

**Mecanismos de adaptación autonómica de arquitectura software para la  
plataforma Smart Campus UIS**

**Daniel David Delgado Cervantes**

**Trabajo de grado para optar el título de Ingeniero de Sistemas**

**Director**

**PhD. Gabriel Rodrigo Pedraza Ferreira**

**Escuela De Ingeniería De Sistemas e Informática**

**Codirector**

**MSc. Henry Andrés Jiménez Herrera**

**Escuela De Ingeniería De Sistemas e Informática**

**Universidad Industrial de Santander**

**Facultad De Ingenierías Fisicomecánicas**

**Escuela De Ingeniería De Sistemas E Informática**

**31 de julio de 2023**

## Dedicatoria

## Agradecimientos

## Contenido

Introducción . . . . .	1
<b>1. Generalidades del proyecto</b>	<b>3</b>
1.1. Planteamiento del problema . . . . .	3
1.2. Justificación del problema . . . . .	4
1.3. Objetivos . . . . .	4
1.3.1. Objetivo General . . . . .	4
1.3.2. Objetivos Específicos . . . . .	4
1.4. Marco Referencial . . . . .	5
1.4.1. Marco Teórico . . . . .	5
1.4.2. Marco Referencial . . . . .	5
1.4.3. Estado del Arte . . . . .	11
<b>2. Lenguajes de Descripción de Arquitecturas</b>	<b>12</b>
2.1. La necesidad de una arquitectura de referencia . . . . .	12
2.2. Criterios de selección . . . . .	12
2.3. Búsqueda de Alternativas . . . . .	14
2.4. Un nuevo modelo para Smart Campus . . . . .	16
2.5. Sintaxis de la notación . . . . .	19
<b>3. Comparación de Arquitecturas</b>	<b>22</b>
3.1. El manejador . . . . .	22
Referencias . . . . .	23

## A. Ejemplo de un YAML de una aplicación válida

27

## Lista de figuras

1.	El ciclo auto-adaptativo MAPE-K. . . . .	7
2.	Versión 2 del modelo concepto planteado por . . . . .	17
3.	Versión 1 del metamodelo planteado para SCampusADL . . . . .	18
4.	Diagrama de rail de la sintaxis definida para la notación de SmartCampusADL	19
5.	Diagrama de rail de la sintaxis definida para la notación del contexto geográfi- co de la aplicación . . . . .	20
6.	Diagrama de rail de la sintaxis definida para la notación de los componentes de la aplicación . . . . .	20

**Lista de tablas**

1. Criterios usados para la determinación de la notación a utilizar . . . . . 13
2. Evaluación de las alternativas en función de los criterios establecidos . . . . . 16

## Introducción

La computación autónoma, concebida inicialmente por IBM en el año 2001, se refiere al uso de sistemas auto-gestionados con la capacidad de operar y adaptarse, o en lo posible, sin la intervención de un ser humano. En este sentido, este acercamiento tiene como objetivo la creación de sistemas computacionales capaces de reconfigurarse en respuesta a cambios en las condiciones del entorno de ejecución al igual que los objetivos del negocio (Horn, 2001).

Esta autonomía es adquirida con el uso de ciclos de control, en el caso de la computación autónoma, de los ciclos más populares es el ciclo MAPE-K, sigla de *Monitor Analyze Plan Execute - Knowledge* (Arcaini, Riccobene, y Scandurra, 2015). Estos le dan la capacidad al sistema de monitorear tanto su estado actual como el entorno en el que este se encuentra, analizar la información recolectada para luego planear y ejecutar los cambios requeridos sobre el sistema a partir de una base de conocimiento (Rutanen, 2018).

Una de las áreas que pueden verse especialmente beneficiadas de la computación autónoma es la del internet de las cosas (IoT). Algunos de estos aspectos están relacionados con la heterogeneidad de estos dispositivos, en cuanto a marcas, protocolos y características; la dinamicidad, en cuanto al movimiento que estos presentan entre entornos de ejecución o incluso una desconexión; al igual que la distribución geográfica de estos lo cual dificulta la intervención directa sobre ellos (Tahir, Mamoon Ashraf, y Dabbagh, 2019).

Esto puede verse el Smart Campus UIS, una plataforma IoT de la Universidad Industrial de Santander, que permite usar dispositivos con el fin de monitorear y recolectar de información en tiempo real con el objetivo de apoyar la toma de decisiones, mejora de servicios, entre otros (Jiménez, Cárcamo, y Pedraza, 2020).

Ahora, esta plataforma ha tenido esfuerzos en el desarrollo de características propias de un sistema autónomo. Uno de estos ha sido la integración de mecanismos para la auto-descripción de la arquitectura desplegada en un momento dado (Jiménez Herrera, 2022). En el contexto de la computación autónoma, esta capacidad hace parte del monitoreo dentro



del ciclo de control.

Partiendo de lo anterior, y con la intención de dar continuidad con los esfuerzos de desarrollo realizados en Smart Campus UIS, se plantea como caso de estudio, a partir de las capacidades de auto-descripción que la plataforma, se pretende el proveer a esta la capacidad de auto-configuración a partir de un conjunto de mecanismos de adaptación que permitan, desde la definición de un objetivo a lograr, la alteración de la arquitectura desplegada.

# Capítulo 1

## Generalidades del proyecto

### 1.1 Planteamiento del problema

La complejidad de los sistemas computacionales tiene origen en diversos factores. El aumento de la cantidad de dispositivos que los componen; la heterogeneidad debido a diferentes marcas, protocolos y características; e incluso las cambiantes condiciones de sus entornos de ejecución; dificulta la administración de los sistemas computacionales (Salehie y Tahvildari, 2005).

Una de las posibles maneras de dar solución a esta problemática, en cuanto al manejo de sistemas, está en el área de la computación autónoma. Este acercamiento basado en conceptos biológicos busca solventar los problemas de complejidad, heterogeneidad e incertidumbre (Salehie y Tahvildari, 2005) a partir de la abstracción de las metas de los administradores y delegación del manejo del sistema a sí mismo (Lalanda, Diaconescu, y McCann, 2014).

