Extensión del metamodelo RAI4.0 para el diseño, configuración y despliegue de aplicaciones intensivas de datos en entornos mixtos

Ricardo Dintén[1], Marta Zorrilla Pantaleón[1], y Patricia López Martínez[1]

[1]Facultad de Ciencias, Universidad de Cantabria, Santander, España
{ricardo.dinten,marta.zorrilla,patricia.lopez}@unican.es

Abstract. La cuarta revolución industrial aboga por la reformulación de los procesos industriales para lograr la digitalización de extremo a extremo (proveedor-cliente) del sector industrial. Como es bien sabido, el entorno industrial es muy complejo, en él los sistemas heredados deben interoperar e integrarse con sensores y dispositivos modernos. Por ello, las arquitecturas basadas en la compartición de datos y la implementación de servicios se consideran una de las soluciones tecnológicas más flexibles y adecuadas para lograr gradualmente la deseada integración horizontal y vertical de la cadena de valor. Con el objetivo de disminuir la complejidad que entraña el diseño y despliegue de aplicaciones intensivas de datos, nuestro grupo de investigación ha desarrollado RAI 4.0 [1], un metamodelo para caracterizar los diferentes elementos a configurar en una DIA y facilitar su despliegue haciendo uso de una herramienta creada al efecto, que genera ficheros de configuración, y despliegue y los envía a los nodos correspondientes para su ejecución. Este artículo señala las ventajas de las arquitecturas distribuidas y centradas en datos para hacer frente al reto de la integración y la interoperabilidad en sistemas de datos y presenta la extensión del metamodelo RAI4.0 que ahora permite especificar también aplicaciones desplegadas en contenedores sobre servicios de orquestación en entornos mixtos cloud, fog, edge, de manera que cada servicio puede ser alojado de acuerdo a sus requerimientos funcionales y no funcionales, principalmente de tiempo real, seguridad y dependencias de hardware. Para validar la flexibilidad del metamodelo, se han planteado y modelado diferentes escenarios de despliegue para un caso de uso de una ciudad inteligente.

Keywords: Diseño basado en modelos, plataforma digital, contenerización, servicios, IoT.

1 Introducción

Las industrias están siguiendo un proceso de transformación digital promovido

por la Industria 4.0 (I4.0). La I4.0busca un cambio en la organización y los procesos de manufactura hacia un entorno más flexible, controlado e inteligente dirigido por los datos. Según [2], para que la I4.0 se haga realidad, es esencial implementar la integración horizontal entre corporaciones, la integración extremo a extremo de la cadena de valor de los productos y la integración vertical dentro de las fábricas. Esto lleva a la necesidad de diseñar y planificar de forma estratégica el despliegue de una plataforma digital que permita una implementación gradual de servicios y procesos totalmente digitalizados.

Para alcanzar ese objetivo, Alemania publicó RAMI [3], un modelo arquitectural de referencia unificado que ofrece una visión conjunta de los estándares I4.0. Por otro lado, el Industrial Internet Consortium (IIRA) [4] habilita a los arquitectos de sistemas IIoT a diseñar sus sistemas usando un vocabulario común, un framework de arquitectura y una arquitectura de referencia. Al mismo tiempo, otras organizaciones han propuesto guías de referencia como NIST o MIIT en China. En particular, nuestra propuesta se basa en el patrón de arquitectura de bus de datos propuesto en IIRA como elemento central de la arquitectura, que es modelado junto con otros componentes necesarios para construir DIA que cumplan los principales requisitos de los escenarios I4.0 (IoT, sistemas ciber físicos y fabricas inteligentes). Estos requisitos, de acuerdo con [5], son interoperabilidad, virtualización, descentralización, tiempo real, orientación a servicios y modularidad, que fueron posteriormente extendidos y refinados en [6]. Además, es de suma importancia que la arquitectura permita la integración de sistemas heredados presentes en el entorno y habilite la interoperabilidad con los dispositivos de última generación de manera transparente. Esto se consigue haciendo uso de un modelo de comunicación centrado en el dato basado en el esquema publicador-suscriptor, en el que las aplicaciones simplemente se suscriben a la información que necesitan y publican la información que producen.

