



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Middleware para la monitorización de la calidad de servicios cloud

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Jiménez Gómez

Tutor: Silvia Abrahao Gonzales

2014-2015

Agradecimientos

Este trabajo ha sido posible gracias en primer lugar al apoyo de mi familia, compañeros y amigos.

Especial agradecimiento a Priscila por toda la ayuda prestada a lo largo de este año y a Silvia, Emilio, Javier y los chicos del laboratorio por su apoyo.



Resumen

La computación en la nube es una nueva corriente en la forma de desarrollar y usar software. El uso de servicios alojados en la nube utilizando el modelo de Software as a Service (SaaS) está siendo adoptado por multitud de organizaciones. Estos servicios suelen ir acompañados de un documento de acuerdo de nivel de servicio (*Level Service Agreement* o SLA) el cual define las características de calidad que el proveedor del servicio ofrece a los clientes. Es por ello necesario el uso de sistemas de monitorización que se encarguen de extraer los datos necesarios para poder controlar el cumplimiento de dichas características de manera que ofrezcan información acerca de los atributos funcionales y no funcionales del servicio. Esta información ayudará a los *stakeholders* a ser capaces de tomar decisiones para mejorar la calidad del servicio.

En este trabajo se propone un enfoque de monitorización de servicios definiendo un componente Middleware encargado de dicha tarea. Para comprobar el pragmatismo de este acercamiento se ha realizado una implementación orientada a la plataforma en la nube Microsoft Azure©.

Además este trabajo se encuentra contextualizado por el proyecto “Desarrollo Incremental de Servicios Cloud Dirigido por Modelos y Orientado al Valor del Cliente (Value@Cloud)” realizado por el grupo de investigación de Ingeniería del Software y Sistemas de Información (ISSI) del departamento de Sistemas Informáticos y de Computación de la Universidad Politécnica de Valencia. En particular, el trabajo realizado forma parte del paquete de trabajo “Definición y gestión de mecanismos de monitorización del valor en tiempo de ejecución”. Este proyecto tiene por objetivo investigar cómo los equipos de desarrollo de software pueden identificar, crear, desplegar y monitorizar servicios cloud de forma incremental, haciendo uso de los principios del desarrollo de software ágil, en el dirigido por modelos y en la mejora de proceso de negocio.

Palabras clave: Computación en la nube, Modelos en tiempo de ejecución, Calidad de Servicios, Monitorización, Microsoft Azure.

Abstract

Cloud computing is a new trend in the way of developing and using software. The use of services hosted in the cloud by employing the Software as a Service (SaaS) model is currently being employed by many organizations. These services are usually accompanied by a Level Service Agreement (SLA) document which define the quality characteristics that a service provider offers to its customers. Consequently, the use of monitoring systems for extracting the necessary data for assessing the fulfillment of cloud services quality requirements is required. This information will help stakeholders to make decisions regarding how to improve the service's quality.

In this work, we propose an approach based on models at run-time for monitoring the quality of cloud services by defining a middleware to support this task. To demonstrate the feasibility of this approach, we show the instantiation of the proposed middleware for monitoring services deployed on the Microsoft Azure© cloud platform.

This work is supported by the Value@Cloud (Model-Driven Incremental Development of Cloud Services Oriented to the Customer's Value) research project conducted by the ISSI research group from the Department of Information Systems and Computation (DSIC) at the Universitat Politècnica de València. Specifically, this work contributes to the “definition and management of monitoring mechanisms at run time” working package. The objective of this project is to investigate how software development teams can identify, create, deploy and monitor cloud services incrementally by using the principles of agile, model-driven software development and business process improvement.

Keywords: Cloud computing, Models at runtime, Services quality, Monitoring, Microsoft Azure.

Tabla de contenidos

Agradecimientos	3
Listado de Figuras	8
1. Introducción	11
1.1 Motivación	11
1.2 Objetivos	13
1.3 Contexto	13
1.4 Estructura del documento.....	14
2. Análisis del estado del arte.....	15
2.1 Introducción a la monitorización de servicios en la nube	15
2.1.1 Tipos de datos	15
2.2 Monitorización en sistemas distribuidos tradicionales.....	16
2.3 Herramientas actuales de monitorización en la nube	16
2.3.1 Monitorización basada en agentes	17
2.3.2 Monitoring as a Service (MaaS)	18
2.3.3 Monitorización del SLA.....	19
2.3.4 Comparación de herramientas de monitorización.....	20
2.3.5 Conclusiones	21
3. Cloud computing	22
3.1 Definición de computación en la nube.....	22
3.2 Ventajas e inconvenientes del cloud computing	23
3.3 Clasificación de servicios cloud.....	26
3.3.1 Modelos de despliegue.....	26
3.3.2 Modelos de servicio	26
3.4 El Acuerdo de Nivel de Servicio.....	27
4. Arquitectura de Monitorización basada en Modelos en Tiempo de Ejecución	30
4.1 Presentación de la Arquitectura de Monitorización basada en Modelos en Tiempo de Ejecución.....	30
4.1.1 Empleo de los modelos en tiempo de ejecución	31
4.2 El proceso de monitorización.....	31
4.2.1 Configuración de la Monitorización	32
4.2.2 Proceso del Middleware.....	36
4.2.3 Análisis de resultados.....	36

4.3 Componentes del Proceso de Monitorización.....	36
4.3.1 El Configurador del Middleware	37
4.3.2 El Motor de Middleware y Análisis	37
4.3.2 Mecanismos de recolección de datos de la plataforma	38
5. Definición del Middleware de Monitorización	41
5.1 Introducción a las herramientas	41
5.2 Microsoft Azure	42
5.3 El Configurador de la Monitorización	45
5.3.1 Patrón de diseño	45
5.3.2 Selección de la plataforma	47
5.3.3 Selección del modelo de requisitos de monitorización	48
5.3.4 Clasificación de requisitos no funcionales	49
5.3.5 Constructor de fórmulas.....	51
5.4 El Middleware de Monitorización	53
5.4.1 Requisitos y características del middleware de monitorización.....	54
5.4.2 Arquitectura	54
5.4.3 Descripción del funcionamiento	57
6. Caso de estudio	59
6.1 Presentación del caso	59
6.2 Configuración de la monitorización.....	60
6.2.1 Introducción de los datos de la plataforma.....	60
6.2.2 Introducir el modelo de requisitos.....	61
6.2.3 Asociación de métricas	61
6.2.4 Construcción de fórmulas	63
6.3 El Middleware de Monitorización.....	65
7. Conclusiones y trabajos futuros	66
7.1 Conclusiones	66
7.2 Tecnologías	67
7.2.1. El Uso de Modelos en tiempo de ejecución	67
7.2.2 XML y XMI	67
7.2.3 Microsoft Azure	68
7.3 Herramientas finalizadas	68
7.3.1 El configurador de la monitorización.....	68
7.3.2 El Middleware de Monitorización	69
7.4 Trabajos futuros	69
7.5 Publicaciones académicas y otros resultados	70



8. Referencias.....	71
Anexo.....	74
1. Modelos empleados.....	74
1.1 Modelo de Calidad SaaS.....	74
1.2 Modelo de Requisitos de Monitorización.....	74
1.3 Modelo de Monitorización en Tiempo de Ejecución.....	75
2. Modelo en tiempo de ejecución en XML.....	75

Listado de Figuras

Figura 1 – El Proceso de Monitorización.....	32
Figura 2 – La Configuración de la Monitorización.....	33
Figura 3 – Componentes del Proceso de Monitorización.....	37
Figura 4 – Escenarios de Extracción de Datos.....	38
Figura 5 – Prototipo del Configurador de la Monitorización.....	46
Figura 6 – Ejemplo de interfaz con el patrón Asistente.....	47
Figura 7 – Flujo de la aplicación web.....	47
Figura 8 – Selección de la Plataforma.....	48
Figura 9 – Selección del modelo de requisitos de monitorización.....	48
Figura 10 – Modelo de requisitos de la monitorización cargado en la aplicación.....	49
Figura 11 – Asociación de Métricas: Los RNFs.....	50
Figura 12 – Asociación de Métricas: Clasificación en el modelo de calidad.....	50
Figura 13 – Asociación de métricas: El modelo en tiempo de ejecución.....	51
Figura 14 – Constructor de fórmulas.....	52
Figura 15 – Constructor de fórmulas: Resultado en el modelo en tiempo de ejecución.....	53
Figura 16 – Arquitectura del Middleware de Monitorización.....	54
Figura 17 – Ejemplo de operacionalización representada en XML.....	58
Figura 18 – Caso de Estudio: Selección de Plataforma.....	60
Figura 19 – Caso de Estudio: El modelo de especificación de requisitos.....	61
Figura 20 – Caso de Estudio: Clasificación de la confiabilidad.....	62
Figura 21 – Caso de Estudio: Clasificación de la latencia.....	62
Figura 22 – Caso de estudio: Métricas añadidas al modelo en tiempo de ejecución.....	63
Figura 23 – Caso de Estudio: Construcción de la operacionalización de la confiabilidad....	63
Figura 24 – Caso de Estudio: Construcción de la operacionalización de la latencia.....	64
Figura 25 – Caso de Estudio: Resultado de la configuración.....	64
Figura 26 – Anexo: Modelo de Calidad SaaS.....	74
Figura 27 – Anexo: Modelo de Requisitos de Monitorización.....	74
Figura 28 – Anexo: Modelo de Monitorización en Tiempo de Ejecución.....	75

1. Introducción

Este documento tiene a fin presentar el trabajo de investigación y desarrollo realizado para materializar un Middleware para la monitorización de servicios en la nube. Para ello, a lo largo del documento se expondrán los conceptos más relevantes para el proyecto, el desarrollo y resultado final del trabajo.

1.1 Motivación

Los servicios en la nube representan una nueva corriente en el desarrollo y uso de sistemas software. Se trata de una tecnología relativamente nueva que en la actualidad está cobrando importancia capital y que prevé una rápida expansión en el futuro próximo, siendo una de las principales áreas de interés de las empresas tecnológicas, junto con la seguridad y los dispositivos móviles [1]. Actualmente las plataformas en la nube, agrupan sus servicios en tipos de provisión: Infraestructura como Servicio (*Infrastructure as a Service, IaaS*), Plataforma como Servicio (*Platform as a Service, PaaS*), o Software como un Servicio (*Software as a Service, SaaS*).

El modelo de negocio que ofrece la computación en la nube tiene un gran número de ventajas tanto para proveedores como para consumidores. Citando solo algunas ventajas, este tipo de servicios son fácilmente escalables, ofrecen alta disponibilidad, elasticidad, un modelo de pago por uso, máximo aprovechamiento de los recursos hardware, etc. Estas ventajas se han traducido en requisitos de calidad que deben ser cumplidos durante el tiempo en el cual se provee el servicio.

La provisión de SaaS, tipo de servicio en el que se centrará el presente trabajo, incluye la entrega de aplicaciones software o servicios a clientes a través de internet. Este tipo de provisión de servicio es el que más crecimiento ha mostrado y Cisco prevé que en 2018 representará casi el 60% del total de carga de trabajo en la nube [2].

Un servicio SaaS, tiene un número variable de usuarios en ocasiones con picos de tráfico muy intensos (por ejemplo durante la campaña navideña en un servicio de cobro en línea) y por ello la infraestructura SaaS debe poder soportar millones de usuarios con un rendimiento escalable [2] que pueda responder a esta variabilidad, adaptando sus recursos a las necesidades del momento. Para determinar el nivel de cumplimiento de los atributos de calidad de un servicio, los términos en los que un servicio debe de estar provisto deben de estar expresados usando un documento de Acuerdo de Nivel de Servicio (*Service Level Agreement, SLA*).

Por tanto, cada servicio suministrado irá acompañado de un documento de este tipo. El SLA define las garantías mínimas que un proveedor de servicios ofrece a sus clientes [3]. Normalmente el proveedor de servicios cloud ofrece algunas garantías de rendimiento para servicios de cómputo y deja la detección de las violaciones del SLA en las manos del cliente. Además, pocos proveedores cloud automáticamente recompensan al cliente por violaciones en el SLA por lo que es necesario que el cliente provea las evidencias de las violaciones [4].

De esta manera, es útil tanto para el cliente como para el proveedor de servicios que exista algún tipo de herramienta que permita conocer automáticamente el estado del comportamiento real de los servicios y si estos cumplen el SLA acordado, y si no lo hacen puedan aportar evidencia para la reclamación. Mientras que el proveedor de servicios está interesado en

incrementar la confianza de los clientes aportando datos objetivos sobre el nivel de provisión de sus servicios.

Por todo ello es necesario el uso de herramientas de monitorización que permitan medir este tipo de atributos de calidad, ayudando a la generación de informes fiables del estado del nivel de provisión del servicio.

El desarrollo de estas herramientas de monitorización supone un reto dado que el despliegue y ejecución de sistemas software en infraestructuras altamente dinámicas introduce un nuevo conjunto de desafíos y requisitos en lo que respecta a la monitorización. En consecuencia, una aplicación de monitorización debe de poseer flexibilidad para adaptarse a cualquier cambio en los requisitos de monitorización y mantenerse según el servicio escale hacia arriba o hacia abajo dinámicamente [5].

El desarrollo de software dirigido por modelos (*Model Driven Engineering*) puede ser una solución para aportar la flexibilidad requerida a las herramientas de monitorización. Sin embargo, establecer todo el conjunto de requisitos de monitorización presentes en el SLA de forma estática (en tiempo de diseño) no siempre es posible debido a que el SLA está abierto a cambios y adición de nuevos requisitos no funcionales (RNFs) a ser monitorizados, incluyendo los aportados por nuevas negociaciones entre las partes [6]. Baresi y Ghezzi [7] apuestan por un futuro de la Ingeniería del Software en la que la investigación se enfoque a proveer soporte inteligente al software en tiempo de ejecución, rompiendo la rígida barrera entre el tiempo de diseño y el tiempo de ejecución. Por ello, es necesario definir aproximaciones que soporten la monitorización en tiempo de ejecución de servicios en la nube y a su vez permitan la adición de nuevos requisitos sin la interrupción de los servicios en ejecución. Este desafío puede resolverse utilizando la aproximación de modelos en tiempo de ejecución (*models@run.time*) [8]. Estos modelos tienen el potencial de ser utilizados en tiempo de ejecución para supervisar y verificar aspectos particulares del comportamiento de los servicios cloud en tiempo de ejecución.

La tecnología de *models@run.time* especifica los datos que deberían ser recogidos por una herramienta de monitorización, por lo que este está basado en los RNFs especificados por el SLA y otros requisitos adicionales. Este modelo estará producido de acorde a un modelo de calidad SaaS que recogerá todas las características, subcaracterísticas, atributos y métricas que permitirán medir los atributos de calidad de los servicios en la nube (seguridad, elasticidad, fiabilidad, etc.).

Los modelos de calidad describen las características de calidad de un producto software, sus relaciones, como estas características pueden ser medidas y como las mediciones pueden ser interpretadas. Los principales problemas asociados a la calidad de un servicio cloud son: la definición de un conjunto de propiedades en un servicio que indique su calidad; la medición del grado de cumplimiento; y la utilización de la información disponible acerca del servicio para mejorar su calidad a lo largo del ciclo de vida.

Utilizando este modelo en el proceso de monitorización, este proveerá un informe de violaciones de los requisitos no funcionales los cuales contienen los requisitos presentados en el SLA y otros requisitos no funcionales adicionales. Este informe ayudará a los *stakeholders* a identificar las características de calidad que están presentando problemas en el servicio para identificar posibles errores y con esa información mejorar la calidad del servicio.

El cálculo de las métricas que permitan medir los atributos de calidad utilizados en estos informes necesitará de datos procedentes del servicio en la nube, que una vez tratados correspondientemente podrán ofrecer una visión de alto nivel del estado del servicio. Es por ello necesaria la elaboración de una infraestructura de monitorización que sea capaz de incorporar la tecnología de *models@runtime*, extraer los datos crudos procedentes del servicio, calcular las métricas asociadas a NFRs expresados en el SLA y almacenarlas para un posterior tratamiento de los datos que permita generar informes útiles para los stakeholders.

Cedillo et al [9] han propuesto un proceso de monitorización para servicios en la nube basado en estos principios que necesita de la implementación de herramientas para verificar y refinar el proceso.

1.2 Objetivos

El propósito de este trabajo es crear un middleware para la monitorización de la calidad de servicios cloud (SaaS) que soporte la infraestructura de monitorización cuyo proceso se presenta en [9] y con ello poner en práctica y refinar el proceso.

Se han tratado de conseguir los siguientes objetivos:

- La creación de una herramienta de monitorización capaz de extraer datos de servicios en la nube situados en esa plataforma en base a las necesidades establecidas en un SLA y estructuradas en un modelo en tiempo de ejecución que permitan realizar los cálculos de las métricas establecidas en el modelo, de manera que su análisis pueda certificar el cumplimiento o no de los requisitos no funcionales del SLA.
- La instanciación de esta herramienta en la plataforma cloud Microsoft Azure, investigando las herramientas que permiten la extracción de datos propios de la plataforma y trabajando con las herramientas disponibles.
- La construcción de una herramienta que apoye la creación de un modelo en tiempo de ejecución que recoja los datos del SLA necesarios para establecer los requisitos no funcionales a monitorizar y clasificar estos requisitos utilizando un modelo de calidad SaaS e indicar en base a él los métodos de medición de esos requisitos.
- La definición de un método para la obtención de datos de monitorización de otras plataformas y su integración en el cálculo de métricas.

1.3 Contexto

Este proyecto forma parte del proyecto de investigación “Desarrollo Incremental de Servicios Cloud Dirigido por Modelos y Orientado al Valor del Cliente (Value@Cloud)” realizado por el grupo de investigación de Ingeniería del Software y Sistemas de Información (ISSI) del Departamento de Sistemas Informáticos y de Computación de la Universitat Politècnica de Valencia. En particular, el trabajo desarrollado forma parte del paquete de trabajo “Definición y gestión de mecanismos de monitorización del valor en tiempo de ejecución”.

Por ello, este proyecto se integra dentro de los flujos de trabajo del proyecto. Recibiendo como entradas los resultados de otras tareas a las cuáles el proyecto debe adaptarse y a su vez las salidas del mismo deben de ser consumidas por otros artefactos.

Este proyecto tiene por objetivo investigar cómo los equipos de desarrollo de software pueden identificar, crear, desplegar y monitorizar servicios cloud de forma incremental, haciendo uso de los principios del desarrollo de software ágil, en el dirigido por modelos y en la mejora de proceso de negocio.

Este trabajo también está soportado por el siguiente “Microsoft Azure for Research Award Program”:

- *Microsoft Azure Research Award from Microsoft Research. Project: “Model-Driven Incremental Development of Cloud Services”. The estimated market value of the award is USD \$ 40,000 per year. July 2014 to July 2016.*

1.4 Estructura del documento

En el primer capítulo se han presentado la motivación, los objetivos que se pretenden alcanzar y el contexto en el que se ubica este Trabajo de Fin de Grado. En los siguientes capítulos se estructura el trabajo de la siguiente forma:

En el Capítulo 2 se presenta el estado del arte acerca de los métodos relacionados con este trabajo: la evolución de las técnicas de monitorización, la monitorización de servicios en la nube y los distintos enfoques así como una comparación entre las herramientas comerciales encontradas en la literatura.

El Capítulo 3 introduce los conceptos principales del *cloud computing* ofreciendo el contexto tecnológico a este trabajo, se define el concepto de *cloud computing*, se presenta la clasificación de los servicios aportados por él y se presentan los acuerdos de nivel de servicio que definen las relaciones entre consumidores de un servicio y los proveedores.

El Capítulo 4 presenta el proceso de monitorización utilizado exponiendo sus distintas fases, tareas y artefactos.

El Capítulo 5 presenta la estructura del Middleware de Monitorización independiente de la plataforma que se propone en este trabajo, presentando sus componentes, definiendo sus funciones y la relación entre ellos.

El Capítulo 6 expone la instanciación del Middleware propuesto a una plataforma *cloud* específica (Microsoft Azure). Primero se presenta el configurador de la monitorización y después el Middleware de monitorización, presentando su estructura, diseño, conceptos relevantes a la hora de su implementación y su funcionamiento.

Por último, el Capítulo 7 describe las conclusiones generales, los problemas enfrentados en el desarrollo, los trabajos futuros y los resultados obtenidos como publicaciones.

2. Análisis del estado del arte

En esta sección se expondrán las tecnologías de monitorización halladas en la literatura. Primero se procederá a introducir la monitorización de servicios en la nube. A continuación, se pondrá en contexto la monitorización de servicios en la nube a través de sus orígenes en la monitorización de sistemas distribuidos. Tras ello se procederá a exponer los distintos acercamientos actuales encontrados en la literatura al problema, describiendo sus diferencias y tendencias. Por último se procederá a hacer una comparación entre las distintas herramientas de monitorización y se detallaran sus virtudes y carencias.

2.1 Introducción a la monitorización de servicios en la nube

La considerable expansión del uso de aplicaciones ubicadas en la nube ha aumentado el interés acerca de las tecnologías encargadas de supervisar su funcionamiento. Las posibilidades que ofrecen los servicios en la nube pueden resultar aplicaciones con una arquitectura muy compleja, muchas veces una sola aplicación puede depender de varios servicios muchas veces confluyendo diversas tecnologías y proveedores y la mayoría de las veces separadas físicamente. Para administrar estos servicios y hacer posible que cumplan con los términos de acuerdo de servicio (SLA) es vital obtener información de rendimiento y calidad de los servicios desplegados.

Según Fatema et al [10] la monitorización de servicios en la nube se define como: *“A process that fully and precisely identifies the root cause of an event by capturing the correct information at the right time and at the lowest cost in order to determine the state of a system and to surface the status in a timely and meaningful manner”*. Un proceso que complete y precisamente identifica la causa base de un evento mediante la captura de la información correcta en el momento correcto y con el menor coste para determinar el estado de un sistema y supervisar su estatus en una forma oportuna y significativa.

Estas herramientas son vitales para proveedores y clientes de servicios en la nube ya que los métodos de pago por uso necesitan de datos precisos de rendimiento para poder realizar la facturación correcta. La monitorización es una parte necesaria para los procesos de provisión de recursos/servicios, planificación óptima de la calidad, comprobación de SLAs, gestión de la configuración, facturación y seguridad y privacidad.

Para este cometido una gran variedad de herramientas y técnicas están disponibles. Estas utilizan distintos tipos de acercamiento a la monitorización. Estas herramientas pueden clasificarse según el tipo de datos que extraen y de cómo los obtienen.

2.1.1 Tipos de datos

Por lo general podemos clasificar los datos de un servicio cloud en tres capas, relacionadas con las tres formas de provisión de servicios cloud: IaaS, PaaS y SaaS. Un monitor que actúe sobre la del primer nivel (IaaS) obtendrá datos relacionados con la infraestructura. Por ejemplo un monitor ejecutado en la infraestructura ofrecida por Amazon EC2 obtendrá datos del nivel más bajo: uso de disco, datos sobre el estado de la red, uso de memoria, consumo de CPU...

En el nivel de PaaS los datos obtenidos por la herramienta de monitorización serán los referentes al entorno de máquinas virtuales sobre los que se ejecuta el servicio (sistema operativo y otras herramientas). Este es el caso de *Diagnostics*, la herramienta de

monitorización propia de Microsoft Azure que permite obtener datos del sistema operativo Windows sobre el que trabajan las herramientas como el tiempo de ejecución de una petición.

Los datos de SaaS por otra parte serán los referentes a la implementación concreta del servicio y variarán dependiendo del dominio del servicio y la tecnología con la que esté implementado. Por ejemplo un servicio que gestiona una página de comercio electrónico o un servicio de gestión de pagos aportarán datos diferentes, en uno podremos obtener datos acerca del tiempo medio que un ítem pasa en el carrito hasta que es comprado y en otro el número de pagos realizados con una determinada sucursal en un día.

2.2 Monitorización en sistemas distribuidos tradicionales

El germen común de los inicios de monitorización de servicios en la nube, se tienen también en las tecnologías que marcaron los inicios de la nube, esto son los sistemas distribuidos de *clusters* de cómputo y *grids*. De estas tecnologías emergieron también las herramientas de monitorización que se aplicarían a la nube.