Considerando lo anterior, una de las aplicaciones de la computación autónoma, dentro del IoT, son los Smart Campus; una variación de las Smart Cities en las cuales se busca la recolección de información y monitoreo en tiempo real con el fin de apoyar la toma de decisiones, mejora de servicios, entre otros (Min-Allah y Alrashed, 2020). Es en este tipo de aplicaciones donde los problemas, en especial la heterogeneidad, dinamicidad, al igual que la distribución geográfica; donde la reducción de la dependencia de intervención humana,

facilitaría el manejo de estos.

## 1.2 Justificación del problema

De lo anterior, surge la pregunta del cómo realizar dichas adaptaciones a la arquitectura, o estado del sistema. Qué clase de mecanismos y requerimientos han de ser satisfechos con el fin de considerar, a un sistema computacional, con la capacidad de auto-adaptarse dados unos objetivos o metas establecidas por los administradores del sistema.

Dentro del marco del presente proyecto, se tiene Smart Campus UIS, una plataforma de IoT de la Universidad Industrial de Santander, en la cual se han realizado implementaciones parciales de una arquitectura autonómica con capacidad de auto-describirse (Jiménez Herrera, 2022), una de las características principales de un sistema autonómico (Horn, 2001).

Dicho esto, se busca explorar algunas de las maneras en las que se realizan adaptaciones con características autonómicas en sistemas computacionales. De esto, se buscaría probar un conjunto de estos mecanismos de modificación del estado, tomando como caso de estudio la implementación de estos en la plataforma de Smart Campus UIS.

## 1.3 Objetivos

### 1.3.1 Objetivo General

- Diseñar un conjunto de mecanismos autonómicos para permitir la adaptación de la Arquitectura Software IoT respecto a un modelo objetivo en la plataforma Smart Campus UIS

### 1.3.2 Objetivos Específicos

- Proponer una notación (lenguaje) para describir una arquitectura objetivo de un sistema software IoT.

- Diseñar un mecanismo para determinar las diferencias existentes entre una arquitectura actual en ejecución y una arquitectura objetivo especificada.
- Diseñar un conjunto de mecanismos de adaptación que permitan disminuir las diferencias entre la arquitectura actual y la arquitectura objetivo.
- Evaluar la implementación realizada a partir de un conjunto de pruebas con el fin de establecer la efectividad de los mecanismos usados.

## 1.4 Marco Referencial

### 1.4.1 Marco Teórico

### 1.4.2 Marco Referencial

#### *Domain-Specific Languages*

Los *Domain-Specific Languages* (DSL), o Lenguaje específico de dominio, son lenguajes de programación los cuales se usan con un fin específico. Estos están orientados principalmente al uso de abstracciones de un mayor nivel debido al enfoque que estos tienen para la solución de un problema en específico (Kelly y Tolvanen, 2008). Ejemplos de este tipo de acercamiento, en el caso del modelado (*Domain-specific modeling*), pueden observarse en los diagramas de entidad relación usados en el desarrollo de modelos de base de datos (Celikovic, Dimitrieski, Aleksic, Ristić, y Luković, 2014).

#### *Computación Autónoma*

El concepto de computación autónoma, definido inicialmente por IBM (2001), se refiere a un conjunto de características que presenta un sistema computacional el cual le permite actuar de manera autónoma, o auto-gobernarse, con el fin de alcanzar algún objetivo establecido por los administradores del sistema (Lalanda y cols., 2014).

Los 8 elementos clave, definidos por IBM, que deberían presentar este tipo de sistemas son:

1. Auto-conocimiento: habilidad de conocer su estado actual, las interacciones del sistema.
2. Auto-configuración: capacidad de reconfigurarse frente a los constantes cambios en el entorno.
3. Auto-optimización: búsqueda constante de optimizar el funcionamiento de sí mismo.
4. Auto-sanación: aptitud de restaurar el sistema en el caso de que se presenten fallas.
5. Auto-protección: facultad de protegerse a sí mismo de ataques externos.
6. Auto-conciencia: posibilidad de conocer el ambiente en el que el sistema se encuentra.
7. Heterogeneidad: capacidad de interactuar con otros sistemas de manera cooperativa.
8. Abstracción: ocultar la complejidad a los administradores del sistema con objetivos de alto nivel de abstracción.

En el caso de que un sistema tenga una implementación parcial de estas características, este podría considerarse autonómico. En este sentido debería tener la capacidad de lidiar con los problemas como la complejidad, heterogeneidad e incertidumbre (Salehie y Tahvildari, 2005) al igual que reducir la cantidad de recursos tanto técnicos como humanos requeridos para mantener los sistemas en funcionamiento.

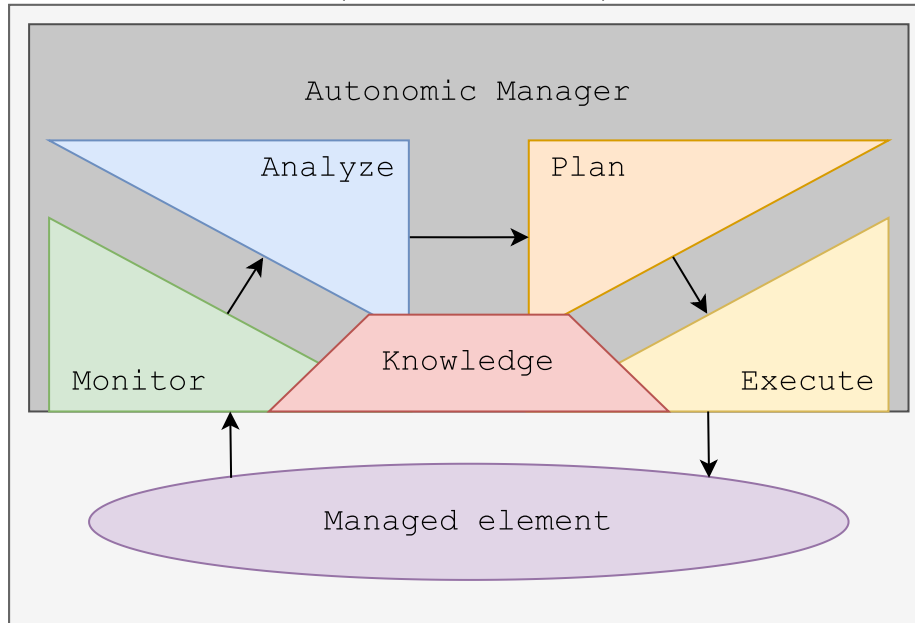
### **MAPE-K**

IBM, en cuanto a la implementación de las características, propone un modelo de ciclo auto-adaptativo, denominado MAPE-K (Krikava, 2013). Este acercamiento, compuesto de cinco fases, es uno de los ciclos de control más usados en implementaciones de sistemas auto-adaptativos y computación autonómica (Arcaini y cols., 2015). En la figura 1, se presentan las fases que *manejador* debe desarrollar para así administrar cada uno de los elementos del

sistema computacional basado en una base de conocimiento común (Gorla, Pezzè, Wuttke, Mariani, y Pastore, 2010).