El diseño, desarrollo y despliegue de DIA es una tarea compleja debido principalmente al gran número de tecnologías que tienen que ser orquestadas, configuradas y desplegadas para cumplir su función [7]. Por otro lado, en la mayor parte de los casos estas aplicaciones deben desplegarse en entornos mixtos cloud/edge con el objetivo de cumplir requisitos de latencia, gestionar un gran volumen de datos y tener una gran capacidad de cómputo. Esto significa que las decisiones de diseño, sobre todo las de despliegue, son más complejas cada vez debido a la heterogeneidad de los dispositivos, la diversidad de los entornos de ejecución, las restricciones temporales y la seguridad y requisitos legales.

Por tanto, es altamente recomendable trabajar con herramientas software que faciliten la concepción y configuración de las aplicaciones, así como el despliegue. Como señalan Ataei et al. [8], las arquitecturas de referencia son artefactos efectivos para afrontar el desarrollo de sistemas big data complejos que aportan a los ingenieros software conocimientos sobre patrones de diseño para solucionar problemas que ocurren frecuentemente. En [9], describimos RAI4.0, una arquitectura de referencia centrada en el dato que permite especificar todos los elementos involucrados en una DIA de la I4.0: fuentes de datos, recursos físicos y virtuales, flujos de trabajo y elementos de monitorización que permitan conocer las necesidades de escalar y reconfigurar los demás sistemas. La arquitectura propuesta se

define por medio de un metamodelo que contiene información detallada sobre cómo implementar todos esos elementos. En este artículo, extendemos el metamodelo para añadir la posibilidad de desplegar aplicaciones en entornos mixtos donde parte de los nodos se encuentran en el edge y parte en cloud, así como de ejecutarlas encapsuladas en contenedores. La contenerización es esencial en este tipo de aplicaciones, ya que es la forma más eficiente de desplegar soluciones software [10] al mismo tiempo que se facilita su portabilidad y reutilización en diferentes infraestructuras.

Para mostrar la utilidad y validez de nuestro metamodelo para diseñar aplicaciones intensivas en datos, así como para desplegarlas en entornos mixtos mediante nuestra herramienta de despliegue, describimos un caso de uso orientado al preprocesamiento y almacenamiento de datos de contaminación procedentes de sensores medioambientales instalados en una ciudad inteligente.

El artículo se organiza de la siguiente manera. La sección 2 describe de forma breve la arquitectura RAI4.0. La sección 3 describe los cambios realizados en el metamodelo RAI4.0 para soportar el diseño y configuración de aplicaciones desplegadas sobre contendores. La sección 4 hace una breve introducción al caso de uso empleado para validar la flexibilidad del metamodelo. Por último, la sección 5 recoge las conclusiones del trabajo y el trabajo en curso.

2 Arquitectura RAI4.0

RAI4.0 es una arquitectura general de referencia para la definición y despliegue de aplicaciones industriales. Se apoya en una arquitectura centrada en los datos y en un conjunto de servicios middleware que proporcionan su gestión y la monitorización de la plataforma. RAI4.0 se formaliza en un metamodelo que describe todos los elementos que intervienen en una plataforma digital intensiva en datos (datos, recursos, aplicaciones y métricas de monitorización), así como la información necesaria para configurar, desplegar y ejecutar aplicaciones sobre ella.

Como se muestra en la Fig. 1, cada modelo que cumple con el metamodelo RAI4 está organizado en torno a un elemento raíz, instancia de la clase *ComputationalSystem*. Éste actúa como contenedor de las tres secciones que componen el modelo de un sistema RAI4:

- El modelo de plataforma, formado por el conjunto de recursos físicos y virtuales y los servicios middleware que componen la plataforma digital. Todos ellos modelados a través de clases que heredan de la clase abstracta *PlatformResource*.
- El modelo de carga de trabajo, constituido por el conjunto de flujos de datos (*topics*, en jerga Kafka) que fluyen en el entorno, modelado a través de la clase *WorkloadStreamData* junto con los flujos de trabajo (conjunto de tareas de procesamiento) que producen y/o consumen estos tópicos (clase *Workflow*).
- El modelo de monitorización, es decir, el conjunto de métricas que deben medirse en tiempo de ejecución.

