HACIA UNA PLATAFORMA GENÉRICA DE GESTIÓN DE APLICACIONES DINÁMICAMENTE RECONFIGURABLES

U. Gangoiti*, A. Armentia*, E. Estévez**, O. Casquero*, M. Marcos*
*Dept. Ingeniería de Sistemas y Automática, UPV/EHU, España
e-mail: {unai.gangoiti, aintzane.armentia, oskar.casquero, marga.marcos}@ehu.eus
**Dept. Ingeniería Electrónica y Automática EPS de Jaén, España
e-mail: eestevez@ujaen.es

Resumen

El llamado Internet de las Cosas (IoT, Internet of Things) ha permitido la interconexión universal de dispositivos de muy diferente naturaleza, dando lugar a aplicaciones que pueden medir su entorno, procesar e interpretar la información adquirida y, en caso necesario, reaccionar frente a cambios producidos en él. Existen plataformas de gestión que gestionan la ejecución de este tipo de aplicaciones. Sin embargo, suelen ser soluciones ad-hoc que resuelven una determinada problemática de un dominio concreto. Este trabajo se centra en una propuesta basada en modelos para la definición de las características propias del dominio, independizando así las aplicaciones de la plataforma de gestión MAS-RECON, basada en MAS y propuesta en trabajos previos. La propuesta ha sido validada en el ámbito de los sistemas de control de proceso descritos a partir de su diagrama de proceso e instrumentos (P&ID).

Palabras clave: Internet of Things; Sistemas Multi-Agente; Modelado de dominio.

1 INTRODUCCIÓN

Los actuales avances en las tecnologías de la información y comunicación han permitido la expansión del llamado Internet de las Cosas (*IoT, Internet of Things*) [4], y su variante industrial IIoT (*Industrial IoT*) [6]: un paradigma basado en la interconexión universal de "objetos" o "cosas" dotadas de identidad digital, y con la habilidad de medir, interpretar/procesar y reaccionar a su entorno, pudiendo colaborar entre ellos para lograr objetivos comunes. Estas aplicaciones se encuentran en ámbitos de aplicación muy diversos, que abarcan desde la monitorización remota para la detección de catástrofes naturales o supervisión médica, hasta ciudades inteligentes o entornos de fabricación flexibles. Sin embargo, a pesar de pertenecer a dominios tan dispares, las aplicaciones IoT presentan ciertas similitudes. Por un lado, sus principales objetivos pueden agruparse en cuatro áreas: monitorización, control, optimización y autonomía. Y por otro lado, demandan una serie de requisitos similares en cuanto a interoperabilidad (gestión de diferentes tipos de dispositivos, redes, protocolos, etc.), disponibilidad (minimización de situaciones de interrupción del servicio), escalabilidad (añadir o eliminar recursos y/o servicios) y adaptabilidad (evolución con cambios en su contexto) [5].

Para la gestión de estas aplicaciones en tiempo de ejecución se emplean plataformas que proporcionan mecanismos para, como mínimo, controlar su arranque, parada y ejecución, comunicando los diferentes módulos que las componen. Además, para poder hacer frente a su dinamismo, algunas plataformas también incorporan medios de gestión de ciertas calidades de servicio (QoS), por ejemplo, ofreciendo mecanismos para modificar la configuración de las aplicaciones sin detener su ejecución [1]. Así, algunas permiten que la propia aplicación evolucione cuando su contexto cambia [8]. En otras, el foco se encuentra en asegurar su disponibilidad, bien incorporando mecanismos en el código de la aplicación [9] o mediante medios de recuperación propios de la plataforma transparentes a las aplicaciones [12], [3]. También existen trabajos que permiten gestionar QoS específicas de un campo de aplicación, controlando los recursos demandados y los disponibles, o incluso gestionando cambios en la QoS de las aplicaciones para optimizar el uso de recursos [13].

No obstante, estas plataformas se suelen implementar como soluciones ad-hoc para resolver una determinada problemática, muchas veces aplicable a un dominio de aplicación concreto. Es por ello por lo que, o no logran hacer frente a todas las demandas de las aplicaciones objeto de estudio o sólo lo hacen para un determinado ámbito. En trabajos previos de los autores se propuso MAS-RECON, un middleware basado en agentes para la gestión de la ejecución de aplicaciones distribuidas sensibles al contexto [2], [7]. Se trata de aplicaciones que supervisan su entorno, pudiendo detectar cambios en él y reaccionar rápida y adecuadamente, por lo que pueden englobarse dentro del marco de las

aplicaciones IoT. Dicha plataforma también dispone de mecanismos de flexibilidad para posibilitar la adaptabilidad de las aplicaciones y asegurar su disponibilidad.