Figura 1: El ciclo auto-adaptativo MAPE-K.

(Gorla y cols., 2010)



Cada una de estas fases son:

- Monitorear (M): Esta fase se compone de la recolección, filtración y reportar la información adquirida sobre el estado del elemento a manejar.
- Analizar (A): La fase de análisis se encarga del interpretar el entorno en el cual se encuentra, el predecir posibles situaciones comunes y diagnosticar el estado del sistema.
- Planear (P): Durante la planificación se determina las acciones a tomar con el fin de llegar a un objetivo establecido a partir de una serie de reglas o estrategias.
- Ejecutar (E): Finalmente, se ejecuta lo planeado usando los mecanismos disponibles para el manejo del sistema.

Es de resaltar que este modelo, aunque útil para el desarrollo de este tipo de sistemas, es bastante general en cuanto a la estructura y no usan modelos de diseño establecidos

(Ouareth, Boulehouache, y Mazouzi, 2018).

### ***Mecanismos de Descripción***

La fase de monitoreo dentro del ciclo MAPE-K es vital para el funcionamiento del manejador autónomo pues es a partir de la información que se construirá la base de conocimiento requerida por las demás partes del ciclo. Parte de esta, está compuesta por el *estado del sistema* el cual incluye la descripción del sistema en un momento dado (Weiss, Zeller, y Eilers, 2011).

Existen varias maneras de realizar implementaciones de mecanismos de auto-descripción y la utilidad de cada uno de estos varía dependiendo en el tipo de sistema de software que se esté usando. Para el marco del proyecto, nos interesan aquellos que estén orientados a los sistemas embebidos e IoT, algunos de estos son:

- **JSON Messaging:** Iancu y Gatea (2022) plantean un protocolo que emplea mensajería entre *gateways* con el fin de recibir información sobre estas. En términos simples, estas funcionan como un *ping* hacia el nodo que luego retorna sus datos, al igual que los dispositivos conectados a ella, al encargado de recolectar toda esta información con el fin de construir una descripción del sistema.-
- **IoT Service Description Model:** O IoT-LMsDM, es un servicio de descripción desarrollado por Zhen y Aiello (2021) el cual está orientado al contexto, servicios e interfaz de un sistema IoT. De este se espera poder contar no solo con descripciones del estado del sistema en términos del ambiente, pero la funcionalidad (es decir, los *endpoints* a usar) al igual que las estructuras de datos que estos consumen.
- **Adaptadores de Auto-descripción:** En este acercamiento a los mecanismos de auto-descripción, se tienen adaptadores los cuales emplean los datos generados por los sensores del componente gestionado con el fin de realizar la determinación de la arquitectura desplegada. De igual manera, este acercamiento permite realizar modificaciones a la

descripción de manera manual en caso de que se detecten problemas (Jiménez Herrera, 2022).

De esto podemos ver no solo las diferentes maneras en las que las implementaciones realizan las descripciones de los sistemas asociados, sino que también el alcance de estos en cuanto a lo que pueden describir.

### ***Mecanismos de Adaptación***

La adaptación, en el contexto de la computación autónoma, es la parte más importante en cuanto a la auto-gestión de un sistema de software se refiere. Así mismo, presenta el mayor reto debido a la necesidad de modificar código de bajo nivel, tener que afrontarse a incertidumbre de los efectos que pueden tener dichas alteraciones al sistema al igual que lidiar con esto en *runtime* debido a los problemas que el *downtime* tendría en los negocios (Lalanda y cols., 2014).

Esta adaptabilidad puede exponerse en múltiples puntos dentro de un sistema de software. Pueden realizarse adaptaciones en sistema operativo, lenguaje de programación, arquitectura e incluso datos (Lalanda y cols., 2014).

Manteniéndose en el marco del proyecto, son las implementaciones relacionadas con la modificación de la arquitectura los cuales nos interesan. Siendo así, nos centraremos en los mecanismos de adaptación de componentes, o de reconfiguración:

- **Binding Modification:** Este mecanismo hace referencia a la alteración de los vínculos entre los diferentes componentes de la arquitectura. Estos tienen el objetivo de modificar la interacción entre componentes, lo que es especialmente común en implementaciones con *proxies*. Este tipo de mecanismo de adaptación fue usado por Kabashkin (2017) para añadir fiabilidad a la red de comunicación aérea.
- **Interface Modification:** Las interfases funcionan como los puntos de comunicación entre los diferentes componentes de la arquitectura. Siendo así, es posible que la mo-



dificación de estos sea de interés con el fin de alterar el comportamiento de un sistema al igual que soportar la heterogeneidad del sistema. Esto puede verse en el trabajo desarrollado por Liu, Parashar y Hariri (2004) en donde definen la utilidad de dichas adaptaciones al igual que la implementación de las mismas.

- **Component Replacement, Addition and Subtraction:** En términos simples, este mecanismo se encarga de alterar los componentes que componen la arquitectura; de esta manera, modificando su comportamiento. Ejemplos de esto puede verse en el trabajo de Huynh (2019) en el cual se evalúan varios acercamientos a la reconfiguración de arquitecturas a partir del remplazo de componentes a nivel individual al igual que grupal.

Este acercamiento a la mutación de la arquitectura también puede verse en el despliegue de componentes como respuesta a cambios en los objetivos de negocio de las aplicaciones al igual que como respuesta a cambios inesperados dentro de la aplicación. Esto puede verse en trabajos como el de Patouni (2006) donde se realizan este tipo de implementaciones.