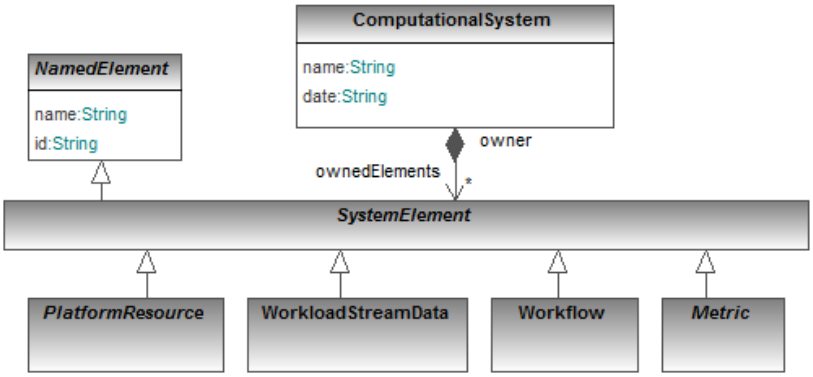

Fig. 1. Elementos raíz del metamodelo RAI4.0

El modelo de plataforma es la sección más afectada por la ampliación propuesta en este trabajo, por lo que a continuación se ofrece una explicación más detallada del mismo. La Fig. 2 muestra las clases que heredan de *PlatformResource* y que, por tanto, se utilizan para describir los distintos tipos de recursos disponibles en la plataforma. *ProcessingNode* representa los nodos de procesamiento que participan en el sistema, ya sean nodos físicos a nivel *dew* o *edge computing* [11], modelados como instancias de una clase *PhysicalProcessingNode*, o recursos virtuales contratados en la nube, representados como instancias de la clase *VirtualProcessingNode*. Las instancias *PlatformResource* pueden agruparse en elementos *ResourceCluster*, si es necesario.

PlatformServer es una clase abstracta raíz que recoge información genérica sobre la configuración, despliegue e identificación de los brokers que dan acceso a los servicios distribuidos disponibles en la plataforma. Como se muestra en la Fig. 2 esta clase está, a su vez, especializada en siete clases abstractas, una por cada tipo de servicio incluido en la arquitectura: *SerializationServer*, *CommunicationServer*, *DistributionServer*, *SchedulingServer*, *SecurityServer*, *MonitoringServer* y *PersistenceServer*.

Como se ha dicho antes, estas clases se describen a nivel abstracto, por lo que deben especializarse cuando se elige un determinado conjunto de tecnologías específicas para su implementación.

En nuestro caso, se utiliza una plataforma basada casi por completo en tecnologías Apache, por lo que se añadieron al metamodelo las siguientes clases: un servicio de comunicación basado en Kafka (*KafkaServer*), un servidor de distribución basado en Zookeeper (*ZookeeperServer*), un servidor de planificación Apache Storm (*StormServer*) y un servidor de persistencia Cassandra (*CassandraServer*). Asimismo, se ha incluido un servidor de monitorización basado en Prometheus (*PrometheusServer*). Dos de ellos se muestran en detalle en la Fig. 2, recogiendo cada clase los atributos con la información específica necesaria para la configuración y despliegue de los servicios correspondientes.

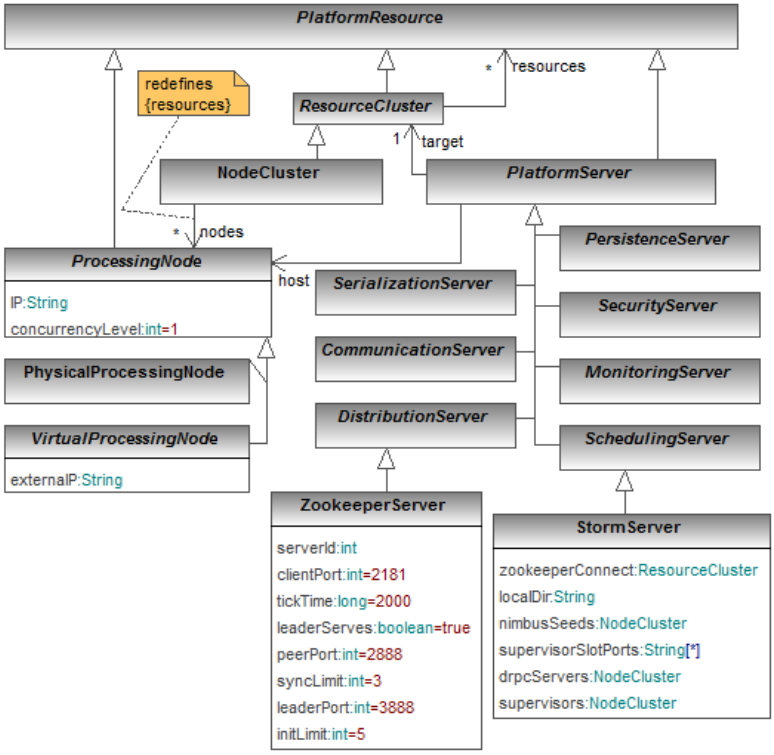

Fig. 2. Recursos de plataforma y servicios middleware (Modelo de Plataforma)