Sin embargo, MAS-RECON, aún siendo genérica para un amplio abanico de dominios, tuvo que re-diseñarse hasta cierto punto para aplicarla al ámbito de sistemas de fabricación flexible [11], ya que la estructura de aplicación es diferente a la gestionada por MAS-RECON. Es por ello que, teniendo en cuenta que existen requisitos comunes a todos los campos, principalmente los relativos a la gestión de aplicaciones (registrar, arrancar parar…), aún es necesario avanzar hacia el desarrollo de una plataforma de gestión genérica y que al mismo tiempo pueda ser personalizable. La aportación concreta de este trabajo se centra precisamente en una propuesta que permita particularizar MAS-RECON con las características propias de las aplicaciones de dominio, pudiendo hacer la gestión de la ejecución independiente del mismo. La propuesta está basada en modelos que permiten definir las particularidades de cada dominio. Los mecanismos de gestión deberán ser adaptados consecuentemente, haciendo la plataforma extensible para poder cubrir requisitos específicos.

El resto de este trabajo está estructurado de la siguiente manera: la sección 2 se centra en la definición del dominio, de forma que permita describir la estructura y características del tipo de aplicaciones de interés. En la sección 3 se describen los procesos genéricos de registro en la plataforma, tanto de aplicaciones como de recursos del sistema, que se realiza a través de una interfaz genérica independiente del dominio. La sección 4 está dedicada a la evaluación de la solución propuesta en el ámbito de los sistemas de control de proceso, mediante un caso de estudio simple que realiza el sistema de control de temperatura y nivel de un tanque. Por último, en la sección 5 se resumen las conclusiones más importantes y el trabajo futuro.

2 DEFINICIÓN DE MODELO DE SISTEMA GENÉRICO

Como se ha comentado, el objetivo final de la plataforma es ofrecer servicios de gestión de aplicaciones para arrancarlas, pararlas y gestionar su ejecución, asegurando su disponibilidad y cualquier otra calidad de servicio. Para ello, la plataforma dispondrá de "módulos de sistema" que gestionan entidades de dominio clasificadas en dos grupos: 1) relacionadas con la funcionalidad de las aplicaciones; 2) que ofrecen recursos para la correcta ejecución de las aplicaciones.

Para poder llevar a cabo estas tareas, la plataforma necesita conocer el estado actual del sistema, que guarda en el llamado Modelo de Sistema y que comprende información de dos tipos. Por un lado, se debe recopilar información relativa al diseño de las aplicaciones y sus requisitos de ejecución, durante un proceso de registro previo a su puesta en marcha. Dichos datos de registro es necesaria para poder tomar decisiones en tiempo de ejecución. Por otro lado, también se debe disponer de información referente al estado tanto de las aplicaciones en ejecución como de los recursos disponibles. Por lo tanto, se trata de un elemento vivo y cambiante ya que contiene información sobre el estado de todas las entidades que conforman el sistema en un determinado instante: aplicaciones registradas y los agentes que las ejecutan tras ser arrancadas; recursos disponibles y los agentes que los supervisan.

Precisamente, uno de los principales problemas que hacen que MAS-RECON no sea genérico reside en la estructura de su Modelo de Sistema. En efecto, se trata de una estructura estática totalmente ligada a una arquitectura de aplicación concreta: escenarios formados por aplicaciones funcionalmente relacionadas, y aplicaciones como conjunto de componentes que cooperan intercambiando datos. Sin embargo, si bien es una arquitectura aplicable en muchos ámbitos, tales como eSalud, prevención de catástrofes o video-vigilancia, resulta una propuesta restrictiva en otros muchos casos, como el ya comentado de sistemas de fabricación flexible.

Es por ello que el primer paso para lograr una plataforma genérica consiste en la propuesta de una metodología para la definición de las entidades del dominio en base a modelos que permiten personalizar dicha plataforma. En este contexto, para la implementación del modelado de dominio se ha optado por tecnologías XML (*eXtensible Markup Language*). Así, el Modelo de Sistema se guarda en un fichero con formato XML (*SystemModel.xml*), que sigue una gramática que define las entidades del dominio, descrita mediante un fichero XML *Schema* (*SystemModel.xsd*). Un XML Schema (XSD) [14], junto con el conjunto de reglas *schematron* apropiado [10], permite definir lenguajes de dominio incluyendo: el léxico, el estilo arquitectónico y las reglas de composición que los modelos deben cumplir.

La estructura básica del Modelo de Sistema se ilustra en la Figura 1. El sistema está formado por las aplicaciones de dominio y los recursos de los que hacen uso. De este modo, la particularización a un dominio concreto consiste en definir un meta-modelo de dominio que extienda este XML Schema básico, identificándose: 1) las diferentes entidades que componen el sistema, tanto de aplicación como de

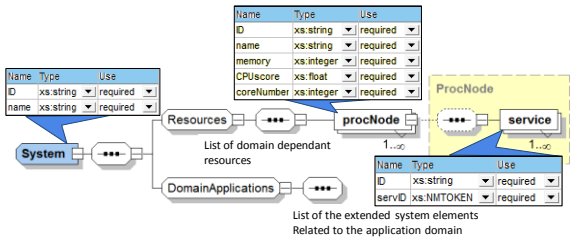

Figura 1: Estructura básica del Modelo de Sistema (*SystemModel.xsd*).

recurso; 2) las propiedades que las caracterizan; y 3) las relaciones que existen entre ellas, tanto de jerarquía como de dependencia.