### ***Internet of Things***

El Internet de las cosas, o IoT (Internet of Things); es una de las áreas de las ciencias de la computación en la cual se embeben diferentes dispositivos en objetos del día a día. Esto les da la capacidad de enviar y recibir información con el fin de realizar monitoreo o facilitar el control de ciertas acciones (Berte, 2018).

Esta tecnología, debido a su flexibilidad al igual que el alcance que puede tener, presenta una gran cantidad de aplicaciones que va desde electrónica de consumo hasta la industria. Encuestas realizadas en el 2020 reportan su uso en smart homes, smart cities, transporte, agricultura, medicina, etc. (Dawood, 2020). Su impacto no ha sido poco.

## ***Sistemas Embebidos***

Los sistemas de cómputo embebidos hacen referencia a un sistema compuesto de micro-controladores los cuales están orientados a llevar a cabo una función o un rango de funciones específicas (Heath, 2002). Este tipo de sistemas, debido a la posibilidad de combinar hardware y software en una manera compacta, se ha visto en múltiples campos de la industria como lo son el sector automotor, de maquinaria industrial o electrónica de consumo (Deichmann, Doll, Klein, Mühlreiter, y Stein, 2022).

## ***Smart Campus***

Un Smart Campus, equiparable con el concepto de Smart City, es una plataforma en la que se emplean tecnologías, sumado a una infraestructura física, con la cual se busca la recolección de información y monitoreo en tiempo real (Min-Allah y Alrashed, 2020). Los datos recolectados tienen el objetivo de apoyar la toma de decisiones, mejora de servicios, entre otros (Anagnostopoulos, 2023).

Estas plataformas, debido a su escala y alcance en cuanto a la cantidad de servicios que pueden ofrecer, requieren de infraestructuras tecnológicas las cuales den soporte a los objetivos del sistema. Es posible ver implementaciones orientadas a microservicios en trabajos como los de Jiménez, Cárcamo y Pedraza (2020) donde se desarrolla una plataforma de software escalable con la cual se pueda lograr interoperatividad y alta usabilidad para todos.

## ***Location-based Services***

### **1.4.3 Estado del Arte**

## Capítulo 2

# Lenguajes de Descripción de Arquitecturas

### 2.1 La necesidad de una arquitectura de referencia

La necesidad de una arquitectura de referencia, o arquitectura objetivo; es una parte esencial dentro de la computación autonómica. Esta forma parte directamente a la base de conocimiento (K), y, de manera indirecta, a los objetivos del administrador del sistema (Lalanda y cols., 2014, p. 24).

Con el fin de fijar un objetivo para nuestro sistema autonómico, fue necesario determinar una manera de realizar la declaración de dicha arquitectura, que estableciera un punto de referencia. Esto permitiría, durante el proceso de comparación de la arquitectura establecida con la que se encuentra en tiempo de ejecución en un momento dado, identificar el estado del sistema.

### 2.2 Criterios de selección

Con el fin de establecer la notación a usar para la declaración de las arquitecturas objetivo, se establecieron unos lineamientos con los cuales se realizaría la evaluación de las diferentes notaciones ya desarrolladas anteriormente. De esta manera se podría escoger la manera a

representar los modelos, o en el caso de ser necesario, establecer los criterios por el cual se podría desarrollar uno.

En la tabla 1, se presentan los criterios usados para la selección, con su respectiva explicación y valor considerado para la selección.

Tabla 1: Criterios usados para la determinación de la notación a utilizar

	Criterio	Explicación	Valor
C1	Describir la arquitectura de un sistema IoT	Este criterio es una base a establecer con el fin de descartar aquellos lenguajes de notación generales o no necesariamente usados para la descripción de arquitecturas IoT.	Alto
C2	Permitir la especificación de la ubicación del componente	La especificación de la ubicación de los componentes es importante en los sistemas IoT, especialmente dentro del contexto del proyecto en el cual se está trabajando con un Smart Campus; ya que los componentes pueden estar distribuidos en diferentes ubicaciones físicas y la evaluación de su integridad puede depender de su presencia en un lugar dado.	Alto
C3	Habilitar el modelado del componente a nivel de sus entradas	Es importante poder describir las entradas de los componentes, específicamente los datos que manejan, así como su rol dentro del sistema.	Alto
C4	Modelar el comportamiento de los componentes	La notación debe permitir modelar el comportamiento de los componentes, de manera que se puedan entender sus interacciones y su función en el sistema IoT. Dentro del contexto del proyecto, no es tan relevante, ya que no se está evaluando la funcionalidad del sistema, sin embargo, para futuros trabajos, podría facilitar la extensión de Smart Campus UIS.	Bajo
C5	Posibilitar el establecer los estados de los componentes	La notación debe poder definir los estados de los componentes. Estos estados pueden ser tanto de comportamiento o operacionales.	Medio
C6	Permitir de exportar el modelo descrito a gráficas u otros formatos	Es importante que la herramienta permita exportar el modelo descrito en diferentes formatos para facilitar su integración con otras herramientas y sistemas, y para permitir su visualización en diferentes formatos.	Medio

### 2.3 Búsqueda de Alternativas

Una vez establecidos los criterios de selección, se realizó una exhaustiva búsqueda de alternativas disponibles en la literatura y en la industria para describir arquitecturas para describir la arquitectura de sistemas IoT. Esta búsqueda se realizó a partir de la revisión en diferentes bases de datos, como *Scopus*; al igual que algunas de las revistas especializadas en el tema, y el internet en general.

Durante la búsqueda, se identificaron una gran variedad de opciones. Sin embargo, la gran mayoría de estos se filtraron, o descartaron; a partir de los criterios de selección establecidos. Esto se debe a que los LDAs usados en la tanto en la industria y academia, como AADL (Architecture Analysis and Design Language), tienen un enfoque a los campos de aviónica, equipos médicos y aeronáutica (lo que complicaría su implementación hacia sistemas de software IoT) (Carnegie Mellon University, 2022, 2017); o SysML, que son demasiado genéricos y abarcan hardware, software e incluso personas y procesos (Object Management Group, 2015).