La computación en la nube se diferencia de la de los *clusters* en que el sistema es considerado como una unidad y la tarea a desarrollar es la misma en cada nodo de los *clusters*. Esto no es así en la computación en la nube donde en cada nodo diferentes máquinas virtuales comparten recursos cada una pudiendo servir a un propósito diferente. La comparación con el *grid* es más compleja dado que comparten varios puntos en común y de hecho la infraestructura de la computación en la nube puede verse como un *grid*. Según Hassan [11] la diferencia radica en los servicios ofrecidos por ambas plataformas, mientras que los *grids* están diseñados para ser de propósito general y las interfaces se mantienen a un nivel más bajo, la nube ofrece un conjunto más limitado de características pero a un nivel más alto ofreciendo servicios y recursos más abstractos (como por ejemplo un sistema operativo).

Aunque las características de este tipo de sistemas no sean idénticas a las condiciones de la nube, tanto como por requisitos a ser monitorizados como por las características de la tecnología que los constituyen, algunos de los sistemas y técnicas de monitorización han sido adaptadas para utilizarse en la nube y han surgido otras específicas para este tipo de entornos.

Por ejemplo la herramienta de monitorización Nimsoft originalmente era utilizada para extraer datos relacionados con la infraestructura (red, servidores, bases de datos...) pero después evolucionó para ofrecer sus servicios como un Monitor de Nube Unificado para monitorizar servicios alojados externamente.

2.3 Herramientas actuales de monitorización en la nube

Actualmente existen numerosas herramientas académicas y comerciales que cumplen con las funciones de monitorización de servicios. Algunas ofrecen solamente medios de extracción de datos del servicio monitorizado mientras que otras ofrecen soluciones integrales que se ocupan desde la extracción de datos hasta la generación de informes y su visualización.

Según la revisión realizada en 2014 por Kaniz Fatema et al [10], se pueden clasificar estas herramientas según el alcance del tipo de datos que pueden obtener del servicio a ser monitorizado, tal y como se ha comentado la sección 2.1 de este capítulo. Muchas de las herramientas de monitorización de servicios en la nube como Amazon Cloud Watch [12], Azure Watch, Nimsoft [35] y Monitis [36] son capaces de monitorizar tanto al nivel de la infraestructura como de la aplicación. Sin embargo, otras herramientas solo son capaces de monitorizar o bien al nivel de la infraestructura, por ejemplo Cloud Kick [17] y PCMONS [37];

y otras al nivel de la plataforma, como por ejemplo Boundary Application Monitor [38]y Cloud Application SLA Violation Detection(CASViD) [18] .

La gran variedad de tecnologías y plataformas proveedoras de servicios cloud ha tenido como consecuencia que las herramientas se especialicen en una plataforma en concreto y es difícil encontrar una solución multiplataforma. Por una parte, la necesidad de los proveedores de ofrecer un servicio de confianza y calidad a los consumidores ha producido que por su parte los proveedores de servicios en la nube ofrezcan herramientas de monitorización propias especializadas en las características de sus plataformas. Es el caso de Amazon Cloud Watch el monitor de los Amazon Web Services en Amazon EC2, incluyendo instancias de servicios y las diferentes bases de datos ofertadas por Amazon[12]; Azure Diagnostiscs [39] en el caso de Microsoft Azure y sus servicios, instancias de servicios de Windows Azure, bases de datos SQL de Azure y almacenamiento Azure; o Google Cloud Monitoring la herramienta de Google Cloud Platform que permite la monitorización de servicios en la nube, máquinas virtuales y otros servidores de código abierto como Apache, MongoDB, Nginx... [13]

En las siguientes subsecciones se llevará a cabo una revisión de las herramientas, técnicas y tendencias más populares, concluyendo con una comparación de las herramientas más populares y una exposición de los puntos débiles de estas técnicas.

2.3.1 Monitorización basada en agentes

Con respecto a las técnicas utilizadas por estas herramientas para la recuperación de información la más común, presente en propuestas académicas como Varanus [14] y en la popular Nagios [15]. La monitorización por agentes es típica de la monitorización multipropósito.

Este tipo de monitorización consiste en instalar una pieza de software denominada agente en cada servicio a ser monitorizado. Cada agente se encarga de recuperar datos del servicio en el que se ejecutan y, dependiendo de la solución, procesarlos construyendo métricas de más alto nivel. Cada uno de los agentes se encarga de que periódicamente esos datos propios de cada servicio sean enviados a un segundo componente software llamado servidor de monitorización cuya tarea principal es recuperar los datos recolectados por los agentes, tratarlos si fuese necesario y almacenarlos para su análisis posterior. Este servidor suele encargarse también del ciclo de vida de los agentes, detectando agentes desconectados y regulando el comportamiento de estos según el estado en el que se encuentre el sistema (por ejemplo reduciendo la actividad de los agentes en momentos de elevada carga) y además de lanzar alertas si fuese necesario.

Con el propósito de mejorar el rendimiento y características de calidad como la disponibilidad, se han establecido diferentes propuestas que varían la topología de la monitorización. Por ejemplo, para evitar que la monitorización se vea comprometida por un fallo del servidor del nodo que actúa como servidor de monitorización se han propuesto sistemas de comunicación en los que el rol de servidor de la monitorización puede ser adoptado por cualquiera de los agentes llegado el caso.

También con el propósito de mejorar la eficiencia de la monitorización Meng et al. [15] proponen el uso de árboles de agentes que puedan desarrollarse o ser podados de forma dinámica según el servicio varía en su aprovisionamiento según las condiciones de elasticidad así lo requieran.

La elasticidad es una de las propiedades más interesantes de la computación en la nube y a su vez es la que más difícil hace el proceso de monitorización, sobre todo en los casos que la monitorización es llevada a cabo por agentes. Dado que un agente es asociado a un recurso virtual como una máquina virtual que provee un servicio, cuando estas instancias se duplican no solo es un problema la identificación de cada una de las máquinas, sino que la infraestructura de monitorización debe adaptarse (lanzando más agentes o aumentando la carga de agentes libres con las tareas asociadas a esas nuevas máquinas virtuales). A su vez es complicado establecer que recursos físicos está empleando cada máquina virtual y es necesario establecer correlaciones entre los datos aportados por distintas máquinas virtuales. El problema de la elasticidad y la monitorización es tratado por Moldovan et al. [16] ofreciendo distintas soluciones a nivel de arquitectura y de funciones de cálculo sobre estos datos.

2.3.2 Monitoring as a Service (MaaS)

Una tendencia que se hace patente en las últimas publicaciones sobre la materia es ofrecer los servicios de monitorización como un servicio en la nube. La mayoría de herramientas de monitorización ofrecen sus servicios como SaaS, es decir el servicio es alojado y mantenido en una plataforma externa a la del servicio que se quiere monitorizar y un tercero contrata esos servicios de monitorización.

La provisión de un servicio de monitorización de esta manera supone grandes ventajas. Meng et al. [15] son expuestas algunas:

Primero, este tipo de provisión de servicios de monitorización ayuda a minimizar el coste de propiedad de estas herramienta permitiendo obtener los mismos resultados de forma más barata ya que el consumidor de estos servicios ya no tendrá que hacerse cargo de los costes de sobre trabajo que generan este tipo de aplicaciones a la hora de monitorizar los servicios, evitando así costes extras de almacenamiento y procesamiento y la sobrecarga en instantes de alta demanda.

Además este modelo permite beneficiarse de las actualizaciones centralizadas de la plataforma de monitorización, proveyendo software de mayor calidad de manera constante.

También este sistema hace más sencillo para los propietarios de los servicios desplegar monitores a distintos niveles de los servicios en la nube en comparación con la obligación de crear cada uno de estos monitores específicos para tu servicio.

Además de ello, MaaS se beneficia de las ventajas del modelo de negocio de SaaS, de manera que permite el modelo de facturación pago por uso (*pay-per-use*). De esta forma, los costes iniciales necesarios para el uso de estas herramientas sería más bajo que en el formato tradicional, ajustándose así a proyectos de menor envergadura con presupuestos reducidos que podrán beneficiarse de este tipo de herramientas.

Este modelo también permite a los proveedores de servicio consolidar las demandas de monitorización a distintos niveles para conseguir una monitorización eficiente y escalable. Teniendo la capacidad de ceder más recursos según sea necesario para la monitorización de servicios que hayan crecido debido a la alta demanda y así evitando que los servicios de monitorización acaparen más tiempo de procesador del necesario en momentos críticos de la provisión del servicio.

Por último, MaaS incita a los proveedores a invertir en nuevas herramientas de monitorización y ofrecer siempre un servicio de monitorización de la más alta calidad.

Sin embargo, las ventajas ofrecidas por MaaS tienen su contrapartida en la dificultad de implementación. El uso de servicios MaaS es problemático cuando el acercamiento que se usa para la extracción de datos de la monitorización es el de agentes. Esta técnica hace necesaria que los servicios en la nube a ser monitorizados sean capaces de soportar la instalación y ejecución de los agentes de los servicios SaaS encargados de extraer y transmitir los datos del servicio al núcleo de la monitorización. Por ello, estos agentes suelen estar disponibles en código abierto en el lado del cliente para facilitar la incorporación a nuevos servicios. Por ejemplo CloudKick[17] ofrece sus agentes en C#, .Net, Python, Java y Ruby.

Estos servicios además suelen proveer funciones extra como un sistema de alertas que avisa a los clientes de incidencias con el servicio por medio de mensajes SMS, correos electrónicos y otros medios de comunicación. Así como herramientas de visualización del estado de los servicios monitorizados.

2.3.3 Monitorización del SLA

Uno de los objetivos de la monitorización de servicios en la nube consiste en vigilar el cumplimiento del Acuerdo de Nivel de Servicio (*Software Level Agreement*, SLA). Un SLA es un contrato estricto entre un proveedor de un servicio y su cliente en el cual se fijan los términos referentes a la calidad de la provisión del servicio. En él, se establecen y consensan los términos de nivel de calidad de servicio de todos los requisitos no funcionales (RNFs), por ejemplo, garantizar que la latencia del servidor de aplicaciones web sea menor que 100 ms. Las principales partes involucradas en este acuerdo son las partes signatarias: cliente y proveedor del servicio. Además, puede haber terceras partes involucradas que cumplen el rol de auditores encargados de comprobar que los términos del servicio de calidad sean cumplidos.

Para que sea posible comprobar el cumplimiento del SLA es necesario contar con las herramientas y procedimientos apropiados que permitan medir los atributos correspondientes a los RNFs expresados en el documento. Para ello es necesario un monitor que sea capaz de extraer datos en el nivel de aplicación ya que los términos del SLA habitualmente asumen que serán garantizados a este nivel. Sin embargo este es un trabajo complicado para los monitores convencionales, ya que los datos extraídos de la capa de plataforma o la de infraestructura no pueden asociarse de manera obvia con las métricas requeridas en la capa de aplicación para la medición de los RNFs expresados en el SLA [18]. Como se ha comentado en el punto anterior, una aplicación puede compartir una o varias máquinas virtuales así como ejecutarse en varias máquinas físicas a la vez, hecho que dificulta la asociación de la semántica de los datos de bajo nivel con respecto a las características de más alto nivel que quieren ser monitorizadas.

Debido a ello, las técnicas de monitorización del SLA, más que necesitar de un acercamiento a la extracción de datos específico, se centran en aspectos de más alto nivel. Su objetivo es conseguir establecer las relaciones pertinentes entre las especificaciones del SLA (muchas veces escritas en lenguaje natural) y el tipo de datos a obtener del servicio en la nube que permitan medir esos RNFs. De forma que el posterior análisis de los datos obtenidos permita determinar el cumplimiento o no de los requisitos establecidos en el SLA.

CASViD (*Cloud Application SLA Violations Detection*) presentado por Emarkaroha et al. [18] es una muestra de este acercamiento a la monitorización del SLA. Este método de monitorización basa la extracción de datos crudos de monitorización en la ya conocida técnica de monitorización por agentes, comentada en la sección 2.3.1 de este mismo documento. En esta ocasión es de mayor importancia el marco de trabajo de gestión del SLA (*SLA Management*



Framework) establecido por CASViD. Este marco de trabajo es el encargado de gestionar los términos del SLA, y su relación con el marco de trabajo del monitor (*Monitor Framework*) y el desplegador de aplicaciones (*Application Deployer*).

El SLA Management Framework se encarga de traducir las especificaciones del SLA, de más alto nivel, en instrucciones para el *Monitor Framework* y el *Application Deployer* encargadas, respectivamente, de recoger los datos de los servicios a ser monitorizados y de desplegar y configurar los agentes necesarios para obtener dichos datos. El método de monitorización del SLA presentado por CASViD centra la mayor parte de su esfuerzo en extraer los datos de monitorización más relevantes, refinando los períodos de extracción de datos del servicio monitorizado y en realizar un análisis preciso de los resultados obtenidos de manera que sean relevantes para la evaluación del SLA.

2.3.4 Comparación de herramientas de monitorización

Para la comparación de las herramientas de monitorización se han tomado un conjunto de soluciones comerciales y académicas populares, incluyendo las propias de los grandes proveedores de servicios en la nube. Se ha comparado la capacidad de ser utilizada en distintas plataformas, el tipo de tecnología que utilizan, el tipo de datos que son capaces de extraer de los servicios, si ofrecen monitorización y detección de violaciones del SLA y si son capaces de incorporar a su proceso de monitorización métricas personalizadas. La Tabla 1 presenta un resumen de la comparativa de estas herramientas.

Tabla 1. Herramientas de Monitorización de Servicios Cloud

	Multiplataforma	Datos de Infraestructura	Datos de Plataforma	Datos de Aplicaciones	Basada en agentes	MaaS	Monitorización del SLA	Métricas personalizadas	Tipo de solución
Azure Watch/Diagnostics	No	Si	Si	Si*	-**	No	No	Si	Comercial
Amazon Cloud Watch	Windows, Ubuntu, Amazon Linux	Si	Si	Si*	Si	No	No	Si	Comercial
Google Cloud Monitoring	Si***	Si	Si	Si*	Si	No	No	Si	Comercial
Nagios	Linux, Unix, Windows.	Si	Si	Si	Si.	No	No	Si	Comercial, GPLv2
CASViD	Independiente	Si	Si	Si	Si	No	Si	Si	Académica
MonPaaS	Linux, Unix, Windows	Si	Si	Si	Si	Si	No	Si	Académica
CloudVMM [20]	Independiente	Si	Si	No	No	No	No	No	Académica
Zabbix [21]	Independiente	Si	Si	Si	Si	No	No	Si	Comercial, GPLv2
Ganglia [22]	Linux, Solaris, BSD	Si	Si	No. Solo recursos	No.	No	No	Si	Comercial, BSD

*Limitados.

**Detalles de implementación privados.

***Google Cloud Monitoring permite monitorizar distintos paquetes de populares como Apache, MySQL, RabbitMQ, MongoDB, Tomcat y muchos otros [19].

2.3.5 Conclusiones

Las herramientas analizadas y la revisión de la literatura al respecto de la monitorización de servicios en la nube destacan el esfuerzo realizado por encontrar métodos de la extracción de la información de los servicios eficientes y precisos, tratando de salvar los obstáculos presentados por la tecnología en la nube. En su mayoría optan por las técnicas basadas en agentes por ser las que más datos pueden extraer del servicio, pero en casos que esto no sea posible existen herramientas que utilizan otras técnicas.

En su mayoría centran la atención en la extracción de datos de los niveles de infraestructura y plataforma de bajo nivel y se deja en manos de otras aplicaciones o del usuario el análisis de estos datos con el fin de establecer correspondencias entre atributos de calidad (elasticidad, seguridad, fiabilidad) y datos crudos procedentes del servicio. Sin embargo muchas herramientas ya disponen de la posibilidad de incorporar métricas personalizadas al proceso de extracción de datos, sin embargo estas deben ser en muchas ocasiones implementadas por el usuario en el propio servicio, incapaces de aprovechar la infraestructura de monitorización más para, la nada trivial, tarea de comunicar estos datos con un servidor de monitorización.

Destaca el esfuerzo de CASViD por interponer una capa de abstracción en forma de marco de trabajo que permita a los usuarios establecer métricas de mayor nivel, reportando mayor calidad de información.

A sí mismo, la monitorización de los SLAs no aparece en la mayoría de aplicaciones comerciales, el usuario debe interpretar los datos de monitorización con el fin de corroborar el cumplimiento o no de los acuerdos de nivel de servicio.

A excepción de las herramientas propias de las plataformas de provisión de servicios en la nube, el resto de herramientas hacen especial hincapié en estar disponibles para diversas tecnologías. A pesar de la interesante propuesta de [15] de crear la monitorización como un servicio, las propuestas encontradas todavía se encuentran en fase de investigación, hecho que puede estar relacionado con lo reciente de la propuesta (2013).

3. Cloud computing

En esta sección se ahondará en los conceptos relativos al *cloud computing* o computación en la nube. Primero recogiendo una definición general del concepto de *cloud computing*. En segundo lugar se recogerán las ventajas y desventajas de esta tecnología con respecto a la provisión de software tradicional. Se continuará estableciendo la clasificación de los distintos tipos de servicios en la nube ofertados en la actualidad. Por último se tratará el documento de acuerdo de nivel de servicio y la importancia de su rol en los servicios en la nube.

3.1 Definición de computación en la nube

Cloud computing es un término de jerga de las tecnologías de la información nacido a principios de 2007 [11] bajo el cual se agrupan la infraestructura, las herramientas y métodos bajo los cuáles se pueden externalizar las actividades de tecnologías de la información a uno o más terceros que tienen recursos suficientes para cumplir con las necesidades de las organizaciones de forma fácil y eficiente. Estas necesidades de recursos pueden incluir hardware, componentes, redes, almacenamiento y sistemas software, al igual que otras infraestructuras físicas como son el espacio de funcionamiento de los equipos hardware, equipos de refrigeración, electricidad y recursos humanos para mantener todos esos elementos.

El proveedor Cloud o proveedor de servicios en la nube es aquella organización que posee estos recursos y los facilita a los clientes o consumidores de servicios en la nube interesados, encargándose de la gestión y mantenimiento de los mismos.

Este paradigma tiene su origen en los modelos de negocio de *hosting* web (algunos relevantes como 1and1, Fatcow, Godaddy), proveedores de servicios de aplicaciones (Application Service Provider (ASP) como Paypal), proyectos de computación colaborativa (SETI@home, BOINC) y proveedores de almacenamiento online (Dropbox, Megaupload, Google Drive).

Cloud computing es un modelo de computación bajo demanda ofrecido por los proveedores de servicio que definen la calidad de su servicio bajo unos términos de calidad de servicio (*Quality of Service*, QoS). Los atributos que caracterizan la computación en la nube según [11] y la definición de *cloud computing* procedente del NIST (*National Institute of Technology*) [25] son:

- **Modelo de Computación bajo demanda:** Las organizaciones ya no tienen la necesidad de construir y mantener sus propios centros de datos. Sus necesidades están cubiertas por los recursos contratados a un proveedor que posee unos recursos mucho mayores que los que podría obtener la empresa a un precio menor.
- **Autonomía:** La complejidad de gestión y mantenimiento del servicio son completamente transparentes al cliente. Este solo se encarga del consumo del servicio.
- **QoS predefinidos:** Los términos de calidad van expresados en los acuerdos de nivel de servicio (SLA) los cuáles expresan los parámetros de calidad entre los cuales el servicio va a ser ofrecido.
- **Basado en internet:** Todos los servicios están situados fuera de los límites del cliente y son ofrecidos vía internet.
- **Fácil de usar:** Los servicios son proporcionados con interfaces fáciles de utilizar para ser consumidos por el cliente, ya sea a través de interfaces web o en el caso de servicios

en los que el cliente sean otras aplicaciones bajo métodos estándar de intercambio de datos (XML, JSON, SOAP).

- **Escalable:** Al disponer de recursos bajo demanda, las aplicaciones en la nube pueden aumentar sus recursos cuando las condiciones de funcionamiento lo requieran de forma automática.
- **Medible:** Los servicios cloud deben de controlar y optimizar el uso de sus recursos de manera automática. Para ello deben de ser capaces de monitorizar el uso de recursos que puede ser reportado al cliente de manera transparente.
- **Resource pooling:** Los recursos del proveedor de servicios se utilizan para servir varios clientes al mismo tiempo, asignando de forma dinámica los diferentes recursos virtuales o físicos disponibles bajo demanda. Por lo general, el cliente no conoce la ubicación física exacta de los recursos que le están sirviendo aunque puede conocerse a alto nivel (país, provincia o centro de datos).

Este marco técnico va acompañado por un modelo de negocio en el cual los usuarios pagan por el uso de estos servicios externalizados en lugar de comprar, instalar y mantener sus propias infraestructuras. Este modelo contempla el *pay-per-use* y el *pay-as-you go*, modelos bajo los cuales el usuario paga solamente por los recursos que utiliza. Estos dos modelos de facturación son ligeramente diferentes: bajo el modelo *pay-per-use* el consumidor paga periódicamente una cantidad fija por la utilización de cierto paquete de servicios. Bajo el modelo de facturación *pay-as-you-go* el consumidor solamente paga por la cantidad de recursos que efectivamente han sido utilizados durante el período de facturación. Ambas pueden ser utilizadas en combinación por un mismo proveedor de cloud para realizar una facturación más precisa.

3.2 Ventajas e inconvenientes del cloud computing

Las ventajas del *cloud computing* sobre el empleo de métodos tradicionales de aprovisionamiento de software y hardware son muchas, a nivel económico y de calidad de servicio, tanto como para el proveedor de servicios como para el consumidor.

En primer lugar, circunscribiéndose a la faceta económica, una aplicación desplegada en la nube presenta un coste inicial menor para el consumidor de servicios cloud. Puesto que en período inicial de la implantación de un servicio software, la utilización de recursos necesarios para cubrir las necesidades de la aplicación será previsiblemente más bajo que cuando esté a pleno rendimiento, el coste por la utilización de los servicios del proveedor también lo será debido al modelo de facturación de pago por uso. A diferencia de la situación dada por un modelo tradicional de aprovisionamiento de software en el que el coste de la infraestructura y/o de la licencia del propio software deberá ser abonada en su totalidad en las primeras fases de la implantación del servicio.

Este modelo así mismo ofrece ventajas económicas debido a la capacidad de escalar los servicios bajo demanda, mientras que un modelo tradicional supondrá proveerse de la infraestructura necesaria para cubrir los máximos teóricos de demanda (corriendo el peligro de dejar de ofrecer el servicio en caso de que la demanda los supere) con el sobrecoste que ello acarrea. La realidad es que este máximo sólo se alcanzará en momentos puntuales de alta demanda del servicio estando la infraestructura del mismo infrautilizada la mayor parte del tiempo.

Un servicio en la nube en cambio, escalará según las necesidades de la demanda, evitando que el servicio deje de ser prestado por falta de recursos, facturando exactamente el consumo realizado de los recursos del proveedor.



Dado que el servicio es accesible mediante internet el tiempo de puesta en marcha de nuevos servicios se reduce hasta ser prácticamente nulo puesto que no es necesario ni planificar, comprar e instalar recursos hardware adicionales ni tampoco hay que lidiar con problemas de instalación o actualización del software, ya que será el proveedor el encargado de realizar estas gestiones de manera centralizada.

Por último no se deben olvidar las ventajas propias de la subcontratación de un servicio que reducirán los costes de mantenimiento de la infraestructura que implican personal, actualizaciones periódicas del hardware, cambio de elementos deteriorados, mantenimiento de un espacio físico dedicado a la infraestructura, cumplimiento y auditorías periódicas de estándares...

En cuanto a las características de calidad se refieren, los servicios alojados en la nube disponen de alta disponibilidad y facilidad de acceso, en unos niveles especificados en el contrato de acuerdo de servicio por el proveedor cloud. Un proveedor aparte de contar con personal técnico experto que evitará momentos de caída del servicio por problemas técnicos, posee los recursos suficientes como para que en caso de desastre exista la capacidad de realojar los servicios en otra ubicación física. Una situación considerablemente más complicada y costosa de solventar siguiendo un modelo tradicional y en casos en los que el consumidor sea una organización pequeña.