Con el objetivo de que esta extensión a dominio se pueda gestionar de forma genérica por parte de la plataforma, se establecen una serie de directrices en su definición. Más concretamente, cada entidad del dominio se representa por un elemento de esquema, estando agrupados en entidades de aplicación (*DomainApplications*) y recurso (*Resources*).

El único elemento común a todos los dominios es el recurso que representa a los nodos de procesamiento (*procNode*). Estos ofrecen tanto recursos de procesamiento donde se ejecutarán los agentes que representan a las entidades del dominio en tiempo de ejecución, como los recursos físicos que utiliza la funcionalidad asociada (hardware y software de interfaz con el entorno, tales como cámaras, sensores, actuadores, recursos de red, etc.), que se caracterizan como servicios ofrecidos (*services*). Así mismo, cuando el dominio lo requiera, será posible definir las entidades recurso específicas de dominio que ofrecen servicios. La idea de servicio engloba desde conceptos genéricos, tales como la disponibilidad de una determinada librería software o recurso hardware, hasta conceptos muy ligados a un ámbito concreto, como puede ser la posibilidad de realizar una determinada operación en el caso de máquinas en sistemas de fabricación, o robots móviles que realizan operaciones de transporte. Es importante señalar que a nivel de Modelo de Sistema los servicios se caracterizan mediante un identificador único, siendo su semántica definida en el modelo de datos gestionado por las entidades del dominio.

Por otro lado, en el grupo *DomainApplications* se recogen las aplicaciones propias del dominio: las entidades que las forman y sus características, la jerarquía entre ellas y las relaciones de dependencia entre entidades.

La caracterización de las entidades de dominio (recursos y aplicaciones), se realiza mediante los mecanismos que ofrece la tecnología XML Schema: tipos de datos que permiten definir las características de cada concepto (*complex type*), jerarquía entre conceptos y relaciones de dependencia (*key-keyref*), entre otros.

Finalmente, las entidades de aplicación pueden requerir servicios ofrecidos por las entidades recurso, por lo que pueden existir dependencias entre ellas. En la Sección 4 se presenta un caso de estudio concreto y se definen tanto las aplicaciones de dominio como el recurso de procesamiento. Como ya se ha comentado, este recurso es genérico pero sus características pueden ser extendidas en cada dominio particular con los servicios ofrecidos.

3 GESTIÓN DEL MODELO DE SISTEMA

El Modelo de Sistema se crea cuando se registran aplicaciones y recursos de dominio. A partir de que se crea es un elemento vivo en el que pueden existir nuevas altas, bajas y actualizaciones, debido a la gestión de la ejecución de las aplicaciones y del estado del sistema. Sin embargo, el proceso de registro de aplicaciones y recursos es diferente. Los recursos del sistema son parte de la plataforma, y en su arranque se lanza el agente que lo gestiona y que registra tanto el recurso como los servicios que ofrece a las aplicaciones. Sin embargo, las aplicaciones se registran en el sistema de forma externa (el usuario o desde una herramienta que maneja el usuario). Por ello, es necesario un proceso de validación previo, que asegure que la definición de la aplicación cumple con la estructura definida para dicho dominio. En este apartado se describe el proceso de registro de entidades recurso y aplicación, haciendo especial énfasis en las de aplicación, ya que requieren de esta validación previa. Por último, se presenta la interfaz para llevar a cabo su registro. Todo ello, independientemente de la estructura de dominio concreta.

3.1 REGISTRO DE RECURSOS

Como ya se ha comentado, los recursos se auto-registran en el momento de su arranque. Más concretamente, en el arranque se lanza el agente que lo supervisa, quien realiza el registro en el Modelo de Sistema. Por lo tanto, al tratarse de información que proviene de agentes pertenecientes al sistema, no es necesario validarla y se añadirá directamente al Modelo de Sistema (*SystemModel.xml*), creando un nuevo elemento, hijo de sistema, elemento raíz, y dentro del grupo *Resources*. Las características propias proporcionadas se guardan como atributos del nuevo elemento, mientras que por cada servicio ofrecido se creará un elemento hijo. La plataforma asigna un identificador único a cada uno de los elementos añadidos al Modelo de Sistema.

https://doi.org/10.17979/spudc.9788497497169.133

Por ejemplo, en el caso de nodos de procesamiento, sus características propias se refieren a memoria total, número de núcleos, velocidad de cada núcleo, etc. (ver Figura 1). En la sección 4 se describe un ejemplo de recurso extendido con servicios propios del dominio.