Entonces, de las posibles opciones de notación, se seleccionaron cinco las cuales fueron evaluadas con el fin de determinar si alguna de las opciones hubiera podido ser usada, o si era necesario desarrollar nuestra propia notación. A continuación, se presentan las alternativas evaluadas:

- **MontiThings:** Basado en MontiArc, otro LDA más general; está diseñado para el modelado y prototipado de aplicaciones de IoT. Este realiza las descripciones de sus arquitecturas en un modelo componente-conector, donde los componentes están compuestos por otros componentes; y los conectores definen la manera en la que se comunican estos componentes a nivel de los datos y la dirección de estos. (Kirchhof, Rumpe, Schmalzing, y Wortmann, 2022b, 2022a)
- **Eclipse Mita:** Mita, creado por la Eclipse Foundation; es un lenguaje de programación orientado al facilitar la programación de sistemas IoT. Aunque como tal no es un LDA,

está orientada a la descripción de los componentes y el comportamiento del sistema establecido, de esta manera, puede generar el código que debe correr en los dispositivos embebidos (Eclipse Foundation, 2018).

- **SysML4IoT:** Es un perfil de SysML<sup>1</sup> en la cual se usan estereotipos de UML con el fin de abstraer las diferentes partes de los sistemas de IoT. Al igual que SysML, este permite el modelado más allá de dispositivos incluso llegando a personas y procesos, con la diferencia del enfoque dado al *dominio de IoT*<sup>2</sup> (Costa, Pires, y Delicato, 2016).
- **ThingML:** Similar a Mita, ThingML, es un lenguaje de modelado el cual tiene capacidades de generar el código requerido por los dispositivos embebidos. En términos del proyecto, este permite el modelado de los sistemas de IoT a partir de *state machine models*<sup>3</sup> los cuales permiten describir los componentes del sistema al igual que el comportamiento de estos (Harrand, Fleurey, Morin, y Husa, 2016).
- **IoT-DDL:** Iot-DDL es un LDA, implementado en XML, que describe objetos dentro de los ecosistemas IoT con base en sus componentes, identidad y servicios entre otros. Este tiene la capacidad de describir parte de la base de conocimiento que tienen los diferentes componentes (Principalmente relaciones y asociaciones entre componentes) (Khaled, Helal, Lindquist, y Lee, 2018).

Una vez seleccionadas las alternativas a evaluar, se utilizó la matriz de evaluación en la tabla 2 para determinar la solución ideal para el desarrollo del proyecto.

---

<sup>1</sup>Los perfiles se refieren a extensiones a UML, en este caso es una extensión de SysML en sí (Andre, 2007).

<sup>2</sup>El *dominio del IoT*, hace referencia al *Architecture Reference Model* establecido por IoT-A, un consorcio Europeo el cual buscaba el establecer un modelo para la interoperatividad de dispositivos IoT. (IoT-A, 2014)

<sup>3</sup>Los *state machine model*, también conocidas como Autómatas Finitos; son modelos matemáticos que describen todos los posibles estados de un sistema a partir de unas entradas dadas (Mozilla Foundation, 2023).

Tabla 2: Evaluación de las alternativas en función de los criterios establecidos

	MontiThings	Eclipse Mita	SysML4IoT	ThingML	IoT-DDL
C1	✓	✓	✓	✓	✓
C2	✗	✗	✗	✗	✗
C3	✓	✓	✓	✓	✓
C4	✓	✓	✓	✓	✗
C5	✓	✓	✗	✓	✗
C6	✗	✗	✗	✓	✗

Partiendo de los resultados de la evaluación, se puede apreciar que ninguno de estos LDA cumple con los criterios establecidos para el proyecto. Aunque estos están orientados hacia la descripción y desarrollo de sistemas IoT, no están enfocados hacia su contexto en términos de aplicación más allá de la definición de comportamiento. Esto se debe a los objetivos de cada una de estas notaciones, de una u otra manera; buscan el representar como tal el sistema IoT en términos de su funcionalidad técnica y no la aplicación en sí.

## 2.4 Un nuevo modelo para Smart Campus

Dado que no hay una alternativa que se adapte completamente a las necesidades del proyecto, se tuvo que definir nuestra propia notación la cual permita modelar las arquitecturas a nivel de aplicación bajo el contexto de un Smart Campus, tomando en cuenta los criterios definidos como guía para el desarrollo.

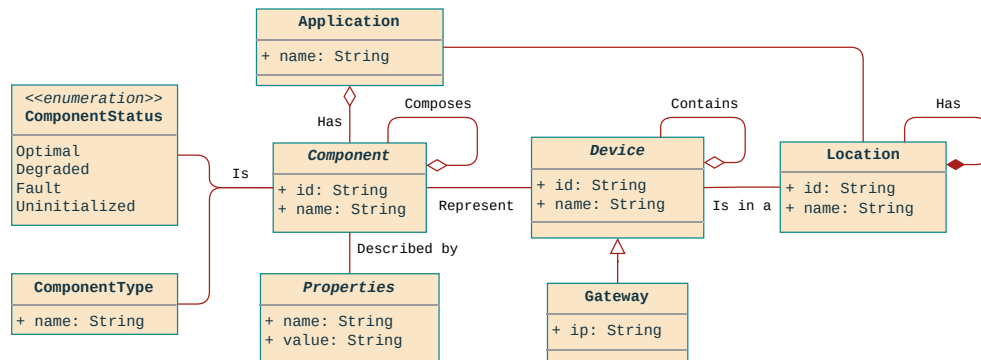
Primeramente, fue necesario que definir un metamodelo el cual permita establecer la manera en la que se verán estas arquitecturas. Para ello, y partiendo de la implementación a realizar, nos basamos parcialmente en el modelo establecido por Jiménez Herrera (2022, p. 63).





Partiendo de esto, se planteó el modelo presente en la figura 3.

Figura 3: Versión 1 del metamodelo planteado para SCampusADL



El enfoque principal del modelo se centra los componentes (*Component*). Estos son las representaciones lógicas, o de software, de las partes de la aplicación. De manera general, pueden ser tanto servicios usados por la aplicación como agregadores de datos, pre-procesadores de datos encargados de transformar los datos recolectados, o incluso los mismos sensores o actuadores que hacen parte del sistema (*componentType*).

Los componentes pueden ser atómicos, funcionando como entidades independientes, o formar parte de componentes, así como integrarse en componentes aún más grandes. Asimismo, los componentes se describen por una serie de propiedades que hacen referencia tanto al comportamiento al igual que las condiciones las con las cuales se puede evaluar el estado del mismo.