Aunque no está directamente afectado por la extensión, es importante explicar cómo se describe el modelo de carga de trabajo en el metamodelo, especialmente para una mejor comprensión del caso de uso expuesto en la sección 4. El resto del metamodelo, por ejemplo, la sección de monitorización, se omite por razones de espacio, pero puede consultarse en [9]. Dos elementos contribuyen a la definición de la carga de trabajo de un sistema: el conjunto de tópicos que fluye a través del entorno y el conjunto de flujos de trabajo de procesamiento que produce/consume/transforma estos tópicos. Estos flujos de trabajo constituyen las aplicaciones que proporcionan la funcionalidad en el entorno. Se conciben como conjuntos de tareas de procesamiento que operan siguiendo una estrategia reactiva, en respuesta a la aparición de un patrón específico de instancias procedentes de uno o varios tópicos, que desencadena la ejecución de la tarea raíz del flujo de trabajo. El resto de tareas que constituyen el flujo de trabajo se ejecutan siguiendo un grafo acíclico dirigido.

El metamodelo describe los tópicos como instancias de la clase *WorkloadStreamData*, mientras que las tareas de procesamiento que explotan los tópicos se organizan en elementos *Workflow*, cuya estructura principal se muestra en la Fig.3. La ejecución de un flujo de trabajo se desencadena en respuesta a la apari-

ción de un tópico (*rootTask.trigger*). A continuación, se ejecuta un conjunto de tareas de procesamiento (*ownedTasks*) relacionadas por el flujo de control. La tarea señalada por el atributo *rootTask* representa la que inicia la ejecución del flujo de trabajo. El flujo de control entre las tareas se implementa mediante un conjunto de tópicos privados (*ownedStreamData*) definidos por el flujo de trabajo, que son instancias de *WorkflowStreamData*, un tipo especial de tópico. Cuando una tarea termina su ejecución, se devuelve un flujo de datos (*returnedTopic*) que, a su vez, puede desencadenar la ejecución de otra(s) tarea(s) del flujo de trabajo. El servicio de planificación responsable de la asignación de recursos y de lanzar la ejecución de las tareas se referencia a través del atributo *scheduler*.

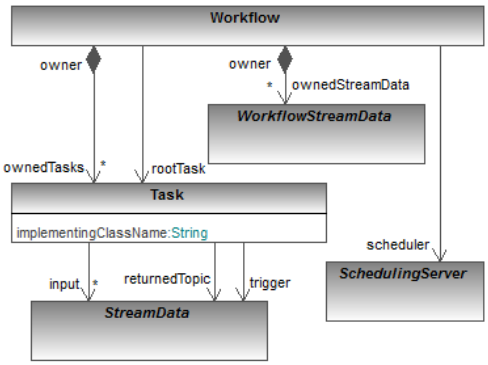

Fig. 3. Modelo de cargas de trabajo

3 Extensión del metamodelo RAI4.0

Para permitir el diseño y despliegue de aplicaciones en contenedores, hemos refactorizado el metamodelo RAI4.0. Estos cambios están dirigidos a permitir que los servidores de plataforma puedan desplegarse tanto en nodos locales o en la nube directamente, como mediante servicios de contenerización, como Kubernetes o Docker Swarm.

En primer lugar, se ha redefinido la clase *PlatformServer*, que pasa a llamarse *PlatformService* (ver la Fig.4) y ya no representa a un único broker con un *ProcessingNode* como host, sino a todo el servicio con un *NodeCluster* como host. Esto significa que, en la versión anterior, un cluster Kafka de tres brokers se modelaba con tres *KafkaServers* alojados cada uno en un *ProcessingNode* y ahora se modela mediante una única instancia de *KafkaService*, alojado en un *NodeCluster* formado por tres nodos. Este cambio reduce el número de elementos necesarios para crear un modelo y, por tanto, facilita la construcción, comprensión y refactorización de los modelos.