3.2 REGISTRO DE APLICACIONES

A diferencia de los recursos, el registro de las aplicaciones lo realizan actores externos a la plataforma, por lo que es necesario validar la información recibida antes de añadirla al Modelo de Sistema. Precisamente con el objetivo de asegurar un Modelo de Sistema siempre correcto y completo (sigue fielmente el XML Schema de dominio definido, *SystemModel.xsd*), se propone un proceso de registro genérico en dos pasos (ver Figura 2):

(1) Registro y validación unitaria de entidades de la aplicación: indicando quién es su padre dentro de la jerarquía que define la aplicación (implica un registro *top-down*). Este primer paso se realiza en base a dos XML Schema distintos:
   - *TConcepts.xsd*: contiene la definición de tipos de entidades de aplicación, lo que incluye sus características relevantes y su correspondiente tipo de dato, distinguiéndose entre obligatorias y opcionales.
   - SystemElements.xsd: incluye al anterior para identificar las entidades registrables.

(2) Validación de la estructura de aplicación: que comprende la validación de la jerarquía y de las relaciones entre entidades. Sólo cuando ésta es correcta, la aplicación pasa a formar parte del Modelo de Sistema (y por tanto, se pueda lanzar su ejecución). Este segundo paso se realiza en base al XML Schema *AppValidation.xsd*, que contiene la definición jerárquica de la aplicación. Para ello, redefine los tipos de entidades que tienen jerarquía mediante extensiones al XML Schema *TConcepts.xsd*. Además, incluye las relaciones de dependencia entre entidades de aplicación no relacionadas jerárquicamente.

Así, cuando se solicita el registro de una nueva entidad de aplicación, en primer lugar se valida la relación de jerarquía padre-hijo. Para ello, se comprueba que existe su padre en el Modelo de Sistema. Es decir, que ya está previamente registrado o que es el propio sistema. Además, también se comprueba que el tipo de padre sea correcto. Si la comprobación tiene éxito, se crea un fichero XML temporal, *Element.xml*, con la nueva entidad y todas sus características, que se valida contra el XML Schema *SystemElements.xsd* (validación unitaria en Figura 2). A continuación, y en caso de que la nueva entidad presente dependencias de recurso, se comprueba que en el Modelo de Sistema existen recursos que proporcionen los servicios requeridos.

El registro validado se va almacenando en un fichero XML intermedio, llamado *Registering.xml*, que contendrá la jerarquía completa. Por lo tanto, tras superar todas las comprobaciones anteriores se asigna un identificador único a la nueva entidad que se añade a dicho fichero, creando un elemento XML hijo del padre indicado.

Finalmente, cuando el usuario lo solicite se valida la jerarquía completa contra el XML Schema *AppValidation.xsd* y, si finaliza con éxito, la nueva aplicación se guarda en el fichero *SystemModel.xml*. De esta forma se asegura que el Modelo de Sistema siempre es correcto, no siendo ya necesario validarlo contra el XML Schema *SystemModel.xsd*.

3.3 INTERFAZ

La plataforma MAS-RECON gestiona un Modelo de

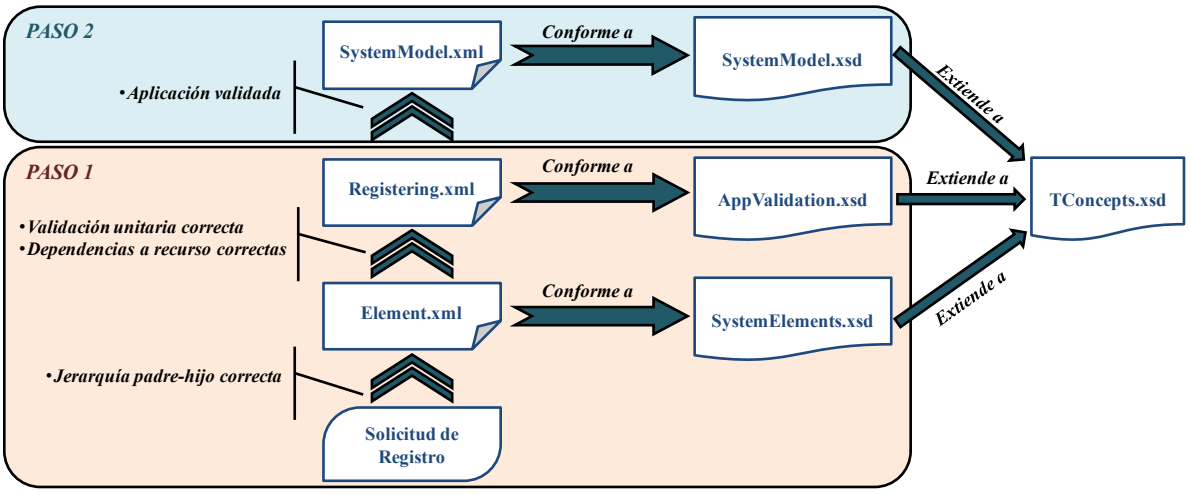

Figura 2: Proceso de registro de objetos de aplicación.