Estos elementos están presentes en diversos dispositivos (*Device*), los cuales se encargan de la ejecución de partes de la aplicación, o puntos de comunicación entre diversos componentes (*Gateway*). Estos pueden verse como los recursos físicos de la aplicación los cuales se encuentran ubicados en diferentes puntos, o locaciones (*location*), dentro del Campus (*Campus*).

Lo anterior nombrado hace parte del contexto de la aplicación. Siendo así, estos están relacionados contra la aplicación la cual sirve como un punto de acceso común entre el contexto geográfico y los componentes de software que deben ser manejados.

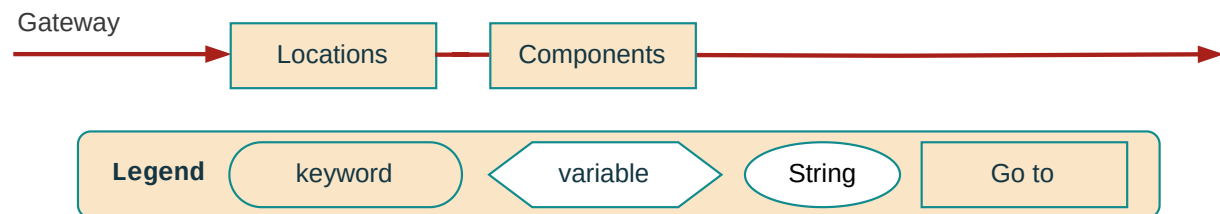
Finalmente, los estados de los componentes (*ComponentStatus*), reflejan el como estos se encuentran dentro del contexto de la aplicación. Estos pueden variar de óptimo (*optimal*), donde todos los componentes están funcionales según las propiedades establecidas; degradado (*degraded*), donde uno o más de los componentes opcionales está fallado; y fallo (*fault*), que es cuando alguno de los dispositivos obligatorios no está en funcionamiento.

## 2.5 Sintaxis de la notación

Partiendo de esto, lo siguiente que se realizó fue la definición de la sintaxis de la notación a usar, basados en lo definido por el metamodelo. Para esto, se decidió a usar **YAML**, un lenguaje de serialización de datos orientado a la legibilidad, reconocido y usado principalmente para la creación de archivos de configuración (Red Hat Foundation, 2023).

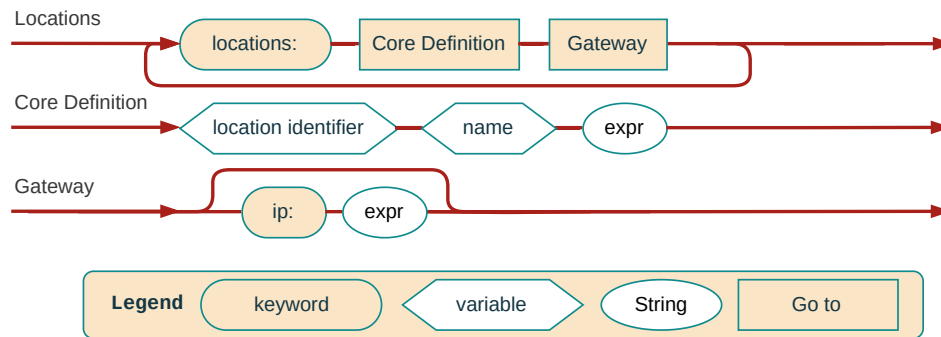
La sintaxis, como puede observarse en la figura 4, se compone de 2 partes: **locaciones**, para establecer el contexto geográfico de la aplicación; y **componentes**, con el cual se definen las partes de la aplicación al igual que las propiedades y lugar en el cual deben estar para el funcionamiento de la aplicación.

Figura 4: Diagrama de rail de la sintaxis definida para la notación de SmartCampusADL



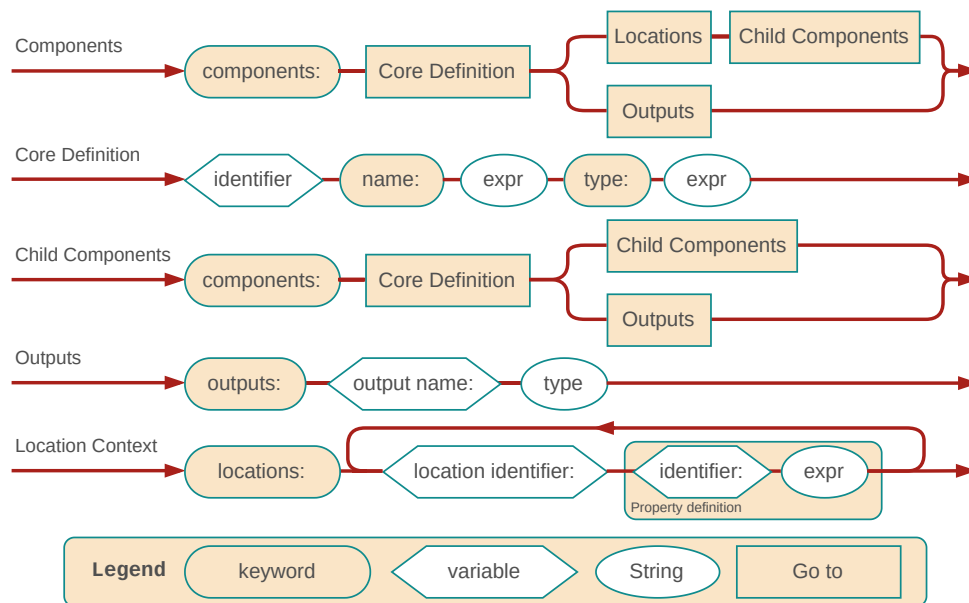
Para especificar las **locaciones** dentro del contexto de la aplicación, se definió la sintaxis que puede observarse en la figura 5. De esta manera, tras especificar la *keyword* **locations**, se pueden los puntos geográficos necesarios, al igual que las locaciones que lo componen. Así mismo, es posible declarar de manera implícita la ip de la *gateway* que se encuentra en el lugar con el fin de poder establecer el origen de los datos recolectados.