Igualmente, el hecho de que los recursos sean accesibles por internet hace posible la gestión de los servicios de forma remota sin la necesidad de desplazarse in situ para operaciones de mantenimiento del servicio. A su vez esto resulta útil a la hora de actualizar el software, dado que el modelo de computación en la nube permite de forma sencilla el cambio de las instancias sin que ello afecte (en la mayoría de las ocasiones) a la disponibilidad del servicio.

A su vez, esta facilidad de acceso permite a una organización acceder a un mercado global en el cual pueden ofertar sus productos software a cualquier organización que pueda acceder a ellos vía internet. En un modelo de negocio completamente tradicional, esta posibilidad no existiría sin que la organización tuviera que desplazar recursos (tanto humanos como de infraestructura) al lugar donde el cliente quiera utilizar su software.

Desde la perspectiva del provisionador de los servicios también existen ventajas. En las organizaciones los recursos de hardware no suelen utilizarse a su máxima capacidad, ofreciendo un rendimiento de los servicios reducido con respecto a su coste de adquisición. En *cloud computing*, dado que todos los servicios comparten recursos es más sencillo para la organización gestionarlos de manera que se utilicen a su máxima capacidad durante la mayor parte del tiempo. De igual manera, se permite el alquiler de recursos hardware a otras organizaciones reduciendo el coste por utilización del hardware y por tanto aportando mayores beneficios a la organización.

El fácil acceso a los servicios permite que estos sean monitorizados permitiendo a los proveedores de servicios cloud y a los consumidores de los mismos tener a su disposición la información adecuada para tomar las decisiones necesarias para mejorar la calidad del servicio de manera continua.

Todos estos beneficios en el lado del proveedor redundan en una reducción de los costes y un aumento de la calidad del servicio por lo que el consumidor también se beneficiará de forma indirecta de la aplicación de este modelo.

Sin embargo la computación en la nube no está libre de aspectos negativos. Al tratarse de una tecnología relativamente nueva su utilización entraña ciertos riesgos para la organización que hay que tener en cuenta antes de implantar este modelo.

En primer lugar no existen estándares de comunicación entre los proveedores de software y los clientes, de manera que el software del proveedor está fuertemente acoplado al del cliente impidiendo que este pueda cambiar de forma fácil de proveedor. Esto es así debido a las tecnologías y estándares propias de cada proveedor y a las APIs no estandarizadas ofrecidas por cada uno de ellos.

En segundo lugar, una parte esencial de la computación en la nube es la delegación de parte de las otrora responsabilidades de la organización en un proveedor de servicios en la nube con lo que se genera una dependencia de este. La organización puede estar preocupada de que el proveedor deje de ofrecer sus servicios o, peor aún, utilice la información extraída de la empresa para beneficio propio y sin el consentimiento de la organización.

Dado que son los proveedores aquellos que tienen el control sobre los recursos cloud estos pueden realizar cambios sin notificar a los clientes, por ejemplo en el modelo de APIs utilizadas o en las características de los servicios ofrecidos. Es necesario que exista transparencia entre el proveedor de servicios y el cliente, acordando cada cambio que puede comprometer los puntos especificados en el SLA y garantizando en todo momento su cumplimiento.

En tercer lugar, uno de los problemas cruciales de la computación en la nube es la seguridad. Cuando una organización utiliza un servicio en la nube pierde el acceso físico a su información. Esta se puede considerar expuesta a que el proveedor o terceros como agencias de inteligencia o competidores puedan acceder a datos sensibles.

A su vez, El uso de tecnologías cloud crea dependencia de la conexión a Internet para el funcionamiento de la organización. Por este motivo la organización debe tomar medidas adicionales, por ejemplo contratación de servicios de internet redundantes, mayores anchos de banda, equipo de comunicaciones especial, etc. Para paliar este problema. Ello puede incurrir en un sobre coste que disminuiría los beneficios de no tener que disponer de la infraestructura propia in situ.

No solamente un corte del acceso a internet en la organización puede hacer peligrar la disponibilidad del servicio. Aunque los proveedores de servicios hacen especial hincapié en conseguir altas tasas de disponibilidad, manteniendo centros de datos redundantes y siguiendo buenas prácticas de seguridad en la construcción de los mismos; pueden darse circunstancias de causa mayor que impidan proveer de servicios en momentos puntuales por lo que las grandes organizaciones suelen preferir hacerse cargo de copias de seguridad de los datos y servicios prestados, de manera que en una situación como esta se consiga paliar el fallo en la provisión de los servicios, aunque el coste de esta medida paliativa no es apto para organizaciones pequeñas.

Como puede comprobarse, la mayoría de los problemas en torno al modelo de *cloud computing* son de confianza entre las organizaciones, esto es debido a que no existe todavía una legislación eficaz que regule las relaciones entre los clientes y los proveedores y garantice los

cumplimientos de los acuerdos de nivel de servicio así como aspectos como las seguridad de la información.

Es deber de la organización valorar las distintas opciones a la hora de contratar un servicio en la nube y elegir aquella que más interese a la organización.

3.3 Clasificación de servicios cloud

En esta sección se expondrán los distintos métodos de clasificación de los servicios en la nube, primero atendiendo al criterio de la forma de despliegue y en segundo lugar al modelo de servicio ofrecido:

3.3.1 Modelos de despliegue

El despliegue de servicios cloud incluye aquellos procesos mediante los cuales se procede a la puesta en marcha de aquella infraestructura y software necesario para que el servicio entre en la fase de producción, es decir, esté listo para su utilización. Según las características de la infraestructura de despliegue se puede clasificar a un servicio como:

- **Private Cloud (Nube Privada):** Este término se aplica cuando la infraestructura es utilizada exclusivamente por una sola organización, estando incluidas las diferentes unidades de negocio. Sin embargo, esta puede ser poseída, operada y controlada por la propia organización, un tercero o una combinación de ambos y esta puede estar localizada físicamente fuera de la empresa. Por ejemplo entraría en esta categoría la intranet de una empresa como Mercadona.
- **Community Cloud (Nube comunitaria):** La infraestructura está provisionada para el uso de una comunidad específica de usuarios pertenecientes a distintas organizaciones que tienen objetivos compartidos. La infraestructura puede ser poseída, operada y controlada por una o más organizaciones de la comunidad, un tercero o una combinación de ambos y puede localizarse dentro o fuera de las premisas de las organizaciones. Por ejemplo, una aplicación en la nube utilizada para transmitir reportes entre dos organizaciones embarcadas en un proyecto conjunto.
- **Public Cloud (Nube Pública):** La infraestructura está provisionada para el uso del público general. Puede estar poseída, controlada y operada por un negocio, una organización académica o gobierno, o una combinación de ellos. Esta existe en las premisas del proveedor. Por ejemplo las diferentes aplicaciones de administración electrónica del Gobierno de España.
- **Hybrid Cloud(Nube Híbrida):** La infraestructura es una composición de dos o más infraestructuras en la nube distintas de entre las anteriormente citadas que, a pesar de seguir siendo entidades únicas en sí mismas, se unen mediante tecnología que les permita la portabilidad de datos y aplicaciones.

3.3.2 Modelos de servicio

Teniendo en cuenta qué tipo de servicio ofrece el proveedor podemos encontrar:

- **Software-as-a-service (SaaS, Software como un servicio):** El proveedor de servicios provee al cliente con una aplicación software ejecutándose en una infraestructura en la nube. El cliente podrá acceder a esta aplicación a través de un cliente ya sea thin (un navegador) o una interfaz de un programa. El cliente no está encargado de gestionar ningún aspecto de la infraestructura ni la plataforma, esto es no estará encargado de

gestionar el servicio a nivel del sistema operativo, interfaces de red, almacenamiento o incluso la modificación de las características de la aplicación (con la excepción de una configuración de las opciones del servicio pensada para tal efecto). De todo ello se encargará el proveedor de servicios.

- **Platform-as-a-service** (PaaS, Plataforma como servicio): El proveedor de servicios ofrece al cliente una plataforma en la que pueda desplegar sus aplicaciones. Esto es, el proveedor facilita sistemas operativos, servidores, bases de datos y red, necesarios para que el cliente utilizando las diferentes librerías y lenguajes de programación, herramientas y servicios soportados por el proveedor cree o compre aplicaciones para su despliegue en la infraestructura proveída sobre la cual tendrá control total.
- **Infraestructure-as-a-service** (IaaS, Infraestructura como servicio): Es el modelo de servicio en el cual el cliente tiene el nivel de acceso a más bajo nivel. El proveedor simplemente aporta capacidad de procesamiento, almacenamiento, redes y otros elementos de hardware fundamentales para que el cliente los utilice para desplegar y ejecutar cualquier tipo de software, incluyendo aplicaciones y sistemas operativos.

3.4 El Acuerdo de Nivel de Servicio

Cómo se ha comentado en la sección 4.2, el mayor problema actual del modelo de computación en la nube es la desconfianza entre el proveedor y el consumidor de servicios. En esta sección se describirá el rol que juega el documento de acuerdo de nivel de servicio para regular esta relación de manera que las partes implicadas conozcan sus obligaciones y derechos en ella.

Los términos de un servicio en la nube contratado viene definido por dos partes que constituyen un mismo documento: (1) el acuerdo de servicio y (2) el Acuerdo de Nivel de Servicio (*Service Level Agreement*, SLA). [26]

El acuerdo de servicio es un documento que define la relación legal entre el cliente y el proveedor de cloud. En él se especifican las reglas legales de la relación entre consumidor y proveedor.

Este es completado por el SLA un documento más corto dónde se especifican los niveles mínimos y máximos acordados acerca del rendimiento del servicio contratado así como el procedimiento para reclamar al proveedor de servicio en el caso de incumplimiento de alguno de ellos, especificando a su vez las compensaciones que recibiría el cliente en cada una de estas situaciones.

Esto es, el SLA describe la calidad mínima de servicio ofrecida por el proveedor cloud. Por norma general en ámbitos comerciales, los proveedores ofrecen unos términos no negociables en el acuerdo de nivel de servicio que el cliente puede aceptar o buscar otro proveedor de servicios más acorde a sus necesidades. Solamente en casos en los que la organización tenga mucho peso en la contratación de un servicio (siendo uno a medida o contratando un gran volumen de servicio) puede llevarse a cabo un contrato personalizado.

Este contrato puede ser suspendido en cualquier momento por cualquiera de las partes ya sea por alguna causa, por ejemplo que el proveedor del servicio viole sistemáticamente las condiciones de nivel de servicio o que el cliente deje de pagar la cantidad acordada en el contrato; o por ningún motivo en concreto.

El SLA está formado por 3 partes básicas: (1) una colección de promesas hechas al consumidor de cloud, (2) una colección de promesas específicamente no hechas al consumidor y (3) un conjunto de obligaciones que el consumidor tiene que cumplir.

El tipo de promesas que suele tener un SLA suelen estar referidas a la disponibilidad, las compensaciones que se harán por incumplimiento del rendimiento, la preservación de los datos y al cuidado legal de la información del consumidor (por ejemplo, el cumplimiento de la Ley Orgánica de Protección de Datos).

La disponibilidad típicamente está ofertada como tiempo de ejecución en un porcentaje entre el 99,5% y el 100%. La disponibilidad es un atributo muy importante en un servicio en la nube por lo que estas promesas se tratan con cuidado y el proveedor suele expresarlas como porcentaje de disponibilidad por intervalos. Los más comunes son los intervalos de 5 minutos, 15 minutos y 1 hora. Para ilustrar esta forma de medir (que debe estar explícitamente indicada en el SLA) si el proveedor especifica un 100% de disponibilidad en intervalos de 1 hora y el servicio no está funcional durante 59 minutos pero sí durante el minuto restante, se sigue manteniendo el 100% de disponibilidad en intervalos de 1 hora. Así mismo, la definición de no disponible no está completamente acotada dado que un servicio puede encontrarse disponible hasta que varios subsistemas fallen antes de que sea juzgado como no disponible. Por ello el proveedor también puede especificar aquellas máquinas virtuales o funciones sobre las cuáles se aplica una determinada disponibilidad y sobre cuáles no.

En el caso de que el proveedor falle a la hora de cumplir alguna de las promesas hechas en el SLA, el proveedor deberá compensar de alguna manera al cliente cloud. Generalmente esta compensación viene dada como crédito en el proveedor del servicio. Típicamente este crédito se calcula en base al tiempo que se ha fallado con el cumplimiento del SLA y se devuelve entre el 10% y el 100% del coste actual del consumidor aunque estos números varían según el proveedor de cloud. El NIST especifica en [26] que en 2011 ninguno de los proveedores de cloud preguntados ofrecía este tipo de retribuciones. Sin embargo en el momento de este escrito Amazon Cloud Services[27] y Google Cloud Platform [28] sí que ofrecen este tipo de retribuciones tal y como se pueden observar en sus acuerdos de nivel de servicio.

Las cláusulas relativas a la preservación de los datos conciernen a los datos que el servicio puede almacenar sobre el consumidor cloud y el tiempo que estos pueden encontrarse almacenados en el proveedor de cloud. En términos generales, si el consumidor rompe el acuerdo de nivel de servicio por una causa (por ejemplo impago) el proveedor cloud suele enunciar que no tienen obligación de preservar los datos del cliente. Si es el cliente el que decide voluntariamente abandonar el servicio, sus datos podrían conservarse, generalmente, durante 30 días. Algunos proveedores especifican que solamente guardan una instantánea de los datos del consumidor con lo que recomiendan que estos datos sean almacenados en una copia de seguridad en otro lugar.

En lo concerniente al uso de la información del consumidor, los proveedores suelen prometer no vender, licenciar o hacer públicos los datos del consumidor excepto en respuesta a una petición legal. Aunque muchos proveedores normalmente se reservan el derecho a monitorizar las acciones del consumidor en la nube e incluso pueden pedir una copia del software del consumidor para ayudarles en ese propósito.

Por último, las limitaciones en el SLA describen excepciones a la hora de tratar las promesas del contrato. Generalmente se suelen incluir en estas limitaciones las paradas del servicio programadas, que no contarán como un fallo de rendimiento aunque deben estar avisadas con antelación y acotadas en duración; eventos de fuerza mayor fuera del control del proveedor, circunstancias como desastres naturales permitirán el no cumplimiento del contrato; cambios en el acuerdo de servicio, generalmente los proveedores se reservan el derecho a realizar cambios unilaterales en el acuerdo de nivel de servicio, incluyendo los cambios en la facturación; la seguridad, en general el proveedor no se hace responsable de los impactos producidos por una brecha de seguridad o cualquiera actividad que concierna la seguridad; y finalmente, los proveedores se suelen reservar el derecho a cambiar la API del servicio en cualquier momento.

Por la parte del consumidor este debe cumplir con una serie de políticas de uso aceptables, recogiendo cómo o con qué fines no puede utilizarse los servicios del proveedor; aceptar la licencia del software de terceros que tenga el proveedor instalado en su servicio y para terminar el consumidor se compromete a pagar puntualmente el importe debido al proveedor dentro del período de facturación.

En la mayoría de los casos, el proveedor del servicio no se hace cargo de la monitorización del cumplimiento de estos acuerdos de nivel de servicio y debe de ser el consumidor del mismo el encargado de obtener estos datos y realizar las reclamaciones oportunas en caso de fallo del proveedor a la hora de cumplir sus promesas.

4. Arquitectura de Monitorización basada en Modelos en Tiempo de Ejecución

En esta sección se presenta la arquitectura de la monitorización basada en Modelos en Tiempo de Ejecución concebida por Cedillo et al [9] y refinada en posteriores trabajos [29] [30] esta arquitectura es la implementada en este trabajo, expuestos sus detalles más importantes en el capítulo 6 de este documento.

En primer lugar se procede a presentar la metodología, y a continuación se presenta el proceso de monitorización y se finaliza describiendo sus componentes más importantes así como las relaciones entre ellos y algunos detalles de diseño.

4.1 Presentación de la Arquitectura de Monitorización basada en Modelos en Tiempo de Ejecución

Como se ha comentado en la motivación, tanto proveedores de servicios cloud como los consumidores de estos servicios están interesados en la monitorización de los servicios en la nube para poder verificar el rendimiento de ese servicio y comprobar que se cumplen los acuerdos de nivel de servicio ofrecidos por el proveedor, así como verificar que la tarificación del servicio es concorde a la utilización real del mismo.

Así mismo, las opciones de monitorización tradicionales han demostrado estar restringidas a un entorno estático y homogéneo y, por lo tanto, no están capacitadas (sin adaptarse) para ser utilizadas en entornos en la nube los cuáles necesitan de estas características para enfrentarse a los problemas propios del entorno dado que la computación en la nube necesita medir la calidad de software en entornos de alto rendimiento, escalables y elásticos.

El carácter mutable y dinámico de la computación en la nube hace necesario que las herramientas de monitorización sean capaces de cambiar los requisitos no funcionales a monitorizar basados en los cambios que los stakeholders vayan introduciendo en cualquier momento a lo largo del ciclo de vida del software. Por ello es necesario definir un acercamiento al problema que permita la adición o modificación de nuevos requisitos no funcionales en tiempo de ejecución sin la interrupción de la ejecución del servicio.

De igual manera, las herramientas de monitorización presentadas en el capítulo 2 de este documento, se encargan de la extracción de datos de bajo nivel, a excepción de CASViD , y no presentan un entorno de trabajo adecuado para la expresión de combinaciones de estos datos que permitan la monitorización de otros RNFs de mayor nivel de manera directa.

Cedillo et al [12] proponen el uso de Ingeniería Dirigida por Modelos (*Model Driven Engineering*, MDE) para enfrentarse a este problema presentando una arquitectura de monitorización basada en modelos en tiempo de ejecución. Este proceso de monitorización se centra en comprobar el cumplimiento de los requisitos no funcionales prometidos en el SLA así como otros requisitos no funcionales que se deseen monitorizar. Debido a sus características, este proceso de monitorización se centra en monitorizar nubes que implementen servicios de Software as a Service. El tipo de despliegue es

irrelevante, dado que la situación, propiedad o uso de la plataforma donde se despliega no afecta a la medición de la calidad del servicio ofrecido.

4.1.1 Empleo de los modelos en tiempo de ejecución

Un modelo en tiempo de ejecución está definido como una abstracción de un sistema en ejecución el cual está siendo manipulado en tiempo de ejecución para un propósito específico [31] o una representación causalmente conectada del sistema asociado que enfatiza la estructura, el comportamiento y los objetivos del sistema desde una perspectiva del problema [32].

El uso de modelos en el ámbito de la monitorización de servicios permite en tiempo de diseño expresar y clasificar los requisitos no funcionales a monitorizar, de manera que estos se puedan expresar en base a datos crudos extraíbles de la plataforma. Sin embargo, no todos los requisitos no funcionales a monitorizar pueden ser expresados en tiempo de diseño por lo que es necesaria una tecnología que sea capaz de romper las barreras entre el tiempo de diseño y de ejecución, aportando así la flexibilidad para el cambio necesaria en el entorno cloud.

Aunque Cedillo et al [12] en el momento de la publicación no ha encontrado estudios que utilicen los modelos en tiempos de ejecución, sostienen que resultan útiles para este propósito dado que los desarrolladores no necesitarían realizar una nueva implementación cada vez que aparecieran nuevos requisitos, con el coste y esfuerzo logístico que ello implica en un contexto en la nube, sino simplemente añadirlos al modelo en tiempo de ejecución encargado de representar estos datos.

La relación entre los atributos de calidad del servicio (*Quality-of-Service*, QoS) dispuestos en el SLA y la arquitectura de monitorización es estrecha, ya que de ellos depende la implementación de la monitorización de los requisitos no funcionales. De esta manera, se ve necesario que exista constantemente una representación de la vista de calidad que relaciona los requisitos no funcionales a ser monitorizados con los datos en crudo que hacen posible la monitorización de estos requisitos de alto nivel.

Por estas razones se decide realizar este proceso tomando como núcleo la expresión de los términos de calidad en un modelo en tiempo de ejecución.

4.2 El proceso de monitorización

El proceso de monitorización presentado consta de tres tareas subdivididas en actividades particulares y con unas entradas y unas salidas claramente definidas. En este apartado se procederá a la exposición de cada una de estas tareas así de como su diseño.

Este proceso de monitorización presentado es un sistema autónomo en el cual se emplea la técnica de bucle de control (*Control Loop*). Los sistemas autónomos se caracterizan por ser capaces auto-regularse, adaptándose a cambios no predecibles mientras ocultan su complejidad intrínseca a los usuarios. Una de las ventajas de esta técnica es, por tanto, la reducida necesidad de intervención de un componente humano para operar el software, reduciendo así los errores derivados en consecuencia. El bucle de control es una arquitectura que permite al sistema retroalimentarse con sus resultados para tomar medidas de manera autónoma.

El propósito principal de este proceso de monitorización es medir los parámetros del sistema, analizarlos y con los resultados obtenidos planificar medidas para la mejora de la calidad del servicio para ser ejecutadas de manera autónoma.

Para llevar a cabo este objetivo el proceso se ha organizado en las presentes tres fases de las que constará el bucle autónomo: Configuración de la Monitorización (1), Proceso de Medición (2) y Análisis de resultados (3).

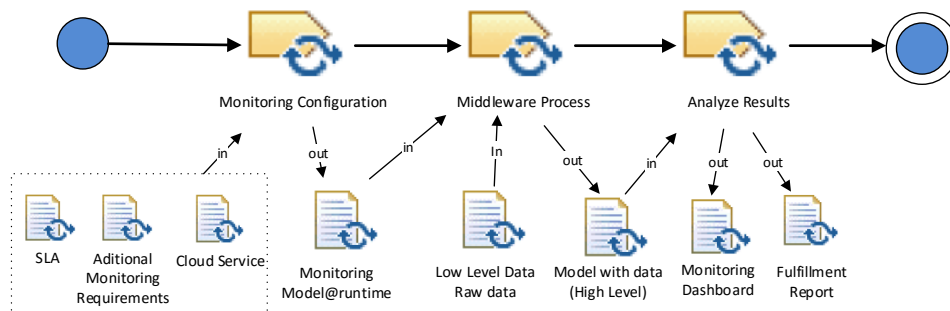


Figura 1 – El Proceso de Monitorización.

El proceso de monitorización comienza con la Configuración de la Monitorización que creará el modelo en tiempo de ejecución que será utilizado por el middleware de monitorización en su proceso de extracción de datos y medición de los mismos. Los datos obtenidos en este proceso serán de mayor nivel y permitirán a la tarea de Análisis de Resultados obtener un informe sobre la calidad del servicio.

A continuación se procede a exponer las características de cada una de estas tareas así como las entradas y salidas empleadas.

4.2.1 Configuración de la Monitorización

El propósito de la tarea de Configuración de la Monitorización es crear el modelo en tiempo de ejecución poblado con los requisitos no funcionales y las métricas que serán usadas a posteriori para realizar las tareas de Proceso del Middleware y Análisis de Resultados.