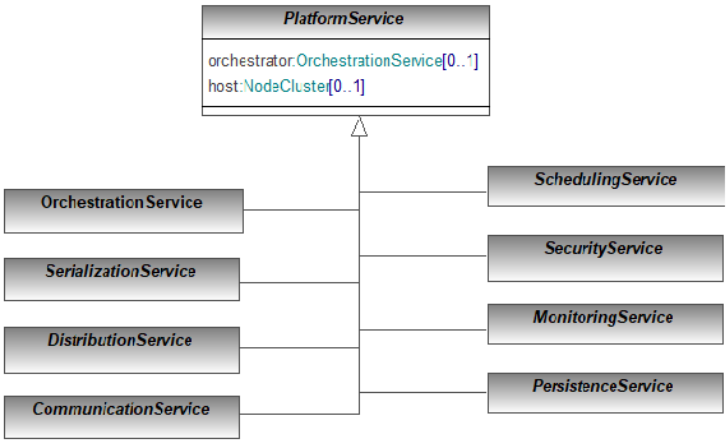

Fig. 4. Extensión de RAI4.0 para soportar aplicaciones desplegadas sobre contenedores.

El segundo cambio aplicado al metamodelo es la inclusión de un nuevo hijo de *PlatformService* (antes *PlatformServer*) llamado *OrchestrationService* (ver Fig. 5). Éste representa el software encargado de gestionar y orquestar los contenedores sobre los que se despliegan los demás *PlatformService*. Dos clases concretas heredan de *OrchestrationService*, modelando las dos principales tecnologías de contenerización actuales: *KubernetesService* y *SwarmService*.

La tercera modificación realizada es la incorporación del parámetro *orchestrator* a la clase *PlatformService* (ver Fig. 5) para que los servicios puedan desplegarse tanto en un *NodeCluster* (vía *host*) como en un *OrchestrationService* (vía *orchestrator*).

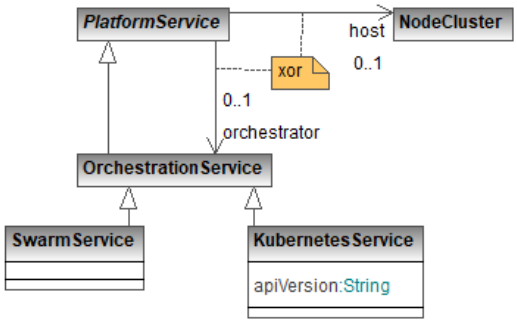

Fig. 5. Modificación del atributo host y nuevo atributo *orchestrator*.

Por último, se han realizado algunos cambios en las clases involucradas en la configuración del despliegue: la clase SystemComponent se ha renombrado como

DeployableComponent y se ha añadido una nueva clase DeploymentConfiguration. Como se puede ver en la Fig. 6, *DeploymentConfiguration* está extendida por *PlatformServicesDeploymentConf* que, a su vez, está extendida por dos clases: *OrchestrationServiceDeployment* y *NodeClusterDeployment* para representar la configuración específica de cada modo de despliegue. *NodeClusterDeployment* mantiene los antiguos parámetros de configuración de la clase *SystemComponent* y *OrchestrationServicesDeployment* añade un conjunto de parámetros y clases necesarios para desplegar flujos de trabajo sobre tecnologías de contenerización. Este último recoge los principales metadatos independientes de la tecnología de una aplicación desplegada sobre contenedores, que son:

- *Image*: representa el registro, repositorio y versión de la imagen que se empleará para generar el contenedor.
- *Replica*: número de réplicas que deben ejecutarse en un momento dado.
- *Command*: comando de línea de comandos que se debe ejecutar al iniciar el contenedor.
- *Volume*: modela la configuración de persistencia de las aplicaciones desplegadas.
- *Port*: modela el mapeo de puertos entre los contentedores y los nodos en los que son desplegados para hacer las aplicaciones accesibles desde el exterior.
- *DeploymentConstraints*: modela restricciones para asegurar que las aplicaciones se despliegan exclusivamente en un conjunto de nodos definido por el usuario (vía *labels*) o en nodos que cumplen con unos requisitos mínimos de recursos disponibles (vía *resources*).
- *Env*: recoge las variables de entorno que deben existir en el contenedor.
- *RestartPolicy*: modela el comportamiento de la aplicación en caso de fallo o reinicio.
- *ImagePullPolicy*: indica al servicio de orquestación cuándo debe descargar la imagen del repositorio.