Sistema centralizado, accesible desde cualquier agente del sistema durante el registro y ejecución de las aplicaciones. Es gestionado desde un único agente orquestador que ofrece mecanismos para:

- Añadir/eliminar entidades al modelo.
- Actualizar información en tiempo de ejecución.
- Realizar consultas en tiempo de ejecución.

Estos mecanismos se pueden invocar a través de las tres interfaces mostradas en la Figura 3, centrándose este apartado en los dos primeros que son los empleados durante el proceso de registro. Es importante señalar que esta definición de interfaces se lleva a cabo a nivel conceptual, ya que a nivel de implementación se trata de mensajes ACL intercambiados entre agentes.

En concreto, la interfaz *ISystemElementRegistration* consta de los tres siguientes métodos, que serán usados, bien por agentes del sistema bien por actores externos, para añadir/eliminar entidades al modelo:

```
String resRegister (String seType,
 HashMap<String,String> attribList,
 List<String> provServs);
```

Este método realiza el registro de un recurso, presentado en el apartado 3.1. Además del nuevo tipo de recurso (*seType*), también recibe la lista de características propias en forma de pares "nombre-valor" (*attribList*) y los servicios ofrecidos a las aplicaciones (*provServs*).

```
String appRegister (String seType,
 String parentID, HashMap<String,
 String> attribList, HashMap<String,
 HashMap<String,String>> reqServs);
```

Lleva a cabo el registro de una entidad de aplicación, descrito en el apartado 3.2. Para ello necesita conocer los siguientes datos: el tipo de la nueva entidad (*seType*), el identificador de su padre (*parentID*), la lista de atributos que lo caracterizan (en forma de pares "nombre-valor", *attribList*) y los servicios requeridos (*reqServs*).

En ambos casos, si el registro finaliza con éxito se devuelve el identificador asignado por la plataforma. En caso contrario, retornan un código de error.

```
Void deregister (String seID);
```

Por último, el método *deregister* elimina del modelo la entidad indicada por su identificador (*seID*) y toda su jerarquía, en caso de ser necesario.

También se ha implementado la interfaz *IValidate* que consta de un único método que valida el fichero intermedio *Registering.xml* contra el XML Schema *AppValidation.xsd*. Devuelve un código que indica si la operación ha terminado con éxito o con error:

```
String appValidate;
```

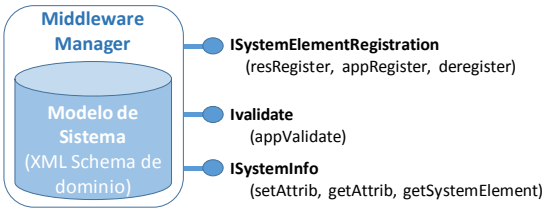

Figura 3: Interfaz para la gestión del Modelo de Sistema.

4 CASO DE ESTUDIO

En este apartado se presenta la validación de la propuesta de modelos de dominio para el caso de sistemas de control de proceso continuo descritos por diagramas de proceso e instrumentación (PI&D). A lo largo de los siguientes párrafos se describen las principales entidades que conforman este tipo de sistemas, identificando sus características relevantes, las relaciones de jerarquía entre ellas y sus relaciones de dependencia. En definitiva, se definen los XML Schema *TConcepts.xsd*, *SystemElements.xsd* (ver Figura 4), *AppValidation.xsd* (ver Figura 5) y *SystemModel.xsd* del dominio.

Desde el punto de vista de los recursos, además de los nodos de procesamiento, comunes a todos los dominios, se precisa también de dispositivos de campo (*fieldDevice*) que ofrecen servicios de medición y/o actuación.

Como ejemplo ilustrativo, se definen los instrumentos básicos de los PI&Ds (controladores y transmisores) y el bucle de control al que pertenecen. La entidad de mayor nivel representa los diferentes equipamientos a controlar dentro de una planta (*ECS*, *Equipment Control System*), que pueden contener diferentes bucles de control (*controlLoop*). Cada bucle es responsable del control de una determinada variable objetivo (*processOutput*) y tiene asignado un número (*loopNumber*) que identifica a todos los instrumentos empleados para llevar a cabo esa función: transmisores y controladores.

Así, dentro de un bucle de control se pueden encontrar uno o varios transmisores (*transmitters*) que se caracterizan por el número de bucle, el tipo de señal de proceso que convierten (*processVariable*), su rango de entrada (*inRange_L*, *inRange_H*) y de salida (*outRange_L*, *outRange_H*).