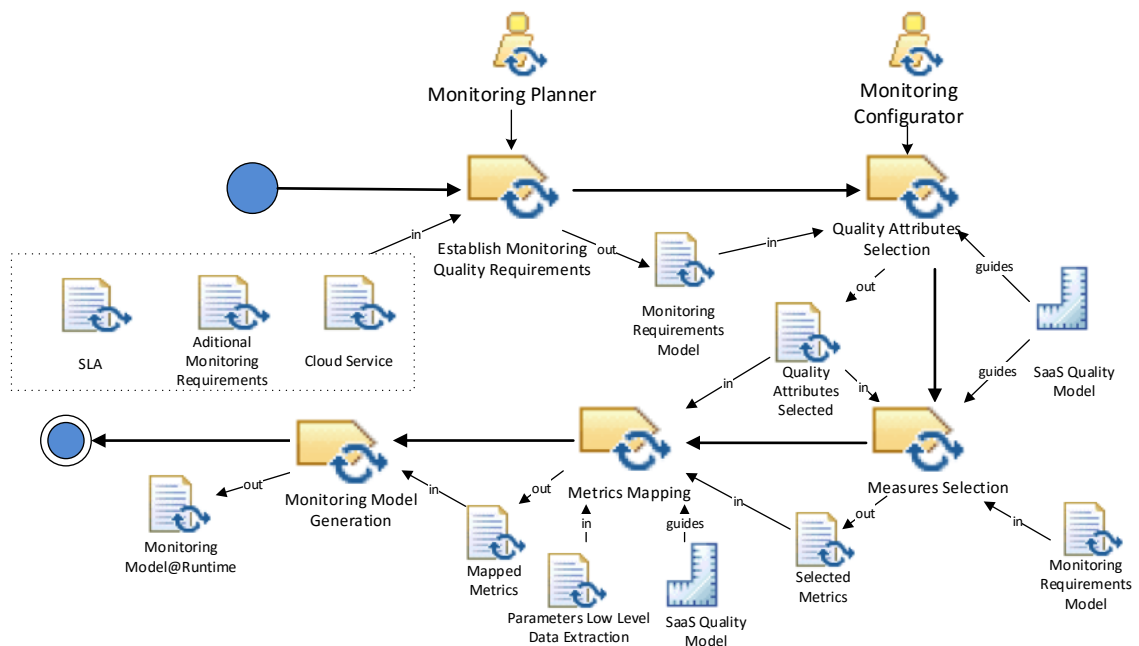


Figura 2 – La Configuración de la Monitorización.

Según se puede observar en la Figura 2 esta tarea se desglosa en cuatro subtareas (en la secuencia establecida: Establecimiento de los Requisitos de Calidad, Selección de los Atributos de Calidad, Selección de las Medidas, Asociación de Métricas y Generación del Modelo de Monitorización) desempeñadas por dos roles diferentes (Planificador de la Monitorización y Configurator de la Monitorización).

4.2.1.1 Artefactos y roles

En esta sección se describirán los artefactos y los roles partícipes en la tarea de Configuración de la Monitorización expuesta en la sección anterior.

- **Planificador de la monitorización:** Este rol debe ser desempeñado por un usuario con conocimientos sobre los acuerdos de nivel de servicio alcanzados y otros requisitos de monitorización útiles para la misión de la organización. A su vez debe tener conocimiento general de la plataforma del servicio y ser capaz de expresar estos conceptos en formulaciones que faciliten su medición, estableciendo los límites aceptables.
- **Configurador de la Monitorización:** Este rol es desempeñado por un usuario con conocimientos profundos del Modelo de Calidad SaaS, conocimientos sobre calidad de software y un conocimiento profundo acerca de los datos que se pueden extraer de la plataforma del servicio a monitorizar para ser capaz de tomar las decisiones oportunas.
- **SLA:** Este artefacto representa al documento del acuerdo de servicio donde se especifican los niveles mínimos y máximos acordados acerca del rendimiento del servicio contratado así como el procedimiento para reclamar al proveedor de servicio en el caso de incumplimiento de alguno de ellos, especificando a su vez las compensaciones que recibiría el cliente en cada una de estas situaciones. Este documento ha sido tratado con mayor profundidad en la sección 4.4

- **Requisitos adicionales de la monitorización (Additional Monitoring Requirements):** Artefacto que representa a los requisitos de monitorización no incluidos en el SLA pero incluidos en la monitorización debido a que responden ante los intereses de la organización.
- **Servicio en la nube (Cloud Service):** Este artefacto representa la información relativa al servicio que quiere ser monitorizado (tipo de servicio, plataforma en la que se ha desplegado, nombre del servicio, configuración, datos de acceso, etc).
- **Modelo de requisitos de la monitorización (Monitoring Requirements Model):** Este artefacto representa a los requisitos que deberán tenerse en cuenta durante la monitorización, los límites aceptables que deben mantener y la forma en la que estos deben medirse, expresada en lenguaje natural. Este modelo puede consultarse en la sección 1.1 del anexo.
- **Modelo de calidad SaaS (SaaS Quality Model):** Este artefacto representa el modelo de calidad específico para la medición de la calidad de servicios en la nube provisionados como SaaS. Este modelo se toma como referencia a la hora de elaborar métricas válidas para la medición de atributos de calidad. La representación de este modelo puede encontrarse detallada en la sección 1.2 del anexo.
- **Atributos de calidad seleccionados (Quality Attributes Selected):** Este artefacto es un producto intermedio de la tarea. Contiene aquellos atributos que se han seleccionado para representar los RNFs presentes en el modelo de requisitos de monitorización.
- **Métricas seleccionadas (Selected Metrics):** Este artefacto es un producto intermedio de la tarea. Representa las métricas independientes de la plataforma seleccionadas para medir los atributos de calidad seleccionados.
- **Parámetros de bajo nivel de la extracción de datos (Parameters Low Level Data Extraction):** Este artefacto contiene la información relativa a los diferentes tipos de datos que pueden extraerse del servicio y la plataforma donde se despliega y que podrán emplearse para medir atributos de calidad. Como ejemplo de este tipo de datos puede consultarse la descripción de Performance Counters de Microsoft Azure en la sección 6. 3.1.
- **Métricas asociadas (Mapped Metrics):** Este artefacto es un producto intermedio de la tarea de configuración de la monitorización. Este artefacto representa las métricas dependientes de la plataforma creadas para ser equivalentes a las métricas seleccionadas para medir los atributos de calidad de los RNFs seleccionados.
- **Modelo de monitorización en tiempo de ejecución (Monitoring Model@Runtime):** Este artefacto es el producto final de la tarea de configuración de la monitorización. Este modelo contiene toda la información de la monitorización del servicio, desde sus datos a las métricas y los datos de bajo nivel empleados para monitorizar el servicio. El modelo en tiempo de ejecución se encuentra explicado en el Capítulo 4 de este documento.

4.2.1.2 Tareas

1. Establecer los Requisitos de Calidad a ser Monitorizados (*Establish Monitoring Quality Requirements*)

Esta tarea es realizada por el planificador de la monitorización. Esta tarea recibe como entradas el Acuerdo de Nivel de Servicio (*Service Level Agreement*, SLA descrito en la sección 4.4 de este documento), un documento con otros Requisitos Adicionales de Monitorización y los datos del Servicios en la Nube (*Cloud Service*) que será analizado en el proceso de monitorización. Toda esta información será recogida de manera estructurada en el documento de Especificación de Requisitos de Monitorización (*Monitoring Requirements Specification*).

2. Selección de Atributos de Calidad (*Quality Attributes Selection*)

Las características en las cuales descomponemos la calidad y sus medidas asociadas pueden ser de gran utilidad no sólo para la evaluación de un producto software, sino también a la hora de definir requisitos de calidad. En este trabajo, la calidad de un servicio cloud se expresa mediante un modelo de calidad alineado con el estándar ISO/IEC 25010 [40] que descompone la calidad del servicio en características, subcaracterísticas y atributos de calidad. El modelo también contiene métricas (con una o más operacionalizaciones) que permiten medir dichos atributos. Una operacionalización indica la fórmula de cálculo de una métrica específica ya que relaciona los atributos y métricas del modelo de calidad con los distintos parámetros y contadores de las plataformas cloud.

A continuación, se toma el documento de Especificación de Requisitos de Monitorización y utilizando como guía un Modelo de Calidad SaaS (*SaaS Quality Model*) el Planificador de la Monitorización se encarga de relacionar cada Requisito de Monitorización con el atributo del modelo de calidad SaaS apropiado. Dando como lugar al artefacto de Atributos de Calidad Seleccionados (*Quality Attributes Selected*).

3. Selección de Medidas (*Measures Selection*)

A los Atributos de Calidad Seleccionados en la tarea de Seleccionar Medidas, se los relacionará uno por uno con una métrica elegida, según los criterios del Configurador de la Monitorización, tomando en cuenta las indicaciones del Modelo de Calidad SaaS.

Las medidas definirán de manera independiente de la plataforma cómo deben medirse los atributos de calidad seleccionados que representan las propiedades de los RNFs.

La salida de esta tarea serán el conjunto de medidas asociadas a los atributos de calidad seleccionados identificadas como el artefacto Métricas Seleccionadas (*Selected Metrics*).

4. Asociación de Métricas (*Metrics Mapping*)

Las Métricas Seleccionadas todavía se encuentran expresadas en términos independientes de la plataforma no válidos para su puesta en marcha dentro de la tarea de medición del Middleware, de manera que estas deberán ser asociadas con otras Métricas ahora sí expresadas en términos dependientes de la plataforma. Estas métricas expresan a nivel de datos crudos obtenibles por la plataforma de monitorización fórmulas equivalentes a la métrica seleccionada

procedentes del modelo de calidad SaaS. En la sección 5.5 se detallan qué tipo de datos podrán utilizarse para la expresión de estas métricas.

El resultado de esta tarea se identifica en el artefacto métricas asociadas (*Mapped Metrics*).

5. Generación del Modelo de Monitorización (*Monitoring Model Generation*)

Las métricas ya escritas de manera dependiente de la plataforma situadas en el artefacto Métricas Asociadas pasarán a transformarse en un modelo en tiempo de ejecución siendo este el producto final de la primera tarea del proceso y que pasará al proceso del Middleware dónde será empleado para realizar la monitorización del servicio.

4.2.2 Proceso del Middleware

En esta tarea un componente middleware se encarga de recoger los datos del servicio especificado para ser monitorizado indicados por las métricas dependientes de la plataforma seleccionadas presentes en el Modelo en Tiempo de Ejecución. Sobre estos datos se realizarán las operaciones descritas en las distintas formulaciones de las métricas dependientes de la plataforma, transformando datos crudos en métricas de alto nivel.

Estas serán reportadas al Motor de Análisis el cuál se encargará de comparar las medidas obtenidas con las especificadas por el SLA del servicio. En el caso de que estas no se cubrieran, podrían tomarse decisiones sobre el servicio para mejorar la calidad del mismo o utilizarse para generar informes de resultados para posteriormente reclamar al proveedor si ha existido un incumplimiento de los términos del servicio.

4.2.3 Análisis de resultados

Esta es la parte del middleware que compara los valores obtenidos por el proceso de monitorización con los requisitos no funcionales, analiza los resultados y reporta el análisis. Estos resultados podrían utilizarse para planificar una estrategia de reconfiguración de la arquitectura de los servicios usando sistemas expertos o una base de conocimiento, de manera que el sistema se adapte automáticamente para conseguir cumplir con los requisitos no funcionales cerrando de esta manera el bucle de control autónomo. Sin embargo esta parte del proceso todavía está en fase de investigación [10].

4.3 Componentes del Proceso de Monitorización

En esta sección se presenta la estructura de los componentes de alto nivel que compondrán la estructura de la implementación del proceso de monitorización. En la Figura 3 se destacan dos grandes componentes el Configurator del Middleware (*Middleware Configurator*)(4) y el Middleware de Monitorización y Análisis(*Monitoring and Analysis Middleware*) (5).

A continuación se presentan ambos componentes de forma más detallada.

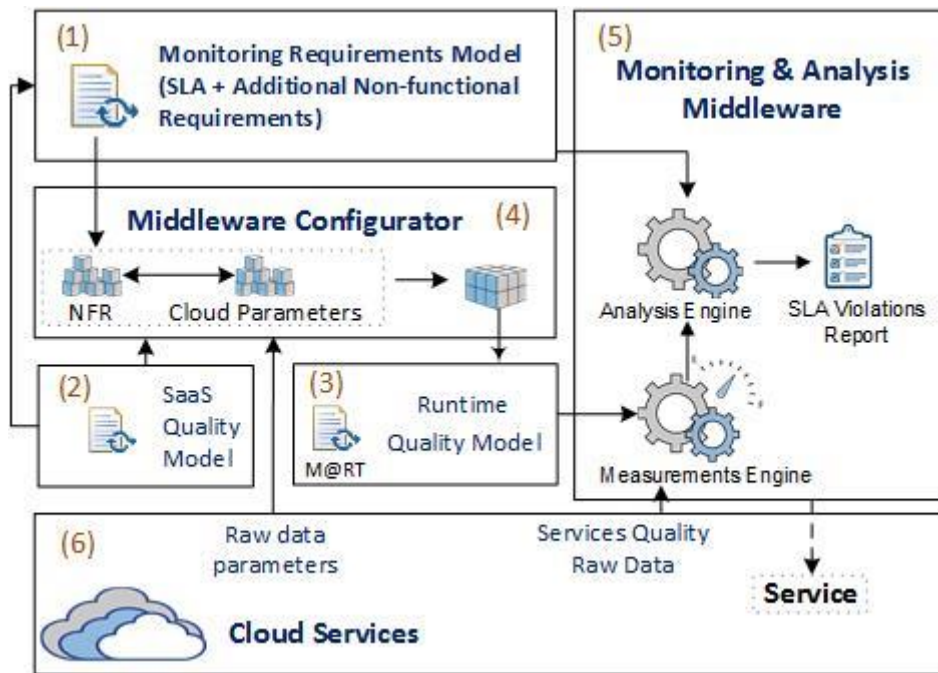


Figura 3 – Componentes del Proceso de Monitorización.

4.3.1 El Configurator del Middleware

El Configurator del Middleware es el componente que da soporte a la primera tarea del proceso de monitorización. Este componente se encarga de, asistido por un usuario, generar el modelo en tiempo de ejecución con los datos de configuración necesarios para llevar a cabo la monitorización. Utiliza como guías los requisitos no funcionales presentes en el SLA y otros adicionales, así como utiliza información relativa a la plataforma del servicio en la nube para establecer qué tipo de parámetros se pueden utilizar como se ha detallado en la sección 5.4.1.

4.3.2 El Motor de Middleware y Análisis

El Motor de Middleware y Análisis es una de las partes esenciales de este proceso, dado que en él se producen las actividades nucleares del proceso de extracción y cálculo de datos, dando apoyo a las subtarefas dos y tres del proceso.

Este componente se encarga de la recolección y procesado de datos de calidad del servicio monitorizado y de su posterior análisis para la validación del cumplimiento del SLA y la creación de reportes.

Este motor está dividido en componentes especializados: *Measurement Engine* (Motor de medición), *Platform Data Retrieval Mechanisms* (Mecanismos de recolección de datos de la plataforma) y *Analysis Engine* (Motor de Análisis).

4.3.2.1 Motor de Medición

Este componente es el encargado de realizar los cálculos sobre los datos extraídos por el Mecanismo de Recolección datos de la Plataforma.

Este motor calcula las métricas, basándose en las operacionalizaciones aportadas por el Modelo en Tiempo de Ejecución para la Monitorización y las almacena para su posterior análisis.

5.3.2.2 Motor de Análisis

Este componente se encarga de tomar los datos calculados por el Motor de Medición y tratarlos (tomar grupos de datos relevantes, por ejemplo seleccionar aquellas medidas sobre disponibilidad en rangos de 5 minutos si así ha sido expresado en el SLA) para comparar los resultados obtenidos con los establecidos por los acuerdos de nivel de servicio, de manera que se pueda comprobar si están siendo cumplidos y de reportar estos resultados tratados para que puedan ser comprensibles y útiles por una persona o software de reconfiguración dinámica autónomo que permita mejorar el sistema.

4.3.2 Mecanismos de recolección de datos de la plataforma

Cómo se ha visto en esta sección una parte vital del proceso de monitorización es el empleo de datos crudos de bajo nivel procedentes del servicio a ser monitorizado con el fin de realizar los cálculos que permitan obtener las métricas. Para la obtención de dichos valores hemos establecido cuatro escenarios de captura de datos con el fin de recoger información de calidad de bajo nivel explotando distintos mecanismo de manera que la propuesta sea flexible y extensible.

Como se ha mostrado en el capítulo 2, existen muchas herramientas comerciales especializadas en la extracción de datos útiles para la monitorización de los servicios en la nube. Mediante estos escenarios se propone aprovechar las herramientas disponibles para conseguir datos de mayor nivel en forma de métricas.

En la Figura 4 se resumen estos cuatro escenarios.

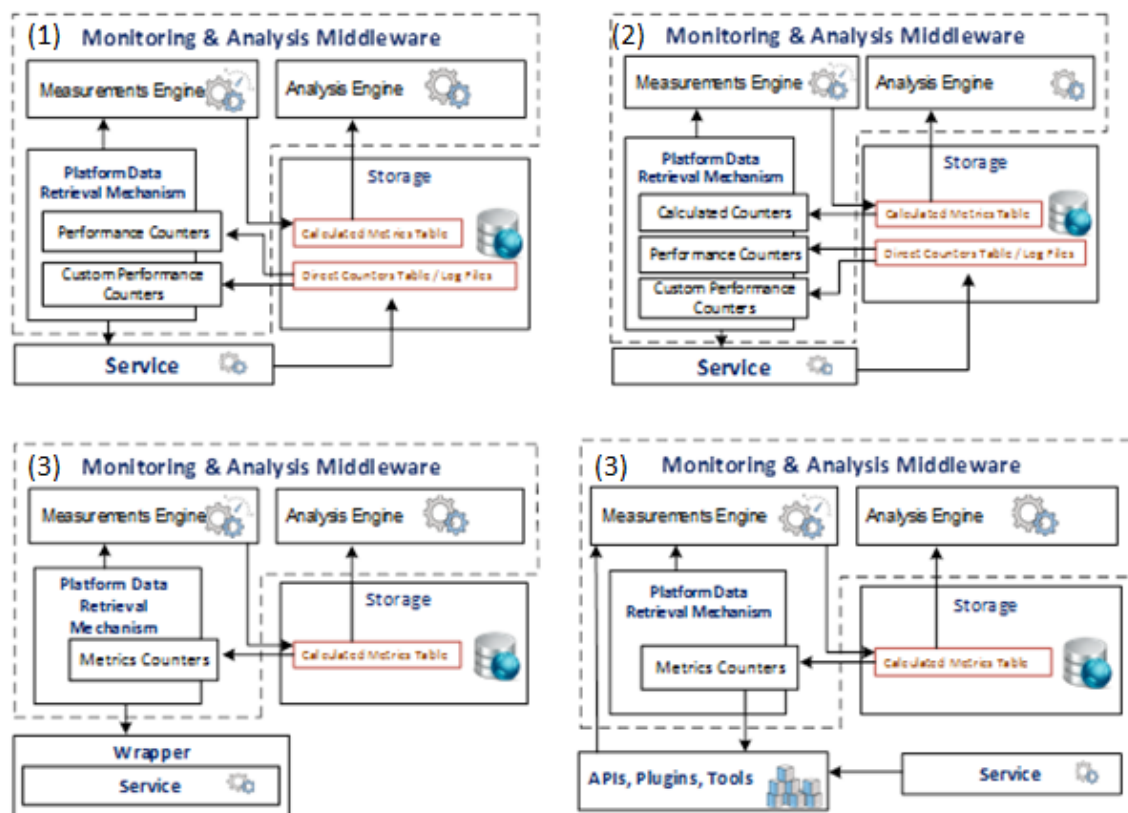


Figura 4 – Escenarios de Extracción de Datos.

El primer escenario está basado en utilizar los mecanismos propios de la plataforma que extraen de los servicios información de calidad. Por ejemplo *Diagnostics* en el caso de Microsoft Azure. A través de librerías u otro tipo de servicios la plataforma provee acceso directo a los datos a través de diferentes contadores de rendimiento. En los servicios ofertados por los proveedores de PaaS este tipo de datos son aportados por el sistema operativo y otras herramientas de extracción de datos instaladas por el proveedor. De manera general este tipo de datos suele estar asociado a información sobre las aplicaciones del servicio de bajo nivel (Número de procesos activos, tiempo de CPU del proceso, recursos utilizados, transacciones con la base de datos, etc) y otros datos referentes a la infraestructura (Uso de memoria, porcentaje de procesador utilizado, uso de la interfaz de red). El caso de Microsoft Azure es especial dado que el proveedor controla tanto la infraestructura como el sistema operativo e incluso las librerías y lenguajes de programación utilizados por lo que ofrece una solución casi integra cual le permite aportar datos de más alto nivel de forma centralizada, véase la sección dedicada a *Diagnostics* en el apartado 5.5 de este documento. Esta tecnología permite la incorporación de contadores personalizados que también podrán utilizarse en este escenario para la medición de métricas.

Este escenario, el motor de medición utiliza los contadores de forma individual, siendo el valor obtenido por el extractor de datos el final de la métrica extraída. En términos de calidad de software, este escenario modelaría el tratamiento de métricas directas. Este valor es almacenado para su posterior tratamiento por el motor de análisis.

En el segundo escenario, al igual que el primero se contempla el uso de los mecanismos propios de la plataforma para extraer los datos de calidad de bajo nivel. En términos de calidad, este escenario modela la definición y uso de métricas indirectas e indicadores (mediante la agregación de otras métricas indirectas). Este escenario permite la combinación de distintos tipos de bajo nivel y/o métricas calculadas para llevar cabo el cálculo de métricas. Estos cálculos son llevados a cabo por el motor de medición y los resultados almacenados para su tratamiento posterior por el motor de análisis.

El tercer escenario permite la programación de envoltorios de servicios que permitan extraer un mayor abanico de tipos de datos. Como ya se ha comentado en el primer escenario, los datos ofrecidos por la plataforma generalmente ofrecen información de bajo nivel o incompleta que en ocasiones puede no ser suficiente para medir satisfactoriamente los requisitos no funcionales a monitorizar. Por ello un usuario podría estar interesado en crear sus propios contadores de rendimiento que extendieran a los ofrecidos por la plataforma en los que pudiera expresar los datos que considere necesarios. Este escenario a su vez abre la posibilidad a extraer contadores propios del servicio, estrictamente ligados al dominio del mismo que mediante las técnicas de monitorización en la nube disponibles no podrían conseguirse.

Esta posibilidad, aunque aumente la flexibilidad de la herramienta, requiere de la programación de estos envoltorios en los servicios del cliente, ya que estos datos no podrían ser extraídos utilizando otra técnica menos invasiva. Este hecho podría llevar a tener que detener y actualizar instancias de los servicios a monitorizar para añadir el software necesario para la recolección de los datos. Esta inconveniencia está resuelta con el uso de otro paquete dentro del proyecto Value@Cloud donde se considera la reconfiguración dinámica de servicios, proveyendo las herramientas necesarias para que este tipo de cambios puedan realizarse de manera autónoma y automática paliando la necesidad de detener el servicio.

Una vez que la información de monitorización es extraída mediante esta técnica, se la puede considerar como un contador del primer escenario (si su valor se corresponde con la métrica a monitorizarse) o del segundo (si es necesario realizar cálculos sobre esta información).

Por último, el cuarto escenario contempla la utilización de herramientas de terceros para obtener los datos de bajo nivel procedentes de los servicios. Cómo se ha expuesto en el apartado de Estado del Arte (Capítulo 2) existen multitud de herramientas especializadas en la recolección de este tipo de información presentes en muchas plataformas. De esta manera, el usuario puede emplear una herramienta de terceros especializada en realizar este tipo de tareas y recolectar la información apropiada para su consumo, aumentando así las posibilidades del proceso de monitorización.

Si la información que se desea monitorizar pertenece a una de estas herramientas, mediante un conector (API, conexión a bases de datos u otros) se comunicará con las fuentes de datos de estas herramientas para la extracción de los datos. Al igual que en el tercer escenario, dependiendo de las características de la información recolectada se podrá recurrir al escenario primero o segundo para interpretar la información. Un ejemplo de este escenario sería la herramienta *de Amazon Cloud Watch* encargada de guardar datos de monitorización y permite acceder a archivos de *logs* de instancias *Amazon Elastic Compute Cloud (EC2)*. Esta información podría recolectarse mediante un conector que tuviera acceso a ella y así ser utilizada por el motor de medición.

Estos cuatro escenarios pueden funcionar simultáneamente creando métricas como combinación de datos procedentes de distintas fuentes si así fuera necesario para establecer el valor de una métrica.

5. Definición del Middleware de Monitorización

En este capítulo se expone la implementación del proceso de monitorización en la forma de una aplicación web y un servicio en la nube. Primero, se expondrán los requisitos de las herramientas, a continuación, se realizará un pequeño análisis a la plataforma seleccionada para la implementación y la influencia que tendrá en la implementación de este proceso. En el siguiente punto, se expondrá el diseño seguido para la realización de la aplicación web y sus características. A continuación, se detallará la implementación del Middleware de Monitorización, y por último, se expondrá un caso de estudio que ejemplificará el funcionamiento del middleware.