Figura 5: Diagrama de rail de la sintaxis definida para la notación del contexto geográfico de la aplicación



Un enfoque similar se aplicó a los componentes. En la figura 6 se presenta la sintaxis a usar para poder definir y ubicar los componentes dentro de la aplicación. De esta forma, se pueden declarar los componentes, al igual que sus propiedades; y algunas de las características que estos deben cumplir para poder ser evaluados dentro del contexto de la aplicación.

Figura 6: Diagrama de rail de la sintaxis definida para la notación de los componentes de la aplicación



Al definir esta notación, hemos creado un marco sólido para representar las aplicaciones de Smart Campus UIS. Este enfoque bien definido nos permitirá avanzar en el desarrollo de nuestras funcionalidades de comparación de los modelos, y la implementación de los

mecanismos de adaptación a usar. Un ejemplo de como se vería un archivo de notación válido para nuestras necesidades puede encontrarse en el anexo A.

## Capítulo 3

# Comparación de Arquitecturas

### 3.1 El manejador

Ahora, con la notación de las arquitecturas objetivo establecidas, podemos continuar con el desarrollo de la implementación del *manejador* encargado de la adaptación y los res

## Referencias

- Anagnostopoulos, T. (2023). Smart campus. En *Iot-enabled unobtrusive surveillance systems for smart campus safety* (p. 17-25). doi: 10.1002/9781119903932.ch3
- Andre, C. (2007). *Uml extensions: the sysml profile*. Descargado 2023-05-19, de [https://www.i3s.unice.fr/~map/Cours/MASTER\\_TSM/Cours7\\_SysML.pdf](https://www.i3s.unice.fr/~map/Cours/MASTER_TSM/Cours7_SysML.pdf) (Presentation on UML extensions)
- Arcaini, P., Riccobene, E., y Scandurra, P. (2015). Modeling and analyzing mape-k feedback loops for self-adaptation. En *2015 ieee/acm 10th international symposium on software engineering for adaptive and self-managing systems* (p. 13-23). doi: 10.1109/SEAMS.2015.10
- Berte, D.-R. (2018, 05). Defining the iot. *Proceedings of the International Conference on Business Excellence*, 12, 118-128. doi: 10.2478/picbe-2018-0013
- Carnegie Mellon University. (2017). *Aadl and osate: A tool kit to support model-based engineering*. Online. Carnegie Mellon University. Descargado de [https://resources.sei.cmu.edu/asset\\_files/FactSheet/2017\\_010\\_001\\_506838.pdf](https://resources.sei.cmu.edu/asset_files/FactSheet/2017_010_001_506838.pdf)
- Carnegie Mellon University. (2022). *Architecture analysis and design language*. Carnegie Mellon University. Descargado de [https://www.sei.cmu.edu/our-work/projects/display.cfm?customel\\_datapageid\\_4050=191439%2C191439](https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439%2C191439)
- Celikovic, M., Dimitrieski, V., Aleksic, S., Ristić, S., y Luković, I. (2014). A dsl for eer data model specification. En *Integrated spatial databases*.
- Costa, B., Pires, P. F., y Delicato, F. C. (2016). Modeling iot applications with sysml4iot. En *2016 42th euromicro conference on software engineering and advanced applications (seaa)* (p. 157-164). doi: 10.1109/SEAA.2016.19
- Dawood, A. (2020, 09). Internet of things (iot) and its applications: A survey. *International Journal of Computer Applications*, 175, 975-8887. doi: 10.5120/ijca2020919916
- Deichmann, J., Doll, G., Klein, B., Mühlreiter, B., y Stein, J. P. (2022, Mar). *Cracking the*

- complexity code in embedded systems development*. McKinsey's Advanced Electronics Practice.
- Eclipse Foundation. (2018). *Eclipse mita*. Eclipse Foundation. Descargado 2023-05-13, de <https://www.eclipse.org/mita/>
- Gorla, A., Pezzè, M., Wuttke, J., Mariani, L., y Pastore, F. (2010, 01). Achieving cost-effective software reliability through self-healing. *Computing and Informatics*, 29, 93-115.
- Harrand, N., Fleurey, F., Morin, B., y Husa, K. (2016, 10). Thingml: a language and code generation framework for heterogeneous targets. En (p. 125-135). doi: 10.1145/2976767.2976812
- Heath, S. (2002). *Embedded systems design*. Elsevier.
- Horn, P. (2001, Oct). *autonomic computing: Ibm's perspective on the state of information technology*. IBM. Descargado de [https://homeostasis.scs.carleton.ca/~soma/biosec/readings/autonomic\\_computing.pdf](https://homeostasis.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf)
- Huynh, N.-T. (2019). An analysis view of component-based software architecture reconfiguration. En *2019 ieee-rivf international conference on computing and communication technologies (rivf)* (p. 1-6). doi: 10.1109/RIVF.2019.8713678
- Iancu, B., y Gatea, A. (2022). Towards a self-describing gateway-based iot solution. En *2022 ieee international conference on automation, quality and testing, robotics (aqtr)* (p. 1-5). doi: 10.1109/AQTR55203.2022.9801938
- IoT-A. (2014). *Internet of things architecture*. Online. Descargado 2023-05-20, de <https://www.iot-a.eu/public/>
- Jiménez, H., Cárcamo, E., y Pedraza, G. (2020). Extensible software platform for smart campus based on microservices. *RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao*, 2020(E38), 270-282. Descargado de [www.scopus.com](http://www.scopus.com)
- Jiménez Herrera, H. A. (2022). *Mecanismos autónomos para la autodescripción de arquitecturas iot distribuidas*. Descargado de <https://noesis.uis.edu.co/items/>