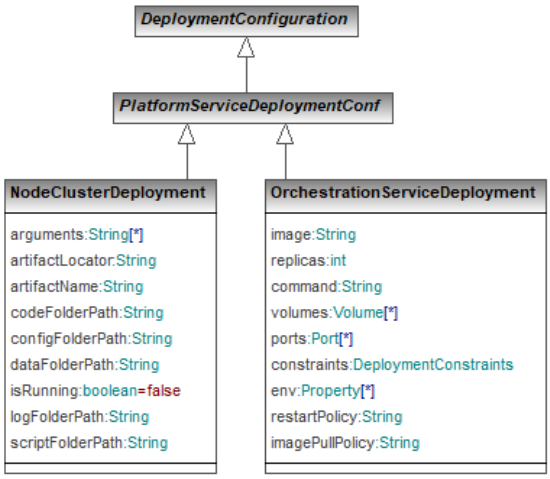

Fig. 6. Clases para la configuración en las diferentes modalidades de despliegue

Estos cambios hacen que el metamodelo generalice mejor y abarque una mayor variedad de configuraciones de despliegue. Sin embargo, también introducen algunas restricciones que deben cumplirse para que el modelo sea correcto, por ejemplo, la relación XOR existente entre los atributos *host* y *orchestrator* de un *PlatformService*. Éstas se han incluido como restricciones OCL en el metamodelo para que, una vez creado el modelo, el usuario pueda validarlas y asegurarse de que el modelo está correctamente formado.

4 Caso de estudio

Para demostrar la validez del metamodelo, se ha diseñado y desplegado un caso de estudio pseudoreal a partir de los datos ofrecidos por un repositorio opendata que surgió como parte de un proyecto Smart city. Los datos contienen los niveles de contaminación de la ciudad de Santander. Estos datos son medidos por unos sensores instalados en vehículos de transporte público.

Partiendo de estos datos se ha desarrollado una aplicación que reporta los datos de contaminación de cada distrito de la ciudad en tiempo real, así como agregados en diferentes ventanas temporales. Como se representa en la figura 7, los datos son leídos de un único tópico Kafka y, una vez procesados, son almacenados en un base de datos Cassandra. De la aplicación se han desarrollado dos versiones, una implementada en Apache Spark y en Apache Storm.

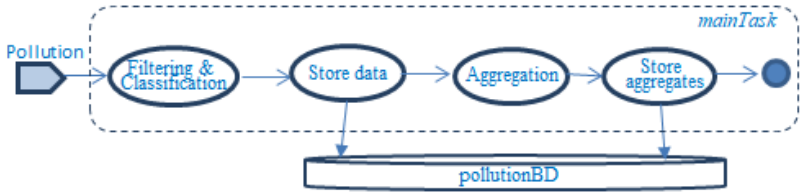

Fig. 7. Workflow de procesamiento de datos de contaminación

Para demostrar la capacidad de reutilización de nuestro metamodelo se han modelado seis escenarios de despliegue de la aplicación:

1. Despliegue en local con todos los servicios desplegados sobre un mismo nodo.
2. Despliegue en cluster local (*on premise*) con todos los servicios distribuidos en un cluster ubicado en el laboratorio ISTR de la Facultad de Ciencias de la UC.
3. Despliegue en cluster cloud con todos los servicios distribuidos en un cluster aprovisionado en *AWS EC2*.

4. Despliegue en cluster local usando docker Swarm como tecnología de orquestación de contenedores.
5. Despliegue en cluster on premise usando Kubernetes como tecnología de orquestación de contenedores.
6. Despliegue en entorno mixto donde algunos servicios corren sobre nodos *on premise*, otros sobre nodos cloud y otros virtualizados sobre contenedores Docker orquestados usando Kubernetes.

En aras de la comprensión, a continuación, se explica cómo, partiendo del escenario de despliegue número 2 mostrado en la figura 8, se puede pasar a los escenarios de despliegue número 3 y 5.