5.1 Introducción a las herramientas

En el presente trabajo se ha dado soporte a dos de las tres tareas principales del proceso de monitorización definido por Cedillo et al. [12] y expuesto en la sección 5.1: las tareas de Configuración de la Monitorización y de Proceso del Middleware. El propósito de esta implementación es comprobar la validez y refinar la propuesta del proceso para ello realizando un prototipo de la **herramienta para la configuración de la monitorización** y un **prototipo del Middleware de Monitorización**.

Estos prototipos se han realizado siguiendo como requisitos la propuesta del proceso de monitorización de manera que se pudiera instanciar en forma de herramientas. Tanto el proceso como las herramientas han ido refinándose durante el desarrollo del proyecto por lo tanto cambiando, adaptando y desechando algunos requisitos conforme el enfrentamiento con la realidad de la implementación lo requiera o creando nuevos cuando las posibilidades de la plataforma ofrecieran esa posibilidad.

La implementación de estos prototipos requería de una plataforma en la nube debido a que la instancia del Middleware de Monitorización requiere de alta disponibilidad dado que este proceso debe estar activo de forma continua para que el procesamiento de resultados aportados en tiempo real por el servicio a ser monitorizado se realice con el mínimo retraso posible de manera que situaciones de riesgo sean detectadas a tiempo.

La plataforma en la nube escogida para tal efecto ha sido Microsoft Azure, una destacada plataforma como servicio (PaaS) que permite la publicación de aplicaciones web y otro tipo de servicios. Cualquier otra plataforma hubiera servido para tal efecto pero Microsoft Azure presentaba una serie de ventajas para la realización del proyecto. En primer lugar, según Business Insider, Microsoft Azure es la segunda plataforma con mayor tamaño de mercado (un 10%, por detrás de los servicios de Amazon que ostentan el 30% del mercado) y es la plataforma de cloud que más rápido ha crecido en el período que va hasta 2015 [1]. Por lo que es una plataforma representativa en el mercado cloud, bien extendida y que podrá ser utilizada como representativa del resto de plataformas en la nube. Segundo, Microsoft ofrece además de los servicios de plataforma, recursos software (librerías, entornos de desarrollo y otras tecnologías) propios que facilitan el desarrollo en la nube, siendo muy importante la librería *Diagnosics* para la realización de este proyecto ya que permitía extraer datos de la plataforma sin tener que recurrir a diseñar un sistema de extracción dependiente de la plataforma completo (objetivo que ocuparía un trabajo aparte) o recurrir a soluciones de terceros que podrían hacer que la solución fuera poco extensible. Por último, el uso de tecnologías en la nube puede presentar un coste económico importante; sin embargo, Microsoft ofrece la posibilidad de usar

licencias académicas para la investigación en tecnologías en la nube como es el caso del presente trabajo. Además, se ha conseguido acceso a las herramientas de Azure mediante un “Microsoft Azure Research Award Program”.

El ecosistema de la plataforma ha determinado el uso de las herramientas para la construcción del software de este trabajo. Se ha utilizado como entorno de desarrollo Visual Studio pues posee las herramientas necesarias para desplegar y utilizar los recursos disponibles en Windows Azure de forma sencilla. El lenguaje de programación elegido ha sido C#, un lenguaje de programación multiparadigma desarrollado por Microsoft como parte del entorno .Net que ofrece la potencia y versatilidad necesaria para llevar a cabo un trabajo de este tipo y además es el lenguaje preferido por Microsoft Azure para desarrollar en la plataforma, ofreciendo toda su documentación en este lenguaje y gran parte de las librerías. En el desarrollo de la aplicación web se ha empleado ASP.NET [3] como *framework* con lenguaje en el servidor C#, mientras en la parte cliente se ha utilizado Javascript y para realizar las vistas HTML y CSS utilizando la librería de estilos Bootstrap [3]. La elección de ASP.NET como *framework* ha estado motivada por utilizar C# como lenguaje en el lado del servidor ya que ofrecía la potencia necesaria para realizar acciones como extraer información en formatos XML y trabajar con modelos de objetos complejos. Esta ha sido desplegada a su vez como servicio alojado en la nube de Microsoft Azure de manera que se facilita la comunicación entre los servicios y se aprovecha la homogeneidad de la tecnología. Bootstrap es un *framework* de HTML, CSS y JS que aporta librerías que permiten realizar sitios web de diseño responsivo, una condición deseable para cualquier proyecto web.

5.2 Microsoft Azure

Dado que la implementación tiene una estrecha relación con la plataforma en la nube en la que está siendo construida, así como con el tipo de servicios que permite monitorizar en este prototipo, es conveniente entrar en detalle en este aspecto en algunos de los aspectos de la plataforma, en especial las librerías de monitorización utilizadas y su funcionamiento.

La plataforma Microsoft Azure fue anunciado por primera vez en el evento PDC (Professional Developers Conference) en 2008 como Windows Azure Platform y lanzado en 2010 como Windows Azure es un servicio IaaS y PaaS ofrecido por Microsoft para construir, desplegar y gestionar aplicaciones y servicios alojados en los centros de datos de Microsoft .

Este servicio es capaz de soportar distintos lenguajes de programación, herramientas y servicios, incluyendo los pertenecientes a terceros así como un ecosistema creado por Microsoft basado en .Net.

Sobre esta plataforma se desplegarán los servicios de monitorización y la página web que componen este proyecto.

Dentro de este entorno, las librerías más relevantes empleadas por este proyecto son las de Azure Diagnostics, librerías creadas para la monitorización de servicios en la nube y en particular los datos que estas emplean, los Performance Counters.

5.2.1 Azure Diagnostics y Performance Counters

Los datos utilizados para la monitorización se han proveído por el software de *Microsoft Azure Diagnostics* [5]. Este software y conjunto de APIs se encarga de “capturar datos del sistema y de *logs* procedentes de las máquinas virtuales y las instancias de las máquinas virtuales que se ejecutan en un servicio cloud de Azure y transfieren esos datos a una cuenta de

almacenamiento de la elección del usuario”. Es por ello el encargado de recuperar los datos de rendimiento de bajo nivel procedentes de los servicios. Estos datos están modelados en forma de instancias de la clase llamada *Performance Counter* [6]. Un Performance Counter contiene datos relativos a únicamente una métrica de bajo nivel. Estos están clasificados por un nombre que representa la ruta de la procedencia de esta información. En la tabla abajo mostrada aparece una pequeña porción de los datos disponibles en un servicio ejecutándose en Windows Azure:

Nombre	Definición	Categoría	Instancia
# Exceps Thrown / sec	Número de excepciones lanzadas por segundo. Contando aquellas manejadas y no manejadas.	Exception Performance Counters	.NET CLR Exceptions(_Global_)
Application Restarts	Número de veces que una aplicación ha sido reiniciada durante la vida del servidor.	Performance Counters for ASP.NET	ASP.NET
Requests Total	Número total de peticiones respondidas desde el inicio del servidor.	Performance Counters for ASP.NET	ASP.NET Applications(__Total__)
% Processor Time	Porcentaje del tiempo de procesador utilizado por el servicio.	Performance Counters for ASP.NET	Processor(_Total)
Requests/Sec	Número de peticiones ejecutadas por segundo.	Performance Counters for ASP.NET	ASP.NET Applications(__Total__)
Connection Failures	Número de veces que las conexiones TCP han hecho una transición directa desde el estado CLOSED al de SYN-SENT o SYN-RCVD, más el número de veces que las conexiones TCP han realizado una transición desde LISTEN al estado de SYN-RCVD	TCP Object	TCPv4

Como se puede comprobar y ya se ha comentado en secciones anteriores, la información disponible por este método es limitada y de bajo nivel, casi por completo relacionada con aspectos de infraestructura (%ProcessorTime representando el porcentaje del tiempo de procesador en uso por la instancia) o de plataforma (Exceps Thrown/sec, representando el número de excepciones lanzadas por los servicios ejecutándose en el sistema operativo). Debido a la fuerte conexión entre las librerías de Microsoft Azure y las aplicaciones realizadas en el entorno de .NET estos Performance Counters también son capaces de conseguir algunos datos relativos a la aplicación (Application Restarts, que posee el número de veces que la aplicación se ha reiniciado o Requests/sec que representa el número de peticiones que la aplicación ha recibido por segundo). Estos contadores de forma aislada pueden representar el estado del rendimiento de la aplicación, pero sin un tratamiento a más alto nivel es complicado que un usuario pueda comprender qué está ocurriendo en el sistema e incluso el propio software de manera automática no podría comprender estos datos pudiendo utilizarlos para mejorar su rendimiento.

La semántica del dato aportado por el contador depende de cada contador individualmente, ya que cada uno aporta una información que debe ser interpretada de manera diferente según las características del mismo. En primer lugar se debe distinguir entre el tipo de dato que ha sido aportado por el contador, en ocasiones este puede interpretarse como un porcentaje o un número



real, así como las unidades en las que se expresa cada uno. En segundo lugar, también es necesario comprender la forma de extracción, pues dará datos acerca de cómo tratar el valor, en este trabajo se han distinguido entre dos tipos: valores totales y valores individuales. Los contadores que poseen valores totales (p.e., *Requests Total*, Peticiones totales hechas al servicio) poseen un valor que se va acumulando e incrementando según se cumplen ciertas circunstancias. De manera que según se avance en el tiempo los valores tomarán siempre valores mayores o iguales que en el momento anterior (siempre que no se dé un reinicio de la aplicación). Esto crea gran número de datos redundantes en casos en los que baste conocer el valor final, por ejemplo en el caso de *Requests Total* si el usuario para dar valor a una métrica solo quiera conocer el número de peticiones totales en la última hora, no es necesario conocer el número de peticiones totales en cada minuto de ese intervalo, solamente la última medida tomada en ese intervalo. Por el contrario, otros *Performance Counters* ofrecen datos de forma individual, por ejemplo, *Application Request Time* (tiempo de respuesta de una petición a la aplicación) obtiene en cada medida el tiempo de respuesta de la última petición realizada al servicio. De esta manera, obtendremos una dispersión de valores que deberán ser tratados de forma estadística, por ejemplo obteniendo medias o medianas, para resultar de utilidad.

Una de las características de los servicios de monitorización en la nube es la capacidad de generar grandes cantidades de datos de manera periódica. Por razones de consumo de memoria y banda ancha así como para facilitar su comprensión necesitan ser tratados para ser de utilidad. Los ciclos de extracción de estos datos en ocasiones necesitan ser muy cortos (del orden de segundos) de manera que puedan detectarse correctamente ciertas situaciones sensibles al tiempo, por ejemplo tiempos de respuesta excesivamente largos solo podrán controlarse con el contador *Request Time* si el valor de este se extrae en cortos espacios de tiempo pues solo obtiene el valor de la última petición respondida de manera que el tiempo de extracción pueda ocultar este tipo de resultados si resulta ser demasiado largo.

De igual manera, aunque el coste de almacenamiento en la nube se encuentra en descenso [7] y es relativamente barato, una política inconsciente en el tratamiento del volumen de datos puede generar grandes costes computacionales y de almacenamiento sin aumentar la calidad de los resultados obtenidos al trabajar con demasiadas muestras redundantes que no aportan valor. El tiempo de extracción y el mejor tratamiento de datos merecen estudio a parte por parte de las disciplinas del *Data Science* pero son vitales para realizar una configuración precisa de la monitorización de los servicios en la nube.

Diagnosics resuelve en parte algunos de los problemas presentados. Esta librería permite a través de una serie de componentes configurar de forma programática así como manual los contadores que se extraen del servicio, el tiempo de extracción de los mismos y la cuenta de almacenamiento de Azure Storage donde se guardarán dichos valores.

Observando los posibles valores extraídos y las facilidades que ofrece la plataforma se ha decidido utilizar este como el motor principal de la extracción de datos. Diagnosics no es un caso especial dentro de las plataformas en la nube, las principales como Amazon Cloud Services o Google Cloud Platform también ofrecen mecanismos propios de extracción de datos de forma similar como se ha visto en el Capítulo 2 de este trabajo.

5.3 El Configurador de la Monitorización

El configurador de la monitorización es una aplicación web cuyo objetivo es asistir al usuario durante la tarea de configuración de la monitorización del proceso monitorización. Esta herramienta da soporte a las distintas subtareas que componen la Configuración de la Monitorización presentadas en el apartado 5.2.1 cuyo objetivo final es la elaboración del modelo en tiempo de ejecución que será empleado para realizar la monitorización del servicio en la nube objetivo.

Para conseguir este objetivo, se ha optado por la implementación de una aplicación web dado que una característica propia de un proyecto cloud es la capacidad de acceso desde cualquier parte del mundo al servicio, por lo que una herramienta que de soporte a un servicio de estas características es recomendable que comparta esta propiedad, en contra de las características de una posible versión de escritorio. Así lo demuestra la tendencia actual de trasladar las aplicaciones de escritorio a la nube, incluso una herramienta tan especializada como la presente. Además, la tecnología web hace posible el uso de la aplicación desde cualquier dispositivo.

Según las subtareas de configuración de la monitorización la aplicación debería ser capaz de:

- Recoger los Requisitos de Calidad para la monitorización especificados por un planificador de la monitorización.
- Ser capaz de contener un modelo de calidad SaaS que ofrezca asistencia al usuario para realizar la configuración.
- Clasificar los requisitos de monitorización relacionándolos con los atributos de calidad presentes en el modelo de calidad SaaS.
- Seleccionar de entre las métricas del modelo de calidad y asociar a cada atributo de calidad asociado a un requisito de monitorización una métrica.
- Ser capaz de expresar las métricas procedentes del modelo de calidad SaaS utilizando las propiedades específicas de la plataforma.
- Generar un modelo en tiempo de ejecución que será consumido por el Middleware de Monitorización para realizar la configuración de la monitorización del servicio.

5.3.1 Patrón de diseño

El configurador está organizado basado en el patrón de diseño de interfaces de usuario del Asistente (Wizard) [4], este patrón tiene como objetivo dividir una tarea en subtareas más pequeñas de manera que sea más sencillo para el usuario completar la tarea principal, en este caso crear el modelo de la monitorización en tiempo de ejecución.

Tomando como guía el diagrama del proceso de configuración de la monitorización resulta evidente el carácter lineal del proceso de generación del modelo. Así mismo, el proceso es complejo y largo debido al tipo y número de decisiones que el usuario tiene que tomar por lo que disminuir la complejidad de las interfaces es una prioridad de primer nivel. De igual manera, debido a la complejidad de la acciones a realizar y de los conceptos empleados, es necesario que al usuario se le guíe a través de la aplicación indicándole las acciones que debe tomar a continuación.

Middleware para la monitorización de la calidad de servicios cloud

Así mismo, la posibilidad de incluir la totalidad del proceso en una sola interfaz fue puesto a prueba en versiones preliminares (véase 5) y se demostró que la cantidad de elementos y posibilidades podría crear confusión al usuario y reducir las funciones de la aplicación con lo que quedó descartada.

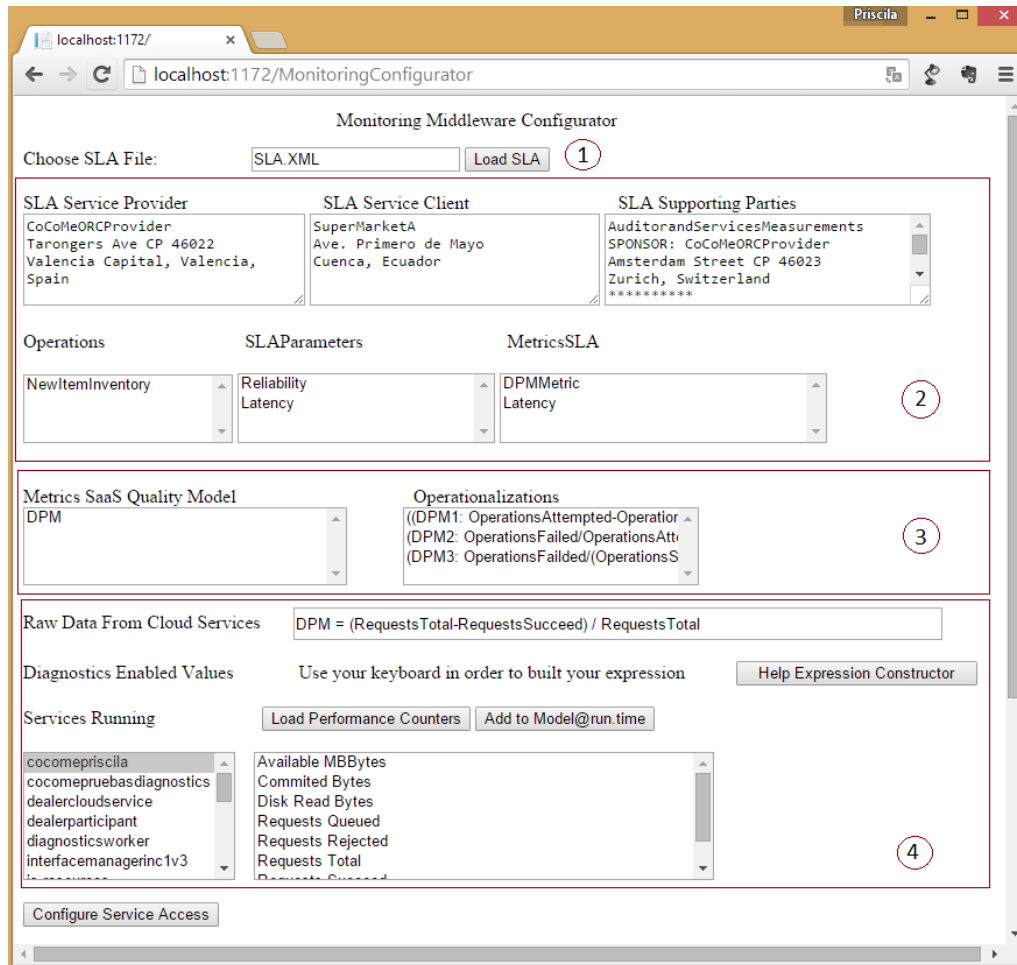


Figura 5 – Prototipo del Configurator de la Monitorización.

El diseño final de la aplicación presenta el habitual diseño de interfaz presente en aplicaciones que utilizan este patrón, ocupando la parte central de la pantalla los formularios a rellenar y situando en la parte inferior los botones de avance y retroceso para navegar entre tareas como puede verse ilustrado en la Figura 6.

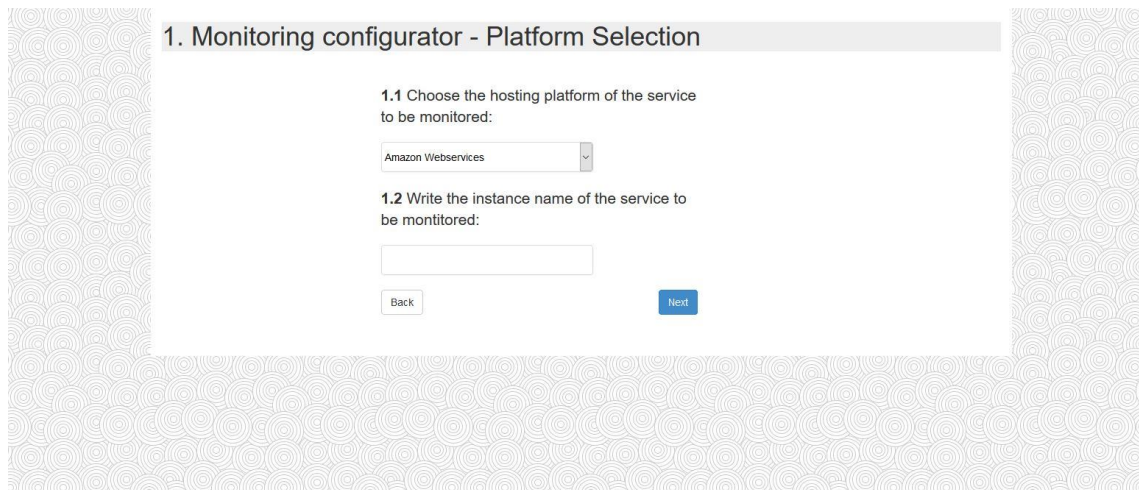


Figura 6 – Ejemplo de interfaz con el patrón Asistente.

A continuación, se procede a realizar un recorrido guiado de la aplicación detallando los puntos más importantes de la misma. El flujo de la aplicación puede verse ilustrado en la siguiente Figura:

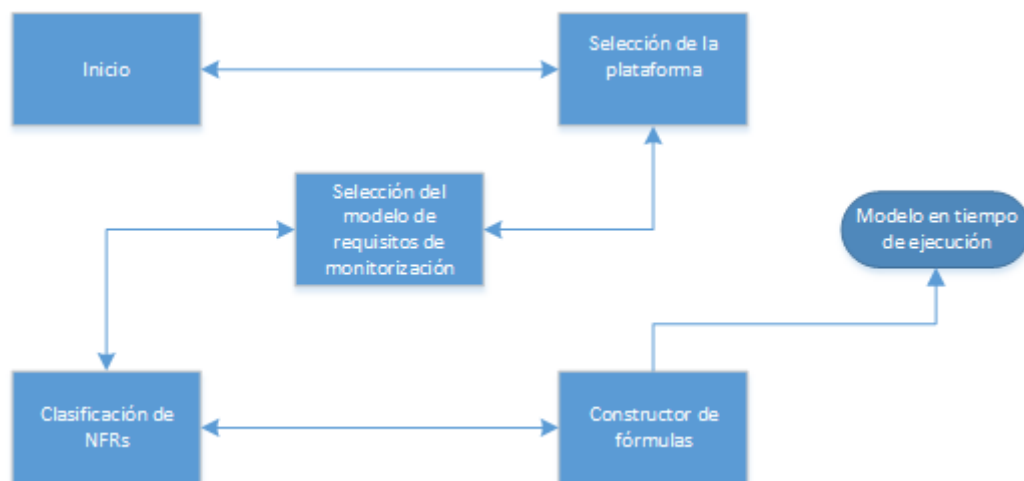


Figura 7 – Flujo de la aplicación web

5.3.2 Selección de la plataforma

En esta página (ilustrada en la Figura 8) el usuario debe introducir los datos del servicio que desea monitorizar. Para ello seleccionará la plataforma en la que el servicio está desplegado de entre las soportadas por la aplicación, en este prototipo se da soporte únicamente a Windows Azure, dado el conocimiento profundo de la plataforma que se requiere para realizar una configuración de estas características quedando fuera del alcance. A continuación se deben introducir las credenciales propias del servicio: su nombre y, en el caso de Microsoft Azure, el identificador del despliegue y la cadena de conexión. El identificador del despliegue determina únicamente el servicio dentro de la plataforma, mientras que la cadena de conexión incluye la información necesaria para acceder a los recursos de almacenamiento de Azure Storage programáticamente, de esta manera se concede el acceso, necesario, a la aplicación para realizar operaciones de comunicación con los datos de monitorización del servicio y el Middleware de Monitorización.

1. Monitoring configurator - Platform Selection

1.1 Choose the hosting platform of the service to be monitored:

Windows Azure

1.2 Write the instance name of the service to be monitored:

1.3 Enter the Azure Service credentials:

Deployment ID

Connection String

Back Next

Figura 8 – Selección de la Plataforma.

5.3.3 Selección del modelo de requisitos de monitorización

El modelo de requisitos de monitorización es la salida de la tarea 1 del proceso de configuración de la monitorización, el cual se realiza fuera de los límites de la aplicación. Este documento contiene la información relativa a los requisitos no funcionales a monitorizar que será necesaria para establecer las métricas capaces de medir estos requisitos.

En esta implementación, cuyo diseño se observa en la Figura 9, se ha optado por aceptar este modelo realizado en formato XMI (XML Metadata Interchange, XML de Intercambio de Metadatos) un formato estándar ISO/IEC 19509:2014 que tiene como objetivo “permitir el fácil intercambio de metadatos entre aplicaciones del proceso de vida del desarrollo (como herramientas de modelado basadas en Unified Modeling Language (UML) y repositorios de metadatos o *frameworks* basados en Meta Object Facility (MOF) en entornos distribuidos y heterogéneos”.