Por otro lado, dentro del bucle también puede haber uno o más controladores. Al ser una prueba de concepto, en este trabajo se muestra cómo caracterizar controladores PID y anticipativos. En el primer caso (*contPID*), además del número de bucle (*loopNumber*) y de la variable de proceso controlada (*processVariable*), se deben recoger aquellos parámetros relevantes para su sintonización (acción

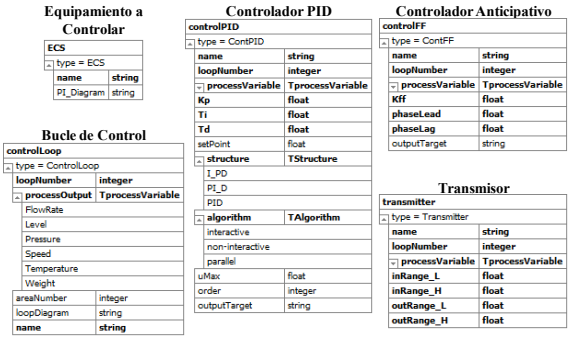

Figura 4: XML Schema *SystemElements.xsd* para sistemas de control de variables de proceso.

proporcional (*Ti*), integradora (*Kp*) y derivativa (*Td*)), su estructura (*structure*), el algoritmo que implementan (*algorithm*) y la consigna (*setPoint*). Cuando sea necesario, se puede indicar el valor limitador que evite el efecto wind-up (*uMax*). En el caso de controladores anticipativos (*contFF*) las características relevantes se refieren a la ganancia (*Kff*), adelanto y retardo de fase (*phaseLead* y *phaseLag*, respectivamente).

Además, es posible crear diferentes estructuras de control, que pueden ir desde la realimentación simple hasta estructuras más complejas, tales como control en cascada, control anticipativo o combinaciones de ellos. Para establecer las relaciones entre controladores que permitan definir estas estructuras de control se han introducido los siguientes atributos opcionales. Para identificar el orden de los controladores PID dentro de una estructura en cascada se ha incluido el atributo opcional *order*. Además, para indicar quién recibe la señal de control generada por el controlador se emplea el atributo opcional *outputTarget*, que hace referencia al nombre del controlador PID que recibe dicha señal de control. Este mecanismo también se puede emplear para relacionar un controlador anticipativo y el correspondiente controlador en lazo cerrado. Para asegurar que el valor de este atributo *outputTarget* se corresponde con el nombre de una entidad de tipo PID, se ha establecido una relación de dependencia entre ambas entidades mediante las restricciones *key-keyref* resaltadas en el XML Schema *AppValidation.xsd* de la Figura 5

Estos XML Schema del dominio se han aplicado para la definición del sistema de control presentado en la Figura 6. El diagrama P&ID de la figura muestra el sistema de control de temperatura y nivel de un líquido en un tanque con agitador. El nivel se controla manipulando el caudal de agua fría que alimenta al tanque, mediante un controlador PID en realimentación simple. La temperatura del fluido se controla manipulando el caudal de vapor al serpentín, mediante un sistema de control avanzado que combina realimentación simple y control anticipativo.

Haciendo uso de la interfaz propuesta en la sección anterior se han registrado las entidades descritas a continuación, obteniendo el fichero intermedio *Registering.xml* de la Figura 6:

- Un equipamiento (*ECS*) que representa el tanque a controlar.
- Dos bucles de control (*controlLoop*), resaltados en color azul.
- El primero, con número de bucle 1, controla el nivel. Consta de un transmisor (*LT1*) que precisa de un *FieldDevice* que ofrezca servicio de sensor de nivel y un controlador PID (*LC1*).
- El segundo, con número de bucle 2, encargado del control de la temperatura. En este caso se emplean dos transmisores: uno para medidas de temperatura (*TT2*) y otro para medidas de flujo (*FT2*), que también requieren los correspondientes servicios de entidades recurso de tipo *FieldDevice*. Este bucle también contiene dos controladores: un PID (*TC2*) y un anticipativo (*FF2*), cuya relación de dependencia se ha resaltado en color verde.