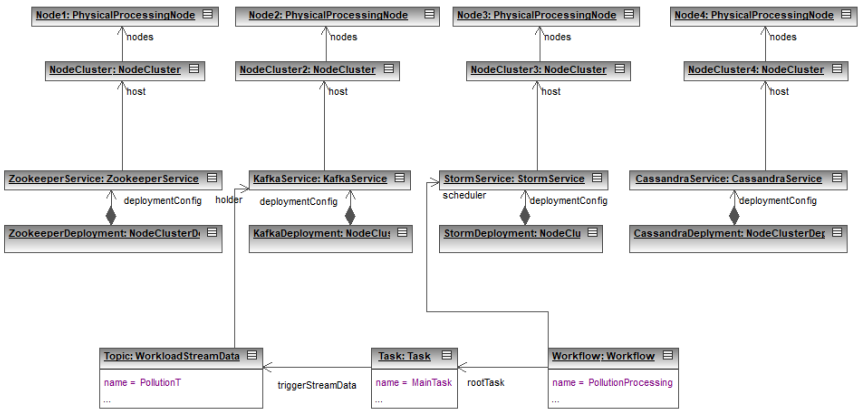

Fig. 8. Modelo parcial del escenario de despliegue en cluster on premise

En el escenario de la figura 8 tenemos los *PlatformServer ZookeeperServer*, *KafkaServer*, *StormServer* y *CassandraServer* cada uno con un *NodeCluster* diferente como host. Cada uno de los *NodeCluster* está compuesto de un único *PhysicalProcessingNode* que representa un nodo del laboratorio de ISTR. Asimismo, cada PlatformServer tiene asociada una instancia del tipo *NodeClusterDeployment* que contiene la información necesaria para el despliegue de la aplicación sobre un cluster de nodos ya sean físicos o virtuales. El *Workflow PollutionProcessing* está compuesto de una única *Task* de nombre *MainTask* que se ejecuta como respuesta a los eventos del tópico *PollutionT*. El tópico se modela como una instancia de *WorkloadStreamData* con el atributo *holder* apuntando a la instancia de *KafkaService.*

Para pasar del escenario de despliegue anterior al escenario de despliegue número 3 en el que los servicios se despliegan sobre sobre nodos virtuales alojados en AWS solamente es necesario sustituir las instancias de *PhysicalProcessingNode* por instancias de *AWSVirtualProcessingNode*. Para obtener el escenario de despliegue número 5 es necesario hacer más cambios. Como se puede ver en la figura 9, en primer lugar, hay que introducir la información del servicio de orquestación mediante una instancia de *KubernetesService* con su atributo host apuntando a un *NodeCluster* formado por instancias de *ProcessingNode* que contengan la

información de los nodos donde se encuentra desplegado Kubernetes. En segundo lugar, se debe eliminar el atributo *host* del resto de *PlatformService* y asignar valor al atributo *orchestrator* de manera que quede apuntando a la instancia de *KubernetesService*. Por último, las instancias de *NodeClusterDeployment* de cada *PlatformService* deben sustituirse por instancias de *OrchestrationServiceDeployment*. En el modelo de carga de trabajo, esto es, el *Workflow* no es necesario realizar ningún cambio, pues lo único que cambia es el despliegue de los *PlatformService* de los que hace uso

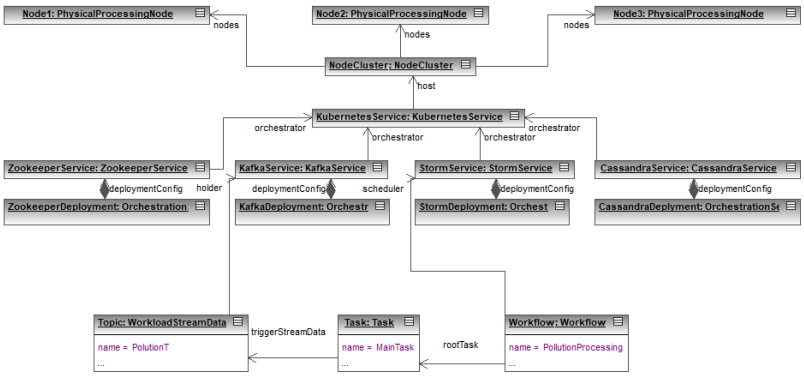

Fig. 9. Modelo parcial del escenario de despliegue en cluster on premise Kubernetes como tecnología de orquestación de contenedores.

Con estos dos ejemplos se muestra la flexibilidad que ofrece el metamodelo para reconfigurar el despliegue de una aplicación para adaptarlo a los cambios o nuevas necesidades del sistema.

5 Conclusiones

Hace unos años, la computación en nube era la solución para la gestión y computación masiva de datos. Sin embargo, las restricciones de tiempo real y los requisitos específicos de los sistemas ciberfísicos presentes en los entornos industriales han llevado a diseñar y desplegar aplicaciones intensivas en datos que deben reconfigurarse fácilmente para que los servicios y recursos utilizados se desplieguen de manera que cumplan los requisitos funcionales y no funcionales de cada aplicación. Pero, lamentablemente, todavía hay pocas herramientas conceptuales que ayuden a los equipos de TI a diseñar y desplegar estas aplicaciones. Para solventar esta carencia, este trabajo presenta un metamodelo dirigido a aplicaciones de Industria 4.0, aunque también puede utilizarse en los ámbitos de la Salud, la Logística o el IoT, entre otros.