2. Monitoring configurator - Monitoring Requirements Model Selection

Upload the Monitoring Requirements Model in xmi format:

Examinar... MonitoringRequirementsModel.xml

Upload Model

Back

Figura 9 – Selección del modelo de requisitos de monitorización.

Aunque en esta ocasión el empleo de XMI está ligeramente desviado de su intencionalidad, pues en esta aplicación se utiliza como contenedor de los datos estructurados de requisitos de monitorización, se ha creído conveniente su uso dado que la aplicación debe trabajar conjuntamente con otras que modifiquen este tipo de modelos. En este caso los modelos son generados desde Eclipse a falta de una herramienta propia para el diseño de los mismos.

Una vez importado el modelo haciendo clic en Examinar y seleccionándolo en el explorador, si se ha importado correctamente en la parte inferior aparece una tabla con la información, resumida, más relevante para el usuario (ilustrado por la Figura 10):

- **NFR Name:** Nombre del requisito no funcional.
- **Threshold:** Expresión del umbral, mayor, menor o igual que el requisito no funcional debe cumplir por acuerdo de los stakeholders. Puede estar expresada como un porcentaje o un número decimal.
- **Metric Name:** Nombre de la métrica o métricas elegidas como posibles candidatas a medir el RNF asociado.
- **Metric Formula:** Fórmula, en lenguaje natural que expresa la forma en la cual se puede medir una métrica. Una métrica puede medirse con una o más fórmulas.

2. Monitoring configurator - Monitoring Requirements Model Selection

Upload the Monitoring Requirements Model in xmi format:

Examinar... No se ha seleccionado ningún archivo.

Upload Model File uploaded!

This is the uploaded Monitoring Requirements Model:

NFR	Threshold	Metric Name	Metric Formula
GuaranteeOfReliability	>99.999	DPM	$((\text{OperationsAttempted} - \text{OperationsSuccessful}) / \text{OperationsAttempted}) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
GuaranteeofAvailability	>99.995	Service Robustness	(available time for invoking SaaS) / total time for operating SaaS
Latency	<130	Latency	$\text{sum}(\text{Request Response Time}) / \text{Number of Requests}$
Stability	>99.999		
Service Accuracy	~100%	DPM	$((\text{OperationsAttempted} - \text{OperationsSuccessful}) / \text{OperationsAttempted}) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
Transactions per Hour	=10000	Transactions Per Hour	TransactionsPerHour

Back Next

Figura 10 – Modelo de requisitos de la monitorización cargado en la aplicación.

5.3.4 Clasificación de requisitos no funcionales

En esta interfaz, el usuario tiene por objetivo seleccionar y clasificar los requisitos no funcionales a monitorizar usando un modelo de calidad SaaS como guía. Para ello el usuario deberá seleccionar uno por uno los requisitos no funcionales, clicando sobre una tabla idéntica a la descrita en el punto anterior situada bajo la explicación del paso 3.1 (véase en la Figura 11). Una vez seleccionado, aparecerá en azul el nombre de la métrica seleccionada y se procederá a

realizar la clasificación en el paso 3.2. En este se procederá a realizar la clasificación dentro de las ramificaciones del modelo de calidad SaaS, tal y como indica la segunda tarea del modelo de proceso, seleccionando para ello primero la característica a la que pertenece de la lista situada bajo el título *Characteristic*, la subcaracterística (en el caso de que sea necesario y exista una para la característica seleccionada aplique) navegando en el árbol de subcaracterísticas, el atributo y finalmente la métrica en sus respectivos controladores (Figura 12).

3. Monitoring configurator - Mapping Selection

In this step the mapping between the Non Functional Requirements from the Monitoring Requirements Model must be classified using de SaaS Quality Model and added to the Runtime Model which will be used to monitor de service. The specification of the formula to calculate these requirements will be created in the next step.

3.1 Select the Non Functional Requirement to be monitored:

Non Functional Requirements

NFR Name	Threshold	Metric Name	Metric Formula
GuaranteeOfReliability	>99.999	DPM	$((OperationsAttempted - OperationsSuccessful) / OperationsAttempted) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
GuaranteeofAvailability	>99.995	Service Robustness	(available time for invoking SaaS) / total time for operationg SaaS
Latency	<130	Latency	sum(Request Response Time) / Number of Requests
Stability	>99.999		
Service Accuracy	=100%	DPM	$((OperationsAttempted - OperationsSuccessful) / OperationsAttempted) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
Transactions per Hour	=10000	Transactions Per Hour	TransactionsPerHour

Figura 11 – Asociación de Métricas: Los RNFs.

El modelo de Calidad SaaS ya está introducido internamente como datos de la aplicación y de igual manera que el Modelo de Requisitos de Monitorización está portado en formato XMI por lo que puede ampliarse o modificarse sin modificar la implementación. Se ha utilizado un modelo de calidad SaaS prototipo ya que la creación de un modelo de calidad SaaS completo se encuentra en el momento de escribir este trabajo en fase de investigación.

Selected NFR from the table:
Selected: Latency

3.2 Map the selected NFR to its appropriate metric from the SaaS Quality Model

To do so, navigate through its corresponding characteristics, subcharacteristics(if it applies) and attributes.

Characteristic	Subcharacteristic	Attribute	Metric
Performance Efficiency	Time Behaviour Resource Utilization Capacity	Response Time	Latency

Select the operationalization which will calculate the metric:

Operationalization	Platform Independent Operationalization
Latency	Request Execution Time

[Add to Model @ Runtime](#)

Figura 12 – Asociación de Métricas: Clasificación en el modelo de calidad.

En el momento de realizar la clasificación, el usuario deberá tener conocimiento teórico sobre este modelo de calidad y deberá ser capaz de, tomando los datos del modelo de requisitos , ser capaz de encontrar la clasificación más apta para los RNFs usando los conceptos del modelo de calidad.

Una vez hecha la clasificación, siguiendo la subtarea tres del modelo de proceso, la Selección de Métricas, se procede a seleccionar la operacionalización del controlador para ello habilitado bajo el título Operationalization. Esta operacionalización describe la forma de medir

la métrica de manera general, en términos independientes de la plataforma. Si la clasificación es correcta, se procederá a añadirla al modelo en tiempo de ejecución resumido en la tabla bajo el título: “Selected metrics added to the Model@Runtime”. En caso de haber un error en la clasificación en este mismo apartado puede seleccionarse la métrica erróneamente clasificada clicando en el Checkbox situado en su fila y pulsando sobre Delete.

Una vez terminada la clasificación de todos los requisitos no funcionales a monitorizar se procede al siguiente paso pulsando siguiente. En la tabla situada en la sección del modelo en tiempo de ejecución (Figura 13) aparecerá entonces una nueva fila conteniendo la información de la clasificación:

- **#NFR:** Número de requisito no funcional. Identifica al requisito no funcional de forma única.
- **Attribute Name:** Nombre del atributo bajo el cual se ha clasificado el RNF. No es necesaria la información pertinente a las características y subcaracterísticas dado que cada atributo es la hoja del árbol de características por lo que la clasificación ya las incluye. No debe haber dos atributos de mismo nombre en el modelo de calidad SaaS.
- **Metric Name:** Nombre de la métrica en la cual se ha clasificado el RNF.
- **Operationalization Name:** Nombre de la operacionalización, en términos independientes de la plataforma, que ha sido elegida para medir la métrica.



Figura 13 – Asociación de métricas: El modelo en tiempo de ejecución.

5.3.5 Constructor de fórmulas

Esta interfaz da soporte a la última parte de la cuarta subtarea del modelo: la Asociación de Métricas. Una vez clasificadas y asignada una operacionalización independiente de la plataforma a la métrica asociada a un requisito no funcional, el usuario debe de escribir esta operacionalización usando términos propios de la plataforma.

En el caso del presente prototipo, eso implica traducir las fórmulas en lenguaje natural en fórmulas especificando los *Performance Counters* procedentes de *Azure Diagnostics* apropiados o en su defecto los contadores personalizados creados con este propósito. Esta fórmula será la que el motor de medición interpretará para extraer la información de rendimiento y realizar las operaciones dispuestas en ella y que constituirán una medida de un atributo de calidad.

De esta manera el configurador da soporte a tres de los cuatro escenarios definidos en los escenarios de extracción de datos vistos en el apartado 5.5.2 de este trabajo.

Para realizar este proceso primero se debe seleccionar uno de los RNF expuestos en la tabla bajo el apéndice 4.1 clicando sobre el *Checkbox* de la fila del RNF deseado. Y pulsar el botón Edit. (Ver Figura 14).

En esta tabla aparece la siguiente información:

- **#NFR:** Número de requisito no funcional. Identifica al requisito no funcional de forma única.
- **Attribute Name:** Nombre del atributo en el cuál este RNF se ha clasificado.
- **Metric Name:** Nombre de la métrica en la cual se ha clasificado el RNF.
- **Platform Independent Operationalization:** Operacionalización seleccionada formulada en términos independientes de la plataforma.

4. Monitoring configurator - Formula Builder

In this step the platform independent operationalization of the selected metrics to be monitored must be associated to a platform dependent operationalization. Select each one of the NFRs, build the formula corresponding with the platform independent operationalization and add them to the Model@Runtime. When you are ready generate the model.

4.1 Choose the NFR:

NFR #	Attribute Name	Metric Name	Platform Independent Operationalization	
0	Service Accuracy	Service Accuracy (SA)	Number of Correct Responses / Total Number of Requests	<input checked="" type="checkbox"/>

Edit

Selected NFR from the table:
SelectedService Accuracy

4.2 Create the formula

Combine platform dependent counters, arithmetic symbols and/or custom counters to make a formula which maps with the operationalization defined for the NFR. When the formula is ready, add it to the Model @ Runtime.

Formula Editor

Performance Counters Calculator Wrapper Counters

\\.NET CLR Networking 4.0.0.0(*)Connections Established
\\.NET CLR Networking 4.0.0.0(*)Bytes Received
\\.NET CLR Networking 4.0.0.0(*)Bytes Sent
\\.NET CLR Networking 4.0.0.0(*)Datagrams Received

Extraction type
Average

Extraction rate

Undo last step

Add to Formula

Add to Model@Runtime

Figura 14 – Constructor de fórmulas.

Una vez seleccionado el RNF se procederá al segundo apartado del formulario (4.2) dónde el usuario seleccionará de entre los *Performance Counters* de *Microsoft Azure*, los términos de la calculadora (símbolos y números naturales) y los *Custom Counters* (Contadores Personalizados) disponibles para realizar la fórmula dependiente de la plataforma tomando como referencia la fórmula independiente tomada del modelo de calidad SaaS y que puede observarse en la tabla del apartado 4.1 (Figura 14).

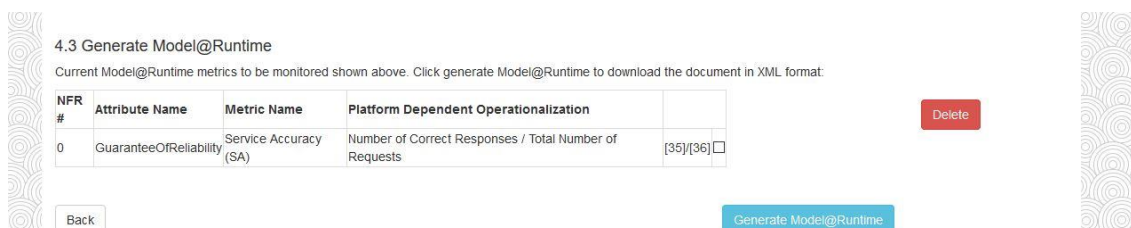
Para añadir un contador (ya sea un *Performance Counter* o un *Custom Counter*) deberá seleccionar la opción del menú clicando en el título de manera que aparezca la vista con las opciones de los contadores. En ella se deberá elegir un contador de la lista clicando sobre él y:

1. Seleccionar un tipo de extracción (Extraction Type): En el prototipo puede escogerse entre media (se realizará una media de los valores obtenidos entre cálculos de la métrica) y total (se escogerá el último valor obtenido).
2. Seleccionar el ratio de extracción (Extraction Rate): Este es un valor numérico (en segundos) en el cuál se extrae un dato del contador seleccionado del servicio.
3. Pulsar Add To Formula: El contador aparecerá en el cuadro de texto de la fórmula y podrá elegirse otro ítem para continuar con la construcción de la fórmula siguiendo el mismo proceso.

Más detalles sobre los tipos y funcionamiento de los contadores de rendimiento y sus propiedades pueden consultarse en el apartado 6.3.1.

En el caso de producirse errores al introducir nuevos elementos en la fórmula puede deshacerse el último elemento añadido pulsando el botón *Undo last step*. En caso de que la fórmula ya esté completada, puede añadirse al Modelo en tiempo de ejecución pulsando *Add to Model@Runtime*. Al pulsar, una nueva fila aparecerá en la tabla localizada en el punto 4.3 (Figura 15) con la siguiente información:

- **#NFR**: Número de requisito no funcional. Identifica al requisito no funcional de forma única.
- **Attribute Name**: Nombre del atributo en el cuál este RNF se ha clasificado.
- **Metric Name**: Nombre de la métrica en la cual se ha clasificado el RNF.
- **Platform Dependent Operationalization**: Fórmula construída en los pasos anteriores con elementos dependientes de la plataforma.



4.3 Generate Model@Runtime

Current Model@Runtime metrics to be monitored shown above. Click generate Model@Runtime to download the document in XML format.

NFR #	Attribute Name	Metric Name	Platform Dependent Operationalization	
0	GuaranteeOfReliability (SA)	Service Accuracy (SA)	Number of Correct Responses / Total Number of Requests	[35]/[36] <input type="checkbox"/>

Buttons: Back, Delete, Generate Model@Runtime

Figura 15 – Constructor de fórmulas: Resultado en el modelo en tiempo de ejecución.

Este proceso debe repetirse con todos los RNF que quieran ser monitorizados. Una vez todas las operacionalizaciones dependientes de la plataforma se hayan cargado en el modelo en tiempo de ejecución pulsando sobre *Gerate Model@Runtime* este será mandado al monitor para inicializar la monitorización y una copia en XML podrá ser descargada. Un ejemplo de este modelo puede encontrarse en el Anexo 1.1.

5.4 El Middleware de Monitorización

El Middleware de monitorización es la pieza de software encargada de dar soporte a la segunda tarea del proceso de monitorización denominada *Middleware Process* expuesta en la sección 5.4.2 de este documento. Esta herramienta es la encargada de: (1) recibir el modelo en tiempo de ejecución y configurar el servicio para su monitorización, (2) extraer los datos de monitorización utilizando los diferentes escenarios de extracción descritos en la sección 5.5.2, (3) realizar los cálculos descritos por las operacionalizaciones de las métricas en el modelo en tiempo de ejecución y (4) almacenar los resultados de las métricas en un sistema de almacenamiento permanente para su posterior consulta.

En este trabajo se ha instanciado un prototipo del middleware de monitorización en la plataforma Microsoft Azure para la monitorización de servicios desplegados en la plataforma Microsoft Azure contemplando los escenarios primero, segundo y tercero de la extracción de datos descritos en la sección 5.5.2. En esta sección se procederá a exponer los requisitos y características del Middleware de monitorización así como su implementación y funcionamiento.

5.4.1 Requisitos y características del middleware de monitorización

Al tratarse de un proyecto de investigación, los requisitos funcionales de este prototipo software han ido refinándose de forma iterativa conforme la investigación sobre el proceso de monitorización de los servicios en la nube avanzaba.

Sin embargo, en la concepción de este prototipo debían tenerse en cuenta los siguientes requisitos no funcionales:

- **Adaptabilidad y extensibilidad:** El monitor debe ser capaz de dar soporte a la monitorización de otros servicios desplegados en otras plataformas, sin que el proceso de monitorización cambie. Extendiendo componentes periféricos si fuera necesario.
- **Interoperabilidad:** El monitor debe ser capaz de interactuar con los servicios desplegados en Microsoft Azure para obtener sus datos así como interactuar con el configurador de la monitorización.

Así mismo el prototipo ofrece los siguientes requisitos funcionales:

- **Configuración de la monitorización:** El monitor debe utilizar el modelo en tiempo de ejecución, configurando si fuese necesario los servicios objetivos para la extracción de los datos de monitorización.
- **Extracción de datos:** El monitor debe ser capaz de extraer los datos correspondientes a los tres primeros escenarios de extracción de datos especificados.
- **Cálculo de métricas:** El monitor debe ser capaz de calcular las fórmulas expresadas por las operacionalizaciones de las métricas en el modelo en tiempo de ejecución.
- **Almacenamiento de medidas:** El monitor debe ser capaz de almacenar las medidas calculadas en una base de datos persistente de manera que puedan ser utilizadas posteriormente en la tarea de Análisis para la generación de reportes.
- **Actualización en tiempo de ejecución:** El monitor debe ser capaz de actualizar los parámetros de su configuración en cualquier momento de su ejecución recibiendo un nuevo modelo en tiempo de ejecución.

5.4.2 Arquitectura

El prototipo desarrollado presenta la arquitectura descrita por el diagrama de clases de la Figura 16:

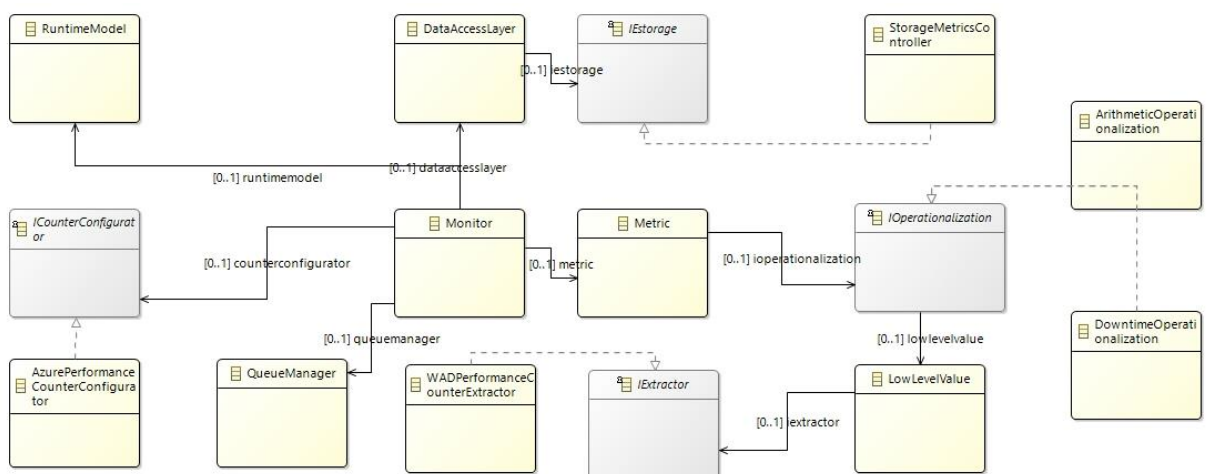


Figura 16 – Arquitectura del Middleware de Monitorización.

A continuación, se realizará una descripción de alto nivel de los componentes descritos en el diagrama de clases del Middleware de Monitorización:

Monitor

Clase encargada del ciclo de vida del Middleware de Monitorización. Se ocupa de recoger los datos crudos, realizar las operaciones de las métricas, guardar los datos y realizar las actualizaciones en el momento oportuno. Es la parte central de la aplicación.

Metric

Representación de una métrica medible a través de una operacionalización. Una vez conseguido el valor de esta, son el output del proceso de medición conteniendo la medida obtenida en cada iteración.

IOperationalization

Interfaz que define los métodos para el cálculo de las operacionalizaciones. Es necesario realizar una especialización entre operacionalizaciones ya que no todas tendrán un carácter aritmético (aunque si serán las más generales) y es probable que otras requieran de cálculos más complejos. Por ejemplo en el caso del *Downtime* (Tiempo de servicio inaccesible), no existe una métrica válida expresada por el software *Diagnostics* y para su cálculo es necesario el uso de condicionales y guardar valores históricos para realizar comparaciones. Para facilitar el acople de este tipo de implementaciones se ofrece esta interfaz.

Arithmetic Operationalization

Implementación de la interfaz *IOperationalization*. Esta clase se encarga de realizar cualquier cálculo de métricas cuya operacionalización pueda ser expresada en una fórmula aritmética. Esta implementación ayudada por la librería *NCalc* [34] es la encargada realizar los cálculos definidos por las fórmulas de la operacionalización de las métricas y de mandar al Extractor la extracción de los valores necesarios para dicho cálculo. El módulo *NCalc* permite el uso de fórmulas que en un formato propio de la librería permite calcular dinámicamente cualquier tipo de fórmula con variables no asignadas estáticamente, por lo que añade flexibilidad a la hora de crear y aplicar fórmulas del modelo en tiempo de ejecución.

DowntimeOperationalization

Implementación de la interfaz *IOperationalization* para realizar una medida personalizada como las que se mencionaban en la descripción de la interfaz. Esta operacionalización del atributo de disponibilidad no podía realizarse de forma aritmética debido al tipo de datos sobre el estado del servicio disponibles en *Azure Diagnostics*. En este caso ha sido necesario realizar una labor de detección de intervalos en los cuáles el servicio está caído (detectado por el reinicio del contador de *Uptime*) y a partir de ahí calcular el tiempo aproximado en el cuál el servicio ha estado caído (la diferencia entre la marca de tiempo del último contador de *Uptime* mayor que cero y el primer contador de *Uptime* vuelto a recibir).

LowLevelValue

Representación de los datos de bajo nivel obtenidos por los medios de recolección de datos de los servicios. Cada uno de estos valores son empleados por las fórmulas de las operacionalizaciones para llevar a cabo los cálculos. Lleva asociada una fuente de datos del cual son extraídos periódicamente.

IExtractor

Interfaz que define los métodos de recolección de datos procedentes de los extractores de datos crudos de monitorización. Esta interfaz posibilita el uso de distintos extractores de datos para datos procedentes de distintas fuentes. De esta manera distintos tipos de extractores pueden trabajar en conjunción con distintos tipos de datos que pueden utilizarse durante el mismo proceso de monitorización o incluso en una misma métrica formada por datos procedentes de distintas fuentes.

IExtractor

Interfaz que define el comportamiento de un extractor de datos de monitorización de una fuente de datos. Utilizado para desacoplar de la implementación el uso de uno o varios adaptadores. Establece dos tipos de muestreo de datos, como totales o como media. Esto es así dado que el elevado número de muestras de datos que puede existir en la fuente no es necesario para obtener medidas fieles ya que si se ha definido un tiempo de recolección de estas métricas muy reducido en los contadores acumulativos obtendremos muchos valores casi idénticos por lo que es recomendable solamente trabajar con totales o también este elevado número de muestras carece sentido a tan baja granularidad y es preferible trabajar con medias en intervalos cortos de tiempo (3-30s) que aportarán mayor significado y relevancia a la métrica.

WADPerformanceCounterExtractor

Implementación de *IExtractor*. Esta implementación toma como fuente de datos los contadores procedentes de Microsoft Azure *Diagnostics* localizados en la tabla *WADPerformanceCounters* del servicio a monitorizar. Para su acceso es necesaria la obtención de la *ConnectionString* del servicio. Esta clase regula la obtención de datos, en caso de que se pidieran datos al extractor con mayor frecuencia de la que *Diagnostics* obtiene valores nuevos, este retornaría un valor vacío de manera que el resto del programa ignoraría esta medición hasta el siguiente ciclo de obtención de datos donde volvería comprobarse si existen valores nuevos y omitiendo cuando no existan procediendo de esta manera hasta obtener nuevas mediciones.

DataAccessLayer

Esta clase es la encargada de centralizar el acceso a la capa de datos de guardado de las métricas calculadas de manera que se desacople la fuente de datos utilizada para almacenar estos valores de la implementación y esta sea intercambiable dinámicamente.

IStorage

IStorage es la interfaz que define los posibles métodos de acceso a la base de datos de guardado de las métricas calculadas.