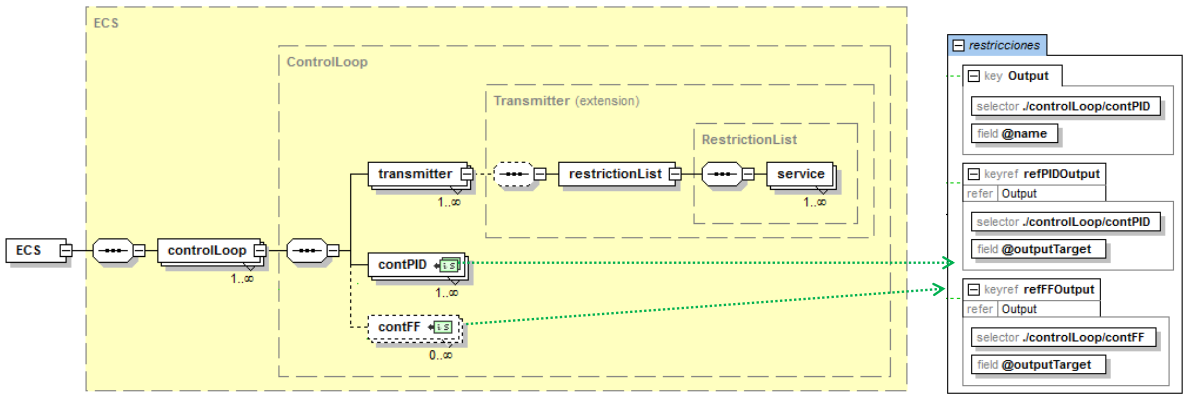

Figura 5: XML Schema *AppValidation.xsd* para sistemas de control de variables de proceso.

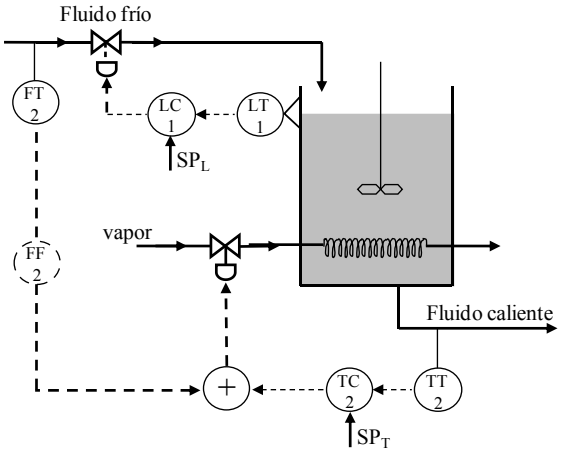

Figura 6: Sistema de Control de temperatura y nivel de un tanque.

5 CONCLUSIONES Y TRABAJO FUTURO

Las plataformas de gestión velan por el cumplimiento de los requisitos de las aplicaciones, gracias a que disponen de una visión y control completo tanto de las aplicaciones como de los recursos, basándose en la información almacenada en su Modelo de Sistema. En este contexto, la metodología basada en modelos propuesta en este trabajo permite definir las particularidades de un dominio con el objetivo de permitir la gestión de sus aplicaciones independiente de dominio.

En concreto, la metodología propuesta permite identificar las entidades que conforman el dominio, su estructura de aplicación y las relaciones que se puedan establecer entre ellas así como sus dependencias a servicios ofrecidos por recursos del sistema. Se trata de una metodología incremental basada en la definición de diferentes XML Schema que recogen toda esta información y que asegura siempre un Modelo de Sistema completo y correcto.

Por lo tanto, y tal y como se ha demostrado en su aplicación en el dominio de sistemas de control de variables de proceso, se ha dado el primer paso para generalizar la plataforma de gestión MAS-RECON. Así, el trabajo futuro está orientado hacia la generalización de toda la plataforma. Para ello se modificará su arquitectura, proporcionando plantillas de agente, APIs y ontologías genéricas, comunes a cualquier dominio. Será necesaria, también, una metodología para particularizar todos estos elementos a un dominio.

Agradecimientos

Este trabajo se ha subvencionado en parte por MCIU/AEI/FEDER, UE bajo el proyecto RTI2018-096116-B-I00, y por el Gobierno Vasco (GV/EJ) bajo el proyecto IT1324-19.