En particular, este artículo describe la extensión del metamodelo RAI4.0 para especificar el despliegue de aplicaciones parcial o totalmente contenerizadas en entornos mixtos. El metamodelo propuesto se define en dos niveles: un modelo independiente de la plataforma, extensible y reutilizable y otro especializado que recoge las tecnologías de vanguardia más populares. Esto permite a los profesionales diseñar sus aplicaciones teniendo en cuenta todos los elementos esenciales en aplicaciones intensivas en datos y, por otro, configurar diferentes despliegues y medir su rendimiento mediante las métricas de monitorización que se implementen.

El trabajo en curso se centra en extender la herramienta de despliegue para soportar los nuevos elementos del metamodelo que facilitan el despliegue de aplicaciones contenerizadas en entornos mixtos, lo que conlleva más versatilidad y agilidad para su implantación (y/o reconfiguración). Además, se planea aplicar el metamodelo a nuevos casos de estudio.

Acknowledgements. Este proyecto ha sido financiado en parte por MCIN/AEI/10.13039/501100011033/FEDER "Una manera de hacer Europa" PID2021-124502OB-C42 (PRESECREL) y el programa predoctoral "Concepción Arenal" financiado por la Universidad de Cantabria y el Gobierno de Cantabria (BOC 18-10-2021).

Referencias

1. P. L. Martínez, R. Dintén, J. M. Drake, M. E. Zorrilla, A big data-centric architecture metamodel for industry 4.0, Future Gener. Comput. Syst. 125 (2021) 263 284. doi: 10.1016/j.future.2021.06.020. URL: https://doi.org/10.1016/j.future.2021.06. 020
2. S. Wang, J. Wan, D. Li, C. Zhang, Implementing smart factory of industrie 4.0: An outlook, International Journal of Distributed Sensor Networks 12 (1) (2016) 3159805. doi:10.1155/2016/3159805. URL https://doi.org/10.1155/2016/3159805
3. R. Heidel, M. Hoffmeister, M. Hankel, U. Döbrich, The Reference Architecture Model RAMI 4.0 and the Industrie 4.0 component, VDE Verlag, 2019. URL https://www.vde-verlag.de/buecher/624990/the-reference-architecture-model-rami-4-0-and-the-industrie-4-0-component.html
4. I. I. Consortium, Industrial internet reference architecture v1.9 (2019).
5. M. Hermann, T. Pentek, B. Otto, Design principles for industrie 4.0 scenarios, in: 2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, pp. 3928‑3937. doi:10.1109/HICSS.2016.488.
6. C. E. Belman-Lopez, J. A. Jiménez-García, S. Hernández-González, Análisis exhaustivo de los principios de diseño en el contexto de industria 4.0, Revista Iberoamericana de Automática e Informática industrial 17 (4) (2020) 432‑447. doi: 10.4995/riai.2020.12579. URL: https://polipapers.upv.es/index.php/RIAI/article/view/12579
7. C. Pahl, A. Brogi, J. Soldani, P. Jamshidi, Cloud container technologies: A state-of-the-art review, IEEE Transactions on Cloud Computing 7 (3) (2019) 677‑692.
8. P. Ataei, A. Litchfield, The state of big data reference architectures: A systematic

literature review, IEEE Access 10 (2022) 113789 – 113807, cited by: 0; All Open Access, Gold Open Access, Green Open Access. doi:10.1109/ACCESS.2022.3217557. URL:https://www.scopus.com/inward/record.uri?eid=2-s2.0-85141501375&doi=10.1109%2fACCESS.2022.3217557&partnerID=40&md5=bfb2aa73fc3e59d67e1afe472b713719

9. U. ISTR group, RAI4 deployment tool and metamodel (2020). URL https://github.com/istr-uc/RAI4DeploymentTool
10. R. Qasha, Z. Wen, J. Ca la, P. Watson, Sharing and performance optimization of reproducible workflows in the cloud, Future Generation Computer Systems 98 (2019) 487 – 502. doi: https://doi.org/10.1016/j.future.2019.03.045. URL http://www.sciencedirect.com/science/article/pii/S0167739X18314377
11. M. Gusev, What makes dew computing more than edge computing for internet of things, in: 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), 2021, pp. 1795–1800. doi: 10.1109/COMPSAC51774.2021.00269.