StorageMetricsController

Implementación de *IStorage* en la cual se implementa un adaptador para el almacenamiento de Azure en el cual se guardan las métricas calculadas.

QueueManager

QueueManager proporciona acceso a la API del sistema de colas de Microsoft Azure mediante el cual se transportan los modelos en tiempo de ejecución que determinarán la configuración de la monitorización.

ICounterConfigurator

Interfaz encargada de modelar los métodos disponibles para la realización de la configuración dinámica de contadores de rendimiento de la plataforma a monitorizar en el caso en que estos estén disponibles de manera programática (como lo es en el caso de Windows Azure). En esencia se podrán configurar los tipos de contadores de datos crudos que se quieren recuperar y el período de extracción en el cual se desea recuperarlos.

AzurePerformanceCounterConfigurator

Instancia de *ICounterConfigurator* encargada de la configuración de los *Performance Counters* disponibles en la plataforma Microsoft Azure. Esta se encargará de activar y desactivar la extracción de datos según las necesidades marcadas por los modelos en tiempo de ejecución recibidos así como determinar el período de extracción de estos datos.

5.4.3 Descripción del funcionamiento

En esta sección se detallará el funcionamiento de una ejecución del Middleware de Monitorización.

Primero, el monitor se encuentra desplegado y ejecutándose en una instancia *Worker Role* en la plataforma Microsoft Azure. Una vez lanzado a ejecución, se crea un objeto *Monitor* que se encargará en un primer momento de buscar un modelo en tiempo de ejecución con el cual iniciar el proceso de monitorización. Una vez generado este modelo por el Configurator de la Monitorización (descrito en el apartado 6.2 de este documento) será enviado por el sistema de colas de Microsoft Azure hasta el monitor que lo recuperará a través de la clase *QueueManager*.

Con el documento ya en el monitor, este será instanciado en un objeto *RuntimeModel* con el cual poder acceder a todos los datos de la configuración. Las operacionalizaciones de las métricas indicarán qué *Performance Counters* son necesarios para realizar las medidas. De manera que el monitor procederá a utilizar la instancia apropiada de la interfaz *ICounterConfigurator*, en este caso Microsoft Azure por lo que se empleará *AzurePerformanceCounterConfigurator* para realizar la activación en el servicio correspondiente.

Una vez terminada la configuración, el *Monitor* procederá a realizar los cálculos de las métricas presentes en el modelo en tiempo de ejecución. Para ello se ha empleado una estructura de la fórmula (véase el ejemplo escrito en XML de una fórmula en la Figura 17) compatible con la librería NCalc [8]. De manera que cualquier fórmula aritmética descrita de esta manera pueda ser calculada dinámicamente. Primero se procederá a dar valores a los parámetros de la fórmula,

para ello extrayéndolos de la instancia de *IExtractor* apropiada, en el caso de ser un contador de Microsoft Azure, usando *WADPerformanceCounterExtractor*, el cual buscará valores recientes (no empleados ya en otros cálculos) para representar ese valor. En caso de no existir todavía nuevos valores, la métrica en cuestión no será calculada en esta iteración y tendrá que esperar a la siguiente. Una vez calculados los valores, *NCalc* procederá a calcular el valor de la métrica la cuál será almacenada en la base de datos destinada a tal efecto por la instancia de *IStorage* seleccionada en la configuración, en este caso la tabla *Calculated Metrics* creada en el Almacenamiento de Azure (Azure Storage).

```
<name>Defective Operations Per Million (DPM)</name>
<operationalization xsi:type="IndirectMetric">
  <name>DPM</name>
  <function>
    <formula>([36]-[35])/[36]</formula>
    <operands>
      <DirectMetric>
        <name>\ASP.NET Applications(*)\Requests Total</name>
        <identifier>[36]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
      </DirectMetric>
      <DirectMetric>
        <name>\ASP.NET Applications(*)\Requests Succeeded</name>
        <identifier>[35]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
      </DirectMetric>
      <DirectMetric>
        <name>\ASP.NET Applications(*)\Requests Total</name>
        <identifier>[36]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
      </DirectMetric>
    </operands>
  </function>
</operationalization>
```

Figura 17 – Ejemplo de operacionalización representada en XML.

Este proceso se repetirá de manera cíclica hasta que el monitor reciba una actualización del modelo en tiempo de ejecución, momento en el cual reconfigurará el servicio y procederá a realizar de nuevo el bucle del proceso.

6. Caso de estudio

En este capítulo se presentará un escenario como ejemplo del funcionamiento de la infraestructura de monitorización. En primer lugar, se presentará el contexto de este caso, a continuación las tareas de configuración llevadas a cabo y para finalizar el funcionamiento del Middleware de Monitorización para la monitorización de ese servicio.

6.1 Presentación del caso

Una empresa de supermercados Market.SL ha decidido contratar un servicio de venta en la nube. En él los clientes pueden realizar compras de los productos del supermercado, incluyendo aquellos ítems que deseen en el carrito de la compra y realizando el pago de manera electrónica para más tarde recogerlos en el establecimiento a su elección.

Este servicio se encuentra desplegado en Microsoft Azure por un tercero en la forma de una aplicación web, cuyo servicio se denomina CoCoMe. Se ha decidido contratar este software como servicio debido a que este modelo ofrece ventajas como la elevada disponibilidad, facilidad de acceso para los clientes, externalización de costes de mantenimiento, posibilidad de proveer con más recursos en temporadas de alta actividad como la navidad, etc.

El supermercado considera críticas para el servicio la eficiencia y la precisión. Market.SL está interesado en que los tiempos de respuesta del servicio sean lo suficientemente bajos para que al cliente no le resulte engorroso realizar sus compras ya que el objetivo es que el proceso de la compra se haga más rápida y cómodamente de lo que sería al realizarlo en un establecimiento físico. A su vez, también es vital que durante el proceso de compra se generen los mínimos errores posibles para evitar frustraciones en el usuario y también errores que puedan suponer un coste a la empresa (pedidos enviados por duplicado, pedidos no finalizados, cobros incorrectos...). Por ello la empresa proveedora del servicio y Market.SL incluyen estas condiciones en el acuerdo de servicio.

Con mayor detalle en el acuerdo de nivel de servicio (SLA) se acuerdan los requisitos no funcionales siguientes:

1. La eficiencia (*Efficiency*) será medida a través de la latencia, para ello se usará la métrica del tiempo de ejecución de la petición y se ha establecido que será de máximo 130 milisegundos y será calculada con la siguiente fórmula:

$$\text{Execution Time} = \text{Request Execution Time}$$

2. La confiabilidad (*Reliability*) será medida a través de la operaciones defectuosas por millón. En este caso, el servicio tendrá un máximo de 10 operaciones defectuosas por millón (99.999% de confiabilidad de servicio). Este requisito será calculado utilizando la métrica correspondiente (*Defective Operations per Million-DPM*)

$$DPM = \frac{\text{Operations Attempted} - \text{OperationsSuccessful}}{\text{Operations Attempted}} * 10^6$$



Estos requisitos no funcionales serán monitorizados para velar por su cumplimiento. En el caso de que no se cumplan, el proveedor de la aplicación de compras deberá reembolsar a Market.SL por las pérdidas ocasionadas.

6.2 Configuración de la monitorización

Una vez acordados los requisitos no funcionales en el SLA y con el servicio ya en marcha, el encargado de la configuración de la monitorización, se dirigirá a la aplicación del Configurador de la Monitorización. En la pantalla de Inicio, escogerá comenzar la configuración.

6.2.1 Introducción de los datos de la plataforma

En la primera interfaz, *Platform Selection* mostrada en la Figura 18, se seleccionará en el punto 1.1 la plataforma en la cual se ha desplegado el servicio, en el este caso Microsoft Azure. Una vez seleccionada aparecen dos campos más con los datos propios de los servicios de Azure.

En el campo 1.2 se introduce el nombre del servicio, en nuestro caso el servicio es la aplicación web CoCoMe. En el caso del *Deployment ID* y el *Connection String* estos son parámetros propios del servicio que deben ser consultados en el panel de control del servicio CoCoMe, el primero es un identificador del servicio y el segundo una clave pública que concede acceso a los datos del servicio, necesarios para llevar a cabo la monitorización.

La configuración finalizada puede observarse en la Figura 18. Una vez terminado se selecciona Next para continuar con el siguiente paso.

1. Monitoring configurator - Platform Selection

1.1 Choose the hosting platform of the service to be monitored:

Windows Azure

1.2 Write the instance name of the service to be monitored:

CoCoMe

1.3 Enter the Azure Service credentials:

Deployment ID

fdc45005307044e0a00073e118ab71b0

Connection String

VERCz4l6tq/K1SZFPTOtr/KBHBeksoGMGw==

Back Next

Figura 18 – Caso de Estudio: Selección de Plataforma.

6.2.2 Introducir el modelo de requisitos

En este apartado deberá aportarse al configurador un modelo de requisitos que contenga los requisitos establecidos por el SLA. En este caso se ha optado por utilizar uno más completo pero que contiene esta información. Este archivo debe estar en formato XMI, como se ha indicado en el punto 5.3.3 de este documento. En la siguiente Figura se ilustra el resultado final del modelo subido:

2. Monitoring configurator - Monitoring Requirements Model Selection

Upload the Monitoring Requirements Model in xmi format:

Examinar... No se ha seleccionado ningún archivo.

Upload Model File uploaded!

This is the uploaded Monitoring Requirements Model:

NFR	Threshold	Metric Name	Metric Formula
GuaranteeOfReliability	>99.999	DPM	$((OperationsAttempted - OperationsSuccessful) / OperationsAttempted) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
GuaranteeofAvailability	>99.995	Service Robustness	(available time for invoking SaaS) / total time for operationg SaaS
Latency	<130	Latency	Request Execution Time
Stability	>99.999		
Service Accuracy	~100%	DPM	$((OperationsAttempted - OperationsSuccessful) / OperationsAttempted) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
Transactions per Hour	~10000	Transactions Per Hour	TransactionsPerHour

Back Next

Figura 19 – Caso de Estudio: El modelo de especificación de requisitos.

Una vez subido este modelo de forma correcta, aparece el botón *Next* que se seleccionará a continuación para llegar hasta el siguiente paso.

6.2.3 Asociación de métricas

A continuación se debe realizar la asociación de los requisitos no funcionales a monitorizar, recordar que en este caso se trata de la Latencia y la confiabilidad, con la métrica correspondiente del modelo SaaS.

Tomando primero la confiabilidad. Se selecciona de la tabla dicho RNF (en la Figura 20 puede observarse su localización junto al punto 3.1) y se procede a realizar la asociación, en este caso la característica a la que pertenece la confiabilidad es *reliability* (confiabilidad), el atributo *fault tolerance* (tolerancia a fallos) y la operacionalización es la correspondiente a DPM. En la Figura 20 puede observarse la asociación realizada por completo.

Middleware para la monitorización de la calidad de servicios cloud

3.1 Select the Non Functional Requirement to be monitored:

Non Functional Requirements

NFR Name	Threshold	Metric Name	Metric Formula
GuaranteeOfReliability	>99.999	DPM	$((OperationsAttempted - OperationsSuccessful) / OperationsAttempted) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
GuaranteeofAvailability	>99.995	Service Robustness	(available time for invoking SaaS) / total time for operationg SaaS
Latency	<130	Latency	Request Execution Time
Stability	>99.999		
Service Accuracy	=100%	DPM	$((OperationsAttempted - OperationsSuccessful) / OperationsAttempted) * 10^6$
		Service Accuracy	Number of Correct Responses / Total Number of Requests
Transactions per Hour	=10000	Transactions Per Hour	TransactionsPerHour

Selected NFR from the table:

Selected: [GuaranteeOfReliability](#)

3.2 Map the selected NFR to its appropriate metric from the SaaS Quality Model

To do so, navigate through its corresponding characteristics, subcharacteristics(if it applies) and attributes.

Characteristic: Reliability Subcharacteristic: Attribute: Fault Tolerance Metric: Defective Operations Per Million (DPM)

Select the operationalization which will calculate the metric:

Operationalization

- DPM
- DPM2
- DPM3

Platform Independent Operationalization
 $((Operations Attempted - Operations Successful) / Operations Attempted) * 10^6$

Add to Model @ Runtime

Figura 20 – Caso de Estudio: Clasificación de la confiabilidad.

A continuación se procede de la misma manera con el RNF de Latencia. En este caso la clasificación se realizará de la siguiente manera: la característica a la que pertenece es Eficiencia de Funcionamiento (*Performance Efficiency*), la subcaracterística comportamiento temporal (*Time Behaviour*), el atributo tiempo de respuesta (*Response Time*), la métrica *Latency* y la operacionalización *Latency*. Tal y como muestra la Figura 21 en la configuración finalizada.

Selected NFR from the table:

Selected: [Latency](#)

3.2 Map the selected NFR to its appropriate metric from the SaaS Quality Model

To do so, navigate through its corresponding characteristics, subcharacteristics(if it applies) and attributes.

Characteristic: Performance Efficiency Subcharacteristic: Time Behaviour Attribute: Response Time Metric: Latency

Select the operationalization which will calculate the metric:

Operationalization

- Latency

Platform Independent Operationalization
 Request Execution Time

Add to Model @ Runtime

Figura 21 – Caso de Estudio: Clasificación de la latencia.

Hasta el momento ya se tienen los RNFs clasificados con unos atributos determinados y unas métricas procedentes del modelo de calidad SaaS independientes de la plataforma determinadas. En el próximo paso se procederá a elaborar las fórmulas dependientes de la plataforma.

Una vez añadidas ambas al modelo en tiempo de ejecución, se pulsa Next para continuar con el proceso.

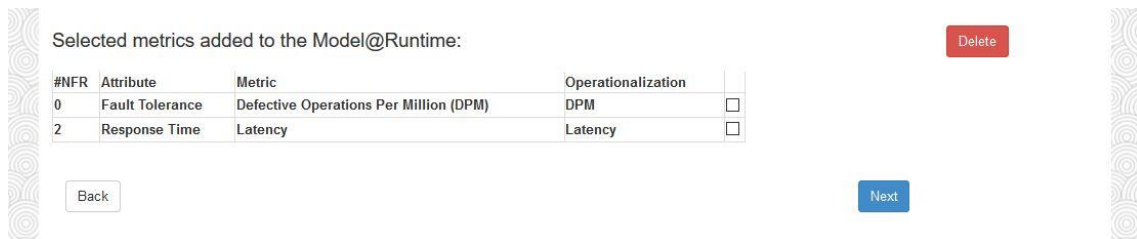


Figura 22 – Caso de estudio: Métricas añadidas al modelo en tiempo de ejecución.

6.2.4 Construcción de fórmulas

En este paso, las operacionalizaciones independientes de la plataforma deben transformarse en operacionalizaciones dependientes de ella. Para ello, con la guía de las operacionalizaciones SaaS, se deben componer las fórmulas con las que se trabajará para calcular las métricas de cada RNF.

En primer lugar se toma de la tabla la métrica de operaciones defectuosas y se realiza la fórmula, tomando los *Performance Counters* que se puedan asociar a cada uno de los elementos de la fórmula del espacio reservado a ellos en la aplicación. En este caso *Operations Attempted* pasa a ser el contador *Requests Total* mientras que *Operations Successful* pasa a ser *Requests Succeeded*. Para añadir el contador Requests Total se busca en la lista y como parámetros se selecciona como tipo de extracción Total (dado que este es un contador acumulativo, véase la sección 5.2 de este documento para más información) y el periodo de extracción 3 segundos. En el caso de *Requests Succeeded* también se toma este período y tipo de extracción, dado que ambos son contadores de funcionamiento similar. Puede observarse la construcción completa en la Figura 23.

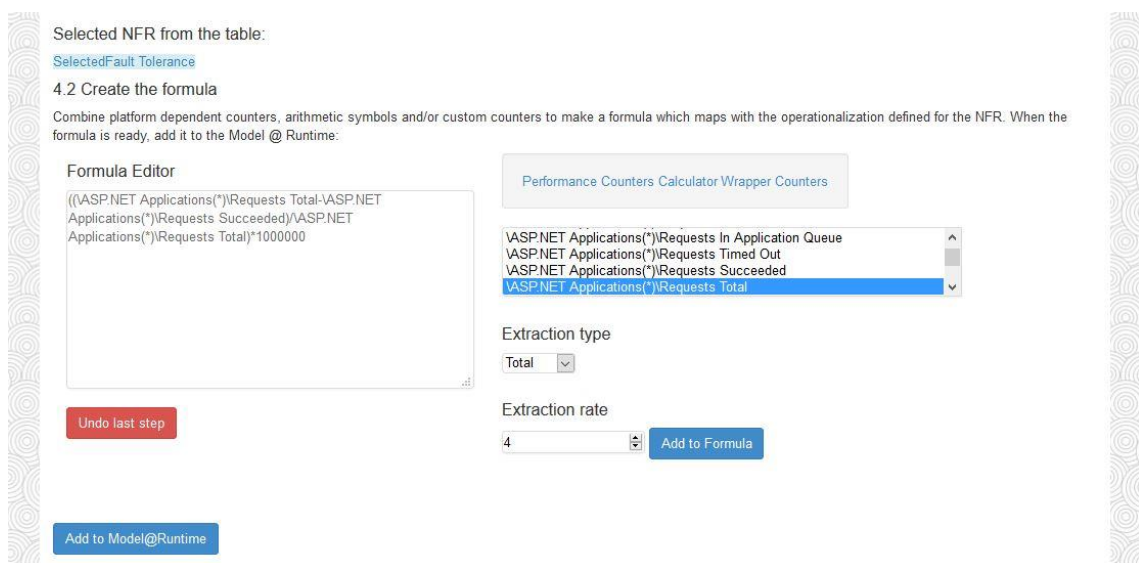


Figura 23 – Caso de Estudio: Construcción de la operacionalización de la confiabilidad.

Middleware para la monitorización de la calidad de servicios cloud

Una vez añadida esta fórmula, se añade al modelo en tiempo de ejecución y se procede a añadir la siguiente métrica, la latencia. En este caso *Request Execution Time* (miembro de la fórmula independiente de la plataforma) tiene correspondencia directa con los parámetros de la plataforma en el nombre. Se procede a añadir este *Performance Counter* tal y como se vio en el caso anterior, solamente cambiando los parámetros. Esta vez se selecciona como tipo de extracción *Average*, dado que este contador representa el valor únicamente de la última petición medida, por lo que no es necesario obtener el último valor sino un valor medio. El tiempo de extracción en este caso será de 2 segundos dado que se necesitarán más muestras para conseguir el mayor espectro de resultados posibles. En la Figura 24 se muestra el resultado de esta configuración.

Selected NFR from the table:
SelectedResponse Time

4.2 Create the formula

Combine platform dependent counters, arithmetic symbols and/or custom counters to make a formula which maps with the operationalization defined for the NFR. When the formula is ready, add it to the Model @ Runtime:

Formula Editor

ASP.NET Applications(*)Request Execution Time

Performance Counters Calculator Wrapper Counters

ASP.NET Applications(*)Forms Authentication Success
ASP.NET Applications(*)Forms Authentication Failure
ASP.NET Applications(*)Viewstate MAC Validation Failure
ASP.NET Applications(*)Request Execution Time

Extraction type
Average

Extraction rate
3

Undo last step

Add to Model@Runtime

Add to Formula

Figura 24 – Caso de Estudio: Construcción de la operacionalización de la latencia.

Una vez añadido esta última fórmula al modelo en tiempo de ejecución se procede a generarlo. Pudiendo descargar una copia en formato XML y mandando al Middleware de Monitorización la información de monitorización presente en el modelo, así configurando el servicio y comenzando el proceso de extracción de datos.

4.3 Generate Model@Runtime

Current Model@Runtime metrics to be monitored shown above. Click generate Model@Runtime to download the document in XML format:

NFR #	Attribute Name	Metric Name	Platform Dependent Operationalization	Platform Independent Operationalization	
0	GuaranteeOfReliability	Defective Operations Per Million (DPM)	((Operations Attempted - Operations Successful)/Operations Attempted)*10^6	((ASP.NET Applications(*)Requests Total-ASP.NET Applications(*)Requests Succeeded)/ASP.NET Applications(*)Requests Total)*1000000	<input type="checkbox"/>
1	Latency	Latency	Request Execution Time	ASP.NET Applications(*)Request Execution Time	<input type="checkbox"/>

Back

Generate Model@Runtime

Delete

El XML generado puede revisarse en el apartado 2 del Anexo al final de este documento.

Figura 25 – Caso de Estudio: Resultado de la configuración.

6.3 El Middleware de Monitorización

Una vez generado el modelo en tiempo de ejecución este será recibido por la instancia del Monitor que esté ejecutándose. Se analizará este modelo en XML y se convertirá la información recibida en instancias cuya información se utilizará para realizar la configuración. Se tomarán las métricas empleadas y usando las fórmulas se procederá a activar en el servicio la recogida de información de los Contadores de Rendimiento seleccionados, en este caso *Requests Succeeded*, *Requests Total*, *Request Execution Time* y *Requests Response Time*.

Una vez se comience a disponer de información del servicio se tomarán las métricas y se procederá a analizar las fórmulas y “llenar” los atributos correspondientes con los valores obtenidos de la extracción de datos. Con estos valores se calculará la fórmula cuyo resultado dará lugar a una medida de este RNF a monitorizar y que se guardará en una base de datos. Revisando las medidas de la Latencia y la Confiabilidad pueden observarse si se cumplen o no los requisitos observados por el SLA.

El Middleware continuará recogiendo datos y calculando las métricas indicadas por este modelo hasta que un nuevo modelo en tiempo de ejecución sea recibido y se actualicen los parámetros.



7. Conclusiones y trabajos futuros

En este capítulo se recogen las conclusiones generales del presente trabajo, a continuación se presentan los trabajos futuros propuestos para continuar con este trabajo de investigación y para concluir los resultados obtenidos en forma de publicaciones académicas.

7.1 Conclusiones

La computación en la nube es una tendencia en continuo ascenso en el mundo de la informática actual desde su llegada. Esta tecnología presenta una serie de ventajas a consumidores y proveedores que la hacen beneficiosa para ambos. Sin embargo una parte vital para la existencia de servicios en la nube, es que se pueda medir la calidad del servicio. Para ello deben existir herramientas y técnicas capaces de extraer la información de los servicios, localizados en una maraña de centros de datos, máquinas y máquinas virtuales de la misma manera que se puede controlar una solución tradicional, de forma que se pueda verificar el cumplimiento de los requisitos que un software en la nube debe presentar y en caso contrario utilizar esta información para mejorar dichos servicios.

El modelo SaaS presenta unas características de calidad diferentes al software tradicional, en algunos aspectos como la disponibilidad con umbrales de exigencia mucho más altos. Debido a las características propias del software en la nube resulta un reto mantener dichos niveles y sin herramientas que permitan controlar la calidad del software desplegado en la nube sería imposible detectar y solucionar los problemas que se presentan. De la misma manera, resulta beneficioso tanto para los consumidores como los proveedores de servicios en la nube que existan herramientas capaces de determinar que los acuerdos de nivel de servicio se cumplen. Por una parte el aumento de confianza en este tipo de soluciones permite a los consumidores apostar por ellas y a los proveedores dar garantías de calidad a estos satisfaciendo así las necesidades de ambas partes.

Con la herramienta presentada se verifica la viabilidad del proceso de monitorización presentado por Cedillo et al [9] demostrando la capacidad de dicho proceso para llevarse a cabo con éxito en un entorno de producción, enriqueciéndose del conocimiento a bajo nivel aportado por este trabajo y refinándose para crear una solución factible para responder a las necesidades de monitorización del presente.

