.com/ (accessed 28 September 2020)

Ghobakhloo, M. The future of manufacturing industry: a strategic roadmap toward industry 4.0, Journal of Manufacturing Technology Management 29 (2018) 910–936.

Guerriero, M., Tajfar, S., Tamburri, D. A., Di Nitto, E. Towards a model-driven design tool for big data architectures, in: Proceedings of the 2Nd International Workshop on BIG Data Software Engineering, BIGDSE '16, ACM, New York, NY, USA, 2016, pp. 37–43. doi:10.1145/2896825.2896835.

Hermann, M., Pentek, T., Otto, B. Design principles for industrie 4.0 scenarios, 49th Hawaii International Conference on System Sciences (HICSS), 2016, pp. 3928–3937.

I. I. Consortium, Industrial internet reference architecture v1.9, http://www.iiconsortium.org/IIRA.htm, accessed 30 April 2019 (2019).

Junqueira. F., Reed B., ZooKeeper: Distributed process Coordination, O,Reilly, 2014.

Kubernetes., https://kubernetes.io/ (accessed 18 Decemeber 2020)

Marino F., Seitanidis I., Dao P., Bocchino S., Castoldi P., Salvadori C. IoT enabling PI: towards hyperconnected and interoperable smart containers, 6th International Physical Internet Conference, 2019, pp. 349-362.

Pérez-Palacín, D., Merseguer, J., Requeno, J. I., Guerriero, M., Di Nitto, E., Tamburri, D. A. A uml profile for the design, quality assessment and deployment of data-intensive applications, Software and Systems Modeling 18 (6) (2019) 3577–3614. doi:10.1007/s10270-019-00730-3.

Petrasch, R., Hentschke, R. Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method, 13th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2016, pp. 1–5. doi:10.1109/JCSSE.2016.7748885.

RAMI 4.0, Reference architectural model industrie 4.0, https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.html, accessed 30 Dec 2019 (2018).

Prometheus exporters, https://github.com/prometheus/node_exporter (accessed 30 April 2019).

Prometheus overview, https://prometheus.io/docs/introduction/overview/ (accessed 30 April 2019).

RAI4 deployment tool and metamodel, https://github.com/istr-uc/RAI4DeploymentTool (accessed 20 July 2020).

Raptis, T. P., Passarella, A., Conti, M. Data management in industry 4.0: State of the art and open challenges, IEEE Access 7 (2019) 97052–97093. doi:10.1109/ACCESS.2019.2929296.

Reza Delavar, M., Gholami, A., Reza Shiran, G., Rashidi, Y., Reza Nakhaeizadeh, G., Freda K., Hatefi Afshar, S. A Novel Method for Improving Air Pollution Prediction Based on Machine Learning Approaches: A Case Study Applied to the Capital City of Tehran, ISPRS Int. J. Geo-Information 8(2): 99, 2019.

Sahal, R., Breslin, J. G., Ali, M. I. Big data and stream processing platforms for industry 4.0 requirements mapping for a predictive maintenance use case, Journal of Manufacturing Systems 54 (2020) 138 – 151. doi:https://doi.org/10.1016/j.jmsy.2019.11.004.

Salkin, C., Oner, M., Ustundag, A., Cevikcan, E. A Conceptual Framework for Industry 4.0, Springer International Publishing, Cham, 2018, pp. 3–23. doi:10.1007/978-3-319-57870-5_1.

Thoben, K.-D., Wiesner, S., Wuest, T. "industrie 4.0" and smart manufacturing – a review of research issues and application examples, International Journal of Automation Technology 11 (1) (2017) 4–16. doi:10.20965/ijat.2017.p0004.

Ungurean, I., Gaitan, N.C. A Software Architecture for the Industrial Internet of Things—A Conceptual Model, *Sensors* 2020, *20*, 5603.

Velásquez, N., Estevez, E., Pesado, P. Cloud computing, big data and the industry 4.0 reference architectures, Journal of Computer Science and Technology 18 (03) (2018) e29. doi:10.24215/16666038.18.e29.

Wiesner, S., Thoben, K.-D. Requirements for models, methods and tools supporting servitisation of products in manufacturing service ecosystems, International Journal of Computer Integrated Manufacturing (2016) 1–11doi:10.1080/0951192X.2015.1130243.

Wingerath, W., Gessert, F., Friedrich, S., Ritter, N. Real-time stream processing for big data, Information Technology 4 (58) (2016) 186–194.

Wortmann, A., Combemale, B., Barais, O. A systematic mapping study on modeling for industry 4.0, ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2017, pp. 281–291. doi:10.1109/MODELS.2017.14.

Yebenes, J., Zorrilla, M. Towards a data governance framework for third generation platforms, Procedia Computer Science The 2nd International Conference on Emerging Data and Industry 4.0 (EDI40), 2019.

Zhong, R. Y., Xu, X., Klotz, E., Newman, S. T. Intelligent manufacturing in the context of industry 4.0: A review, Engineering 3 (5) (2017) 616 – 630. doi:https://doi.org/10.1016/J.ENG.2017.05.015.

Zorrilla, M. E., Ibrain, Á. Bernard, an energy intelligent system for raising residential users awareness, Computers & Industrial Engineering 135 (2019) 492–499. doi:10.1016/j.cie.2019.06.040.