c5d1cce7-3f8f-4e3e-a083-66bd3c4745f3

- Kabashkin, I. (2017). Dynamic reconfiguration of architecture in the communication network of air traffic management system. En *2017 ieee international conference on computer and information technology (cit)* (p. 345-350). doi: 10.1109/CIT.2017.13
- Kelly, S., y Tolvanen, J.-P. (2008). *Domain-specific modeling*. Hoboken, NJ: Wiley-Blackwell.
- Khaled, A. E., Helal, A., Lindquist, W., y Lee, C. (2018). Iot-ddl—device description language for the “t” in iot. *IEEE Access*, 6, 24048-24063. doi: 10.1109/ACCESS.2018.2825295
- Kirchhof, J. C., Rumpe, B., Schmalzing, D., y Wortmann, A. (2022a). *Montithings*. Online. Descargado 2023-05-13, de <https://github.com/MontiCore/montithings>
- Kirchhof, J. C., Rumpe, B., Schmalzing, D., y Wortmann, A. (2022b). Montithings: Model-driven development and deployment of reliable iot applications. *Journal of Systems and Software*, 183, 111087. Descargado de <https://www.sciencedirect.com/science/article/pii/S0164121221001849> doi: <https://doi.org/10.1016/j.jss.2021.111087>
- Krikava, F. (2013, 11). *Domain-specific modeling language for self-adaptive software system architectures*.
- Lalanda, P., Diaconescu, A., y McCann, J. A. (2014). *Autonomic computing: Principles, design and implementation*. Springer.
- Liu, H., Parashar, M., y Hariri, S. (2004). A component-based programming model for autonomic applications. En *International conference on autonomic computing, 2004. proceedings*. (p. 10-17). doi: 10.1109/ICAC.2004.1301341
- Min-Allah, N., y Alrashed, S. (2020, agosto). Smart campus—a sketch. *Sustainable Cities and Society*, 59, 102231. Descargado de <https://doi.org/10.1016/j.scs.2020.102231> doi: 10.1016/j.scs.2020.102231
- Mozilla Foundation. (2023). *State machine*. Online. Descargado 2023-05-20, de [https://developer.mozilla.org/en-US/docs/Glossary/State\\_machine](https://developer.mozilla.org/en-US/docs/Glossary/State_machine)
- Object Management Group. (2015). *What is sysml?* Online. Descargado de <https://www.omg-sysml.org/what-is-sysml.htm>



- Ouareth, S., Boulehouache, S., y Mazouzi, S. (2018). A component-based mape-k control loop model for self-adaptation. En *2018 3rd international conference on pattern analysis and intelligent systems (pais)* (p. 1-7). doi: 10.1109/PAIS.2018.8598529
- Patouni, E., y Alonistioti, N. (2006). A framework for the deployment of self-managing and self-configuring components in autonomic environments. En *2006 international symposium on a world of wireless, mobile and multimedia networks(wowmom'06)* (p. 5 pp.-484). doi: 10.1109/WOWMOM.2006.11
- Red Hat Foundation. (2023). *What is yaml?* Descargado 2023-5-27, de <https://www.redhat.com/en/topics/automation/what-is-yaml>
- Rutanen, K. (2018). Minimal characterization of o-notation in algorithm analysis. *Theoretical computer science*, 713, 31–41.
- Salehie, M., y Tahvildari, L. (2005, 01). Autonomic computing: emerging trends and open problems. *ACM SIGSOFT Software Engineering Notes*, 30, 1-7.
- Tahir, M., Mamoon Ashraf, Q., y Dabbagh, M. (2019). Towards enabling autonomic computing in iot ecosystem. En *2019 ieee intl conf on dependable, autonomic and secure computing, intl conf on pervasive intelligence and computing, intl conf on cloud and big data computing, intl conf on cyber science and technology congress (dasc/picom/cbdcom/cyberscitech)* (p. 646-651). doi: 10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00122
- Wang, Z., Sun, C.-a., y Aiello, M. (2021). Lightweight and context-aware modeling of microservice-based internet of things. En *2021 ieee international conference on web services (icws)* (p. 282-292). doi: 10.1109/ICWS53863.2021.00046
- Weiss, G., Zeller, M., y Eilers, D. (2011). *Towards automotive embedded systems with self-x properties*. Fraunhofer-Gesellschaft. Descargado de <https://publica.fraunhofer.de/handle/publica/224379> doi: 10.24406/PUBLICA-FHG-224379

# Apéndice A

## Ejemplo de un YAML de una aplicación válida

```
locations:
  name: "Universidad Industrial de Santander"
  locations:
    campus-central:
      name: "Campus Central"
      locations:
        laboratorios-pesados:
          name: "Laboratorios Pesados"
          ip: '192.168.0.2'
        mecanica:
          name: "Mecanica"
          ip: '192.168.0.3'
        camilo-torres:
          name: "Camilo Torres"
          locations:
            ct-primero:
              name: "Primer Piso"
              ip: '192.168.0.6'
            ct-segundo:
              name: "Segundo Piso"
              ip: '192.168.0.5'

  malaga:
    name: "Sede Malaga"
    locations:
      reconver-gan:
        name: "Laboratorio De Reconversion Ganadera"
        ip: '192.168.0.127'
      admon-malaga:
        name: "Administracion Sede Malaga"
        ip: '192.168.0.126'

components:
```

```

dol139:
  name: "DOL 139 Environmental Monitor"
  component-type: "sensor"
  locations: # Where context
    malaga: # Requirements
      quantity: 3
      timeout: 15
      required: true
    campus-central:
      quantity: 3
      timeout: 15
      required: true
    camilo-torres:
      quantity: 1
      timeout: 15
      required: false
  components: # What composes it
    temperature-sensor:
      name: "Temperature Sensor"
      component-type: "sensor"
      required: true
      outputs:
        temperature: "float"
    co2-sensor:
      name: "CO2 Sensor"
      component-type: "sensor"
      required: true
      outputs:
        ppm: "float"
    humidity-sensor:
      name: "Humidity Sensor"
      component-type: "sensor"
      required: true
      outputs:
        RH: "float"
pm: # General component
  name: "Particulate Matter Monitor"
  component-type: "sensor"
  locations: # where context
    mecanica:
      quantity: 1
      timeout: 60
      required: false
    admon-malaga:
      quantity: 2
      timeout: 60
      required: true
  components: # what composes it
    pm25:
      name: "PM2.5 Sensor"
      component-type: "sensor"
      required: false
      outputs:
        ppm: "float"

    pm10:
      name: "PM10 Sensor"
      component-type: "sensor"
      required: false

```

```
        outputs:
          ppm: "float"
barometric: # Atomic component
  name: "Barometric Pressure Sensor"
  locations:
    campus-central:
      quantity: 1
      required: true
      timeout: 3600
    malaga:
      quantity: 1
      required: true
      timeout: 3600
  component-type: "sensor"
  outputs:
    atm: "float"
```