English summary

TOWARDS A GENERIC PLATFORM FOR THE MANAGEMENT OF DYNAMICALLY RECONFIGURABLE APPLICATIONS

Abstract

The so-called Internet of Things (IoT) has allowed a global interconnection between devices of very different nature, making possible the development of applications that sense their context, process and interpret the data acquired, and, if needed, react to changes on it. There are management platforms in charge of managing the execution of these

```
<?xml version="1.0" encoding="UTF-8"?>
<ECS xmlns="RepositoryContent" name="tempLevelTank" ID="11" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="RepositoryContent f
%C3%ADculos,%20Congresos,%20etc/JA2019/Control/AppValidation.xsd">
    <controlLoop loopNumber="1" processOutput="Level" name="levelControl" ID="12">
        <transmitter name="LT1" loopNumber="1" processVariable="Level" inRange_L="0" inRange_H="5" outRange_L="0" outRange_H="100" ID="13">
            <restrictionList se="FieldDevice" ID="14">
                <service refServID="levelSensor" ID="15"/>
            </restrictionList>
        </transmitter>
        <contPID name="LC1" loopNumber="1" processVariable="Level" Kp="1.2" Ti="15" Td="0" structure="PID" algorithm="non-interactive" ID="16"/>
    </controlLoop>
    <controlLoop loopNumber="2" processOutput="Temperature" name="TempControl" ID="17">
        <transmitter name="TT2" loopNumber="2" processVariable="Temperature" inRange_L="60" inRange_H="100" outRange_L="0" outRange_H="100" ID="18">
            <restrictionList se="FieldDevice" ID="19">
                <service refServID="tempSensor" ID="20"/>
            </restrictionList>
        </transmitter>
        <transmitter name="FT2" loopNumber="2" processVariable="FlowRate" inRange_L="0" inRange_H="10" outRange_L="0" outRange_H="100" ID="21">
            <restrictionList se="FieldDevice" ID="22">
                <service refServID="FlowSensor" ID="23"/>
            </restrictionList>
        </transmitter>
        <contPID name="TC2" loopNumber="2" processVariable="Temperature" Kp="2.53" Ti="42" Td="10" structure="PID" algorithm="non-interactive" ID="24"/>
        <contFF name="FF2" loopNumber="2" processVariable="FlowRate" Kff="-1.11" phaseLead="40" phaseLag="52.5" ID="25" outputTarget="TC2"/>
    </controlLoop>
</ECS>
```

Figura 7: Fichero Registering.xml para el Sistema de Control de temperatura y nivel de un tanque.

applications. However, they are usually ad-hoc solutions focused on solving a concrete problem in a concrete domain. In this context, this work proposes a model-based approach for defining the characteristics related to a domain, separating the MAS-RECON platform, developed by the authors in previous works, from applications. The proposal has been validated in the domain of process control systems, characterized by their Piping and Instrumentation Diagram (P&ID).

Keywords: Internet of Things; Multi-Agent Systems; Domain models.

Referencias

[1] Almeida J. P. A., van Sinderen M., Ferreira Pires L., Wegdam M. *Platform-independent dynamic reconfiguration of distributed applications*. In Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 2004, pp. 286–291.

[2] Armentia A., Gangoiti U., Priego R., Estévez E., Marcos M. (2015). *Flexibility support for homecare applications based on models and multi-agent technology*. Sensors, Vol: 15, Issue: 12, pp. 31939-31964.

[3] Bajo J., Fraile J. A., Pérez-Lancho B., Corchado J. M. (2010). *The THOMAS architecture in Home Care scenarios: A case study*. Expert Systems with Applications. Vol: 37, Issue: 5, pp. 3986–3999.

[4] Bello O., Zeadally S. (2019). *Toward efficient smartification of the Internet of Things (IoT) services*. Future Generation Computer Systems. Vol: 92, pp. 663–673.

[5] Borgia E. (2014). *The Internet of Things vision: Key features, applications and open issues*. Computer Communications. Vol: 54, pp. 1–31.

[6] Boyes H., Hallaq B., Cunningham J., Watson T. (2018). *The industrial internet of things (IIoT): An analysis framework*. Computers in Industry. Vol: 101, pp. 1-12.

[7] Gangoiti U., Armentia A., Priego R., Estévez E., Marcos M. *MAS-RECON. Middleware Reconfigurable Basado en Multiagentes*. In Proceedings of the XXXVII Jornadas de Automática, 2016, pp. 884-890.

[8] Gui N., De Florio V., Sun H., Blondia C. (2011). *Toward architecture-based context-aware deployment and adaptation*. Journal of Systems and Software, Vol: 84, Issue: 2, pp. 185–197.

[9] Hallsteinsen S., Geihs K., Paspallis N., Eliassen F., Horn G., Lorenzo J., Mamelli A., Papadopoulos G. A. (2012). *A development framework and methodology for self-adapting applications in ubiquitous computing environments*. Journal of Systems and Software, Vol: 85, Issue: 12, pp. 2840–2859.

[10] ISO/IEC. *Information technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation — Schematron. ISO/IEC 19757-3:2016(E)*. [Online]. Available: http://schematron.com/. [Accessed: 31-May-2019].

[11] López M., Martín J., Gangoiti U., Armentia A., Estévez E., Marcos M. *Tolerancia a Fallos en Sistema de Fabricación Flexible basado en MAS*. In Proceedings of the XXXIX Jornadas de Automática, 2018, pp. 799–805.

[12] Noguero A., Calvo I., Pérez F., Almeida L. (2013). *FTT-MA: a flexible time-triggered middleware architecture for time sensitive, resource-aware AmI systems*. Sensors, Vol: 13, Issue: 5, pp. 6229–6253.

[13] Tamura G., Casallas R., Cleve A., Duchien L. (2014). *QoS contract preservation through dynamic reconfiguration: A formal semantics approach*. Science of Computer Programming. Vol: 94, Issue: P3, pp. 307–332.

[14] W3C. *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. 2012.