Cómo se ha observado, la mayoría de herramientas actuales centran sus esfuerzos en obtener los datos de bajo nivel de las plataformas y servicios desplegados en ellas. Siendo este un esfuerzo muy necesario, este trabajo ha demostrado interesante y útil la posibilidad de interponer una capa más entre datos crudos y los intérpretes de los mismos para comprimir el ingente volumen de datos obtenidos interpretándose como información de calidad con mayor significado aumentando así la capacidad de interpretar esta información y tomar decisiones que mejoren la calidad del servicio de una forma más intuitiva. De esta manera, esta herramienta no considera competidoras al resto de herramientas comerciales si no aliadas para conseguir datos de calidad relevantes. Los mecanismos de recolección de información han sido definidos teniendo en cuenta distintos escenarios de captura de información, los mismos que proporcionan al middleware la capacidad de utilizar librerías y herramientas de monitorización propias de la plataforma, la posibilidad de definir fórmulas de calculo personalizadas para medir la calidad de los servicios, y la posibilidad de extender los servicios para que provean información de calidad.

7.2 Tecnologías

En esta sección se exponen algunos de los inconvenientes presentados durante la realización de este trabajo relacionados con las tecnologías utilizadas.

7.2.1. El Uso de Modelos en tiempo de ejecución

El uso de modelos en la ingeniería del software es una tecnología muy útil para agilizar el tiempo de desarrollo de aplicaciones. En este proyecto se han empleado modelos en tiempo de ejecución como el contenedor de la información de la monitorización. Aunque esta tecnología tenga ventajas como la estructuración de la información, la reutilización de los modelos, la capacidad de expresión de realidades complejas, el mantenimiento de la trazabilidad de elementos a lo largo de la aplicación y la flexibilidad que aportan a la aplicación para cambiar su comportamiento en tiempo de ejecución. En concreto, estos modelos permiten cambiar en tiempo de ejecución las fórmulas de cálculo (operacionalizaciones) de métricas específicas para satisfacer los requisitos no funcionales expresados en el SLA o para definir nuevos requisitos no funcionales en tiempo de ejecución.

Su implementación en este trabajo ha estado repleta de complicaciones, en primer lugar a la hora de incorporar estos modelos a la aplicación no se encontró una manera satisfactoria y más o menos automática de transformar estos modelos, creados en formato XMI, a un formato compatible con .NET, a pesar de que la base de XMI fuera XML.

A su vez, trabajar con modelos teóricos diseñados para representar una realidad (compleja) de forma precisa, en ocasiones hace complicado su uso por parte del desarrollador. Esto es debido a que este diseño no está pensado para que sea funcional a la hora de programar incurriendo en grandes dificultades a la hora de navegar estos modelos y a su vez haciendo necesaria la creación de vistas simplificadas del modelo para trabajar programáticamente con aquellas partes necesarias, de manera que existan accesos más sencillos a la información accedida.

Sin embargo, con la tecnología y herramientas apropiadas estos inconvenientes desaparecen dejando solamente las virtudes de estas herramientas.

7.2.2 XML y XMI

Estos dos formatos de documento han ocasionado el mayor número de dificultades a la hora de desarrollar las aplicaciones, siendo necesario invertir muchas horas de esfuerzo en el desarrollo de programas capaces de traducir la información presentes en los modelos de XMI, de forma prácticamente manual, en instancias de la aplicación. Los modelos expresados en XMI son complejos, poseyendo gran número de clases e instancias que hacían necesario un desarrollo personalizado de cada tipo de modelo expresado en este formato. A pesar de la existencia de herramientas que realizan este trabajo de forma automática partiendo de documentos XML, las diferencias de formato con respecto a XML ocasionaban que los métodos de lectura automática de este formato fallarían con XMI.

De igual manera, no se encontraron herramientas capaces de transformar instancias en .NET de nuevo en modelos de XMI por lo que se tuvo que recurrir al uso de XML para las comunicaciones internas entre el configurador y el monitor.

7.2.3 Microsoft Azure

Las tecnologías de Microsoft Azure, tanto la propia plataforma en la nube como las tecnologías empleadas en ellas (C#, Visual Studio y ASP.NET) eran completamente nuevas para mí. Esto ha significado que gran parte del tiempo invertido en este proyecto ha sido empleado en el aprendizaje de estas tecnologías.

Mención aparte merece la librería *Diagnostics* y el resto de herramientas que trabajan con ella. Siendo un componente relativamente nuevo, actualizado en diciembre de 2014 con las actuales funcionalidades, la documentación presente en las páginas de Microsoft Azure en más de una ocasión era completamente escueta si no inexistente, dificultando muchas veces el desarrollo de la aplicación.

A pesar de todo ello, la experiencia con Microsoft Azure, que por otra parte ofrece herramientas de mucha calidad para trabajar en la nube así como una forma muy sencilla e intuitiva de desplegar aplicaciones en la nube (a pesar de que la cantidad de posibilidades puede resultar abrumadora) han sido positivas para el desarrollo del trabajo.

7.3 Herramientas finalizadas

En esta sección se exponen las conclusiones y los problemas encontrados a la hora de elaborar las herramientas de este trabajo: el Configurador de la Monitorización y el Middleware de Monitorización.

7.3.1 El configurador de la monitorización

El configurador de la monitorización es una de las dos herramientas software implementadas para dar soporte al método de monitorización. Esta herramienta consistía en una aplicación web que permitiera la realización de un modelo en tiempo de ejecución.

Por una parte ello implicaba un conocimiento profundo de los modelos que intervenían en el proceso, el modelo de requisitos de monitorización, el modelo de calidad SaaS y el modelo de monitorización en tiempo de ejecución (todos ellos presentes en el Anexo). Estos modelos debían de ser comprendidos perfectamente para la elaboración de la aplicación, lo cual implicaba un estudio de la teoría referente a la calidad de software y la terminología de la medición. A su vez, al tratarse de modelos en fase de investigación se tuvieron que revisar y corregir en aquellas ocasiones en las que el modelo no bastase para representar la realidad de la configuración.

A su vez, la parte técnica del desarrollo de páginas web era prácticamente desconocida para mí por lo que fue necesaria una lección rápida de CSS empleando Bootstrap y el aprendizaje de ASP.NET y sus componentes.

Se optó por utilizar la versión simplificada de ASP.NET denominada WebForms debido a que en un comienzo no se esperaba realizar una aplicación muy compleja. Sin embargo en sucesivas iteraciones, creció la complejidad y se ha conseguido llegar al presente prototipo a pesar de las trabas técnicas que se encontraron. En lo referente al diseño, también se acusó la falta de referentes que pudieran utilizarse para el desarrollo de una aplicación de estas características.

Por ello el presente prototipo todavía presenta margen de mejora en lo que usabilidad y diseño se refieren.

7.3.2 El Middleware de Monitorización

Desde el inicio del proyecto las características de este componente no han estado definidas. Se fue descubriendo qué funcionalidades debería tener según la investigación iba avanzando y nuevas preguntas iban surgiendo lo cual ha implicado un refinamiento del prototipo con el tiempo.

Este proceso ha ocasionado mucho trabajo por duplicado, descartando alternativas que no funcionasen y proponiendo otras nuevas. De la misma manera, las nuevas funcionalidades propuestas iban creciendo hasta en ocasiones sobrepasar el alcance de este proyecto.

Por ello el producto final no contiene toda la funcionalidad que hubiera sido deseable incluir y que se ha propuesto en el siguiente apartado.

Sin embargo, su desarrollo desde cero me ha ayudado a poner en práctica los conocimientos adquiridos sobre patrones de diseño durante la titulación.

7.4 Trabajos futuros

Este proyecto de investigación ha abierto diferentes frentes para futuros trabajos:

- **Mejora del Middleware de Monitorización:** Se deben incluir los adaptadores necesarios para cumplir con el cuarto escenario de extracción de datos: Definir *wrappers* para comunicarse con otros servicios y así conseguir datos de monitorización de terceros que poder incorporar a la monitorización. De igual manera sería necesario mejorar el diseño del ciclo de vida del monitor de manera que aprovechara las posibilidades de la computación en paralelo y la gestión de múltiples instancias.
- **Creación de una aplicación que de soporte al proceso de análisis:** Una vez obtenidos los datos es necesario que una aplicación web sea capaz de utilizar y tratar dichos datos para la obtención de informes sintéticos que permitan al usuario la confirmación del cumplimiento del acuerdo de servicio así como interpretar el estado del mismo.
- **Autorregulación del tiempo de extracción de datos:** Sería conveniente el estudio de los datos extraídos para así definir de forma dinámica los tiempos del ciclo de extracción de datos, de manera que el sistema detecte que las anomalías del servicio. Por ejemplo, que este tenga una actividad reducida y por lo tanto ciertos contadores no sea necesario recuperarlos de forma tan constante dado que los datos aportados son innecesarios. Así mismo sería aconsejable estudiar los tiempos más eficaces para la extracción de datos en relación al valor que pueden aportar con respecto al esfuerzo de extraerlos.
- **Reconfiguración dinámica y automática del servicio monitorizado:** Utilizando los datos aportados por la monitorización podría ser interesante el desarrollo de un sistema experto que determinara los cambios de arquitectura que ayudarían a mejorar el rendimiento del sistema de forma automática.
- **Mantenimiento de los modelos en tiempo de ejecución:** Sería conveniente extender el actual configurador para que sea capaz de editar los modelos en tiempo de ejecución generados de manera que el usuario no tenga que realizar el proceso por completo cada vez que desee realizar un cambio.

También se pretende explorar más a fondo la interoperabilidad del middleware de monitorización propuesto con otras herramientas de monitorización de calidad de servicios por medio de conectores, APIs, etc. Esta interoperabilidad potenciaría nuestra propuesta de

forma significativa, ya que el cálculo de métricas específicas para ciertos atributos de calidad puede requerir de un gran esfuerzo de desarrollo y sin embargo estar disponibles en la nube. También se pretende evaluar la usabilidad/experiencia de usuario del uso de los mecanismos de monitorización propuestos en un caso de estudio más complejo

7.5 Publicaciones académicas y otros resultados

Durante el desarrollo de este trabajo de fin de grado se han realizado diferentes contribuciones en modo de publicaciones. A continuación, se presentan dichas publicaciones ordenadas por importancia y detallando la contribución realizada en este trabajo:

Conferencias internacionales relevantes (Conferencias de nivel A según CORE [33])

- Cedillo, Priscila; Jimenez-Gomez, Javier; Abrahao, Silvia; Insfran, Emilio. Towards a Monitoring Middleware for Cloud Services. Presentado en la 12th IEEE International Conference on Services Computing (IEEE SCC 2015), Nueva York, EE.UU, 27 Junio – 2 Julio, 2015.

En esta publicación se presentaba una versión preliminar del monitor utilizando para ello un caso de estudio en el cuál se comprobaba el proceso de monitorización mediante la medida de una aplicación web desplegada en la nube. Esta publicación fue sometida a un proceso de revisión externa por pares para su aceptación final.

Conferencias nacionales relevantes

- Cedillo, Priscila; Jimenez-Gomez, Javier; Abrahao, Silvia; Insfran, Emilio. Definición de Mecanismos Personalizados de Monitorización de Servicios Cloud:. Presentado en XI Jornadas de Ciencias e Ingeniería de Servicios (JCIS 2015), Santander, 15 al 17 de Septiembre de 2015.

En esta publicación se exponen los cuatro escenarios de extracción de datos detectados durante la investigación así como la implementación de un contador personalizado del tipo del tercer escenario de la extracción de datos. Presentando en él una versión preliminar de la implementación del monitor. Esta publicación fue sometida a un proceso de revisión externa por pares para su aceptación final.

Experimento sobre la viabilidad del proceso de monitorización

Se realizó un experimento controlado en junio de 2014 para comprobar la viabilidad del proceso de monitorización y middleware propuesto. En él participaron alumnos del curso de “Calidad del Software” del Grado en Ingeniería Informática de la Universidad Politécnica de Valencia que emplearon un prototipo para configurar una monitorización de servicios en la nube. Se asistió en el diseño de este prototipo y durante el transcurso del experimento para ayudar en las explicaciones del mismo.

8. Referencias

1. Roundup Of Cloud Computing Forecasts And Market Estimates, 2015. Forbes. [En línea] Enero de 2015. <http://www.forbes.com/sites/louiscolombus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/>.
2. Cisco. (2012). Cisco Global Cloud Index : Forecast and Methodology , 2011–2016. White Paper, 2011–2016. Retrieved from http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns1175/Cloud_Index_White_Paper.html#wp9000816
3. W. Tsai, X. Bai, and Y. Huang, "Software-as-a-service (SaaS): perspectives and challenges", *Sci. China Inf. Sci.*, vol. 57, no. 5, 2014, pp. 1–15, doi: 10.1007/s11432-013-5050-z
4. S. A. Baset, "Cloud SLAs : Present and Future", *ACM SIGOPS Oper. Syst. Rev.*, vol. 46, no. 2, 2012, pp. 57–66, doi: 10.1145/2331576.2331586.
5. Y. Jiang, C. S. Perng, T. Li, and R. Chang, "Self-adaptive cloud capacity planning", *IEEE 9th Int. Conf. Serv. Comput. SCC*, 2012, pp. 73–80, doi: 10.1109/SCC.2012.8.
6. C. Muller, M. Oriol, X. Franch, J. Marco, M. Resinas, A. Ruiz- Cortes, and M. Rodriguez, "Comprehensive Explanation of SLA Violations at Runtime", *Serv. Comput. IEEE Trans.*, vol. 7, no. 2, 2014, pp. 168–183, doi: 10.1109/TSC.2013.45.
7. G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A Self-adaptive hierarchical monitoring mechanism for Clouds", *J. Syst. Softw.*, vol. 85, 2012, pp. 1029–1041, doi: 10.1016/j.jss.2011.11.1043.
8. N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: requirements as runtime entities", in *32nd Int. Conf. on Software Engineering*, 2010, vol. 2, pp. 199–202, doi: 10.1145/1810295.1810329.
9. Cedillo, P., Gonzalez-Huerta, J., Abrahao, S., & Insfran, E. (2014). Towards Monitoring Cloud Services Using Models@ run. time. 9th Workshop on Models@run.time, 31–40. Retrieved from <ftp://ceur-ws.org/pub/publications/CEUR-WS/Vol-1270.zip>
10. Kaniz Fatema, Vincent C.Emeakaroha, Philip D.Healy, John P.Morrison, Theo Lynn, "A survey of Cloud monitoring tools: taxonomy, capabilities, and objectives" *J. Parallel Distrib. Comput.* 74 (2014) 2918–2933
11. Hassan, Q. F., & Computers, F. (2011). Demystifying Cloud Computing. *Cross-Talk The Journal of Defense Software Engineering*, 24(1), 16–21.
12. Amazon. Amazon Cloud Watch. 2014. <https://aws.amazon.com/es/cloudwatch/> (accessed Agosto 2015).
13. Google. What is Google Cloud Monitoring? 2015. <https://cloud.google.com/monitoring/docs> (accessed Agosto 2015).



14. Ward, J. S., & Barker, A. (2013). Varanus: In Situ Monitoring for Large Scale Cloud Systems. 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, 2, 341–344. <http://doi.org/10.1109/CloudCom.2013.164>
15. Meng, S., & Liu, L. (2013). Enhanced monitoring-as-a-service for effective cloud management. IEEE Transactions on Computers, 62(9), 1705–1720. <http://doi.org/10.1109/TC.2012.165>
16. Moldovan, D., Copil, G., Truong, H.-L., & Dustdar, S. (2013). MELA: Monitoring and Analyzing Elasticity of Cloud Services. 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, 80–87. <http://doi.org/10.1109/CloudCom.2013.18>
17. Rackspace. Overview. 2015. <http://www.rackspace.com/cloud/monitoring/> (accessed Agosto 2015).
18. V. Emeakaroha, T. Ferreto, M. Netto, I. Brandic, C. De Rose, CASViD: application level monitoring for SLA violation detection in clouds, in: Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual, 2012, pp. 499–508. <http://dx.doi.org/10.1109/COMPSAC.2012.68>.
19. Google. What is the Monitor Agent. 2015. <https://cloud.google.com/monitoring/agent/> (accessed Agosto 2015).
20. Ruan, L., Peng, J., Xiao, L., & Wang, X. (2013). CloudDVMM: Distributed virtual machine monitor for cloud computing. *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCOM 2013*, 1853–1858. <http://doi.org/10.1109/GreenCom-iThings-CPSCOM.2013.344>
21. Zabbix. Zabbix Main Page. 2015. <http://www.zabbix.com/> (último acceso: Agosto de 2015).
22. Ganglia. What is Ganglia? 2015. <http://ganglia.info/> (accessed Agosto 2015).
23. Sandoval, Y., Gallizo, G., & Curiel, M. (2012). Evaluation of monitoring tools for cloud computing environments. *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*, 1–10. <http://doi.org/10.1109/CLEI.2012.6427251>
24. Fatema, K., Emeakaroha, V. C., Healy, P. D., Morrison, J. P., & Lynn, T. (2014). A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, 74(10), 2918–2933. <http://doi.org/10.1016/j.jpdc.2014.06.007>
25. Mell, P., & Grance, T. (2011). The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, 145, 7. <http://doi.org/10.1136/emj.2010.096966>
26. Badger, L., Grance, T., Patt Corner, R., & Voas, J. (2011). Cloud Computing Synopsis and Recommendations. *Recommendations of the National Institute of Standards and Technology, C O M P*, 84. <http://doi.org/2012>
27. Amazon. Amazon EC2 Service Level Agreement . 2013. <https://aws.amazon.com/es/ec2/sla/> (accessed Agosto 2015).

28. Google. Google Compute Engine Service Level Agreement. 2015. <https://cloud.google.com/compute/sla?hl=en> (accessed Agosto 2015).
29. Cedillo, P., Gonzalez-Huerta, J., Abrahao, S., & Insfran, E. . A Monitoring Infrastructure for the Quality Assessment of Cloud Services, 24th International Conference on Information Systems Development (ISD 2015), Model-Driven Development and Concepts Track, August 25 - 27, 2015, Harbin, China.
30. Cedillo, P., Jimenez-Gomez, J., Abrahao, S., & Insfran, E. (2015). Definición de Mecanismos Personalizados de Monitorización de Servicios Cloud, XI Jornadas de Ciencia e Ingeniería de Servicios (JCIS 2015), Santander, Septiembre 2015
31. Bencomo, N., Blair, G., Götz, S., Morin, B., Rumpe, B.: Report on the 7th Int. Workshop on Models@Runtime. SIGSOFT Softw. Eng. Notes. 38, 27–30. Innsbruck, Austria (2013).
32. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer (Long. Beach. Calif). 42, 22–27 (2009)
33. CORE. Conference Rankings. 2014. <http://www.core.edu.au/index.php/conference-rankings> (accessed Agosto 2015).
34. CodePlex. NCalc- Mathematical Expressions Evaluator for .Net. 2011. <https://ncalc.codeplex.com/> (accessed Agosto 2015).
35. Nimsoft. *Nimsoft*. 2015. <https://support.nimsoft.com/> (accessed Agosto 2015).
36. Monitis. *Network & IT Systems Monitoring- Monitis*. 2015. <http://www.monitis.com/> (accessed Agosto 2015).
37. Pedrovitti. *PCMONS - Github*. 2014. <https://github.com/pedrovitti/pcmons> (accessed Agosto 2015).
38. .Boundary. *Boundary - Server and applications monitoring*. 2015. <http://www.boundary.com/> (accessed Agosto 2015).
39. Microsoft. *Collecting data with Microsoft Azure Diagnostics*. 2014. <https://msdn.microsoft.com/en-us/library/azure/gg433048.aspx> (accessed Agosto 2015).
40. ISO/IEC 25010:2011-Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models, 2011.



1.3 Modelo de Monitorización en Tiempo de Ejecución

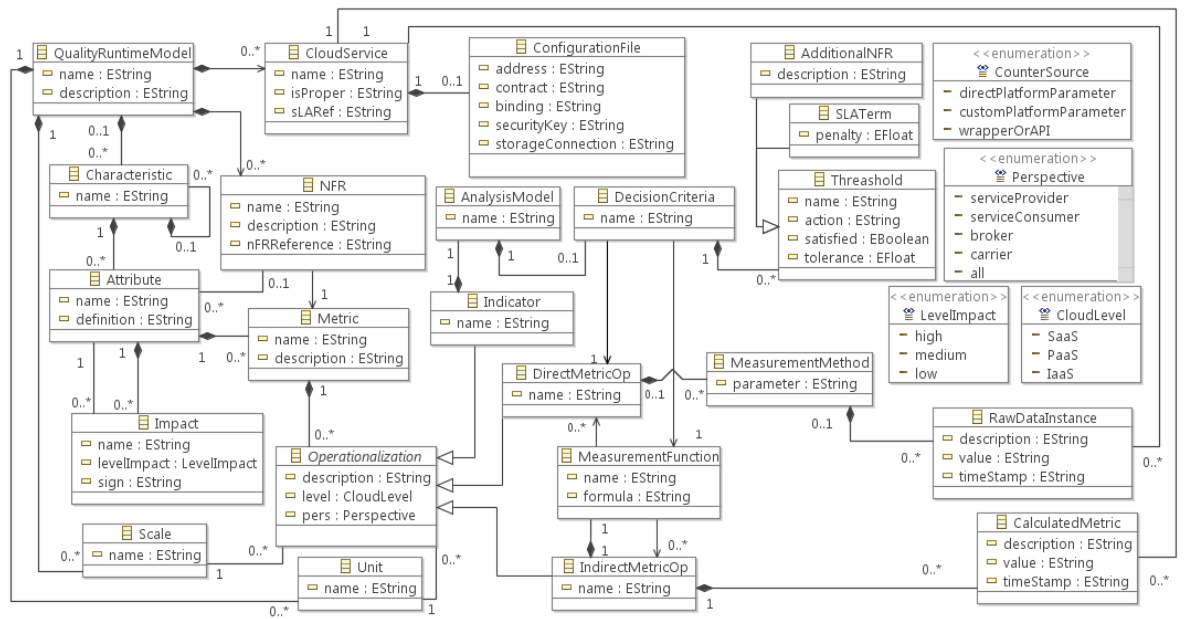


Figura 28 – Anexo: Modelo de Monitorización en Tiempo de Ejecución.

2. Modelo en tiempo de ejecución en XML

```

<?xml version="1.0"?>
<RuntimeModel xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <CloudService>
    <Name>CoCoMe</Name>
    <connectionString>Eby8vdM02xNOCqFlqUwJPLlmeTlCDXJ1OUzFT50uSRZ6IFsuFq2U
VErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==</connectionString>
    <deploymentID>fdc45005307044e0a00073e118ab71b0</deploymentID>
  </CloudService>
  <Characteristics>
    <Characteristic>
      <name>Reliability</name>
      <attributes>
        <Attribute>
          <name>Fault Tolerance</name>
        </Attribute>
      </attributes>
      <metrics>
        <Metric>
          <name>Defective Operations Per Million (DPM)</name>
          <operationalization xsi:type="IndirectMetric">
            <name>DPM</name>
            <function>
              <formula>(( [36] - [35] ) / [36] ) * 1000000</formula>
              <operands>
                <DirectMetric>
                  <name>\ASP.NET Applications (*) \Requests
Total</name>
                  <identifier>[36]</identifier>
                  <extractionRate>4</extractionRate>
                </DirectMetric>
              </operands>
            </function>
          </operationalization>
        </Metric>
      </metrics>
    </Characteristic>
  </Characteristics>
</RuntimeModel>

```



```

        <extractionType>0</extractionType>
    </DirectMetric>
</DirectMetric>
        <name>\ASP.NET
Applications (*)\Requests Succeeded</name>
        <identifier>[35]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
    </DirectMetric>
</DirectMetric>
        <name>\ASP.NET Applications (*)\Requests
Total</name>
        <identifier>[36]</identifier>
        <extractionRate>4</extractionRate>
        <extractionType>0</extractionType>
    </DirectMetric>
</operands>
</function>
</operationalization>
</Metric>
</metrics>
<nfr>
    <name>GuaranteeOfReliability</name>
    <nFRReference>0</nFRReference>
    <attributes />
</nfr>
</Attribute>
</attributes>
<subCharacteristics />
</Characteristic>
<Characteristic>
    <name>Performance Efficiency</name>
    <attributes />
    <subCharacteristics>
        <Characteristic>
            <name>Time Behaviour</name>
            <attributes>
                <Attribute>
                    <name>Response Time</name>
                    <metrics>
                        <Metric>
                            <name>Latency</name>
                            <operationalization xsi:type="DirectMetric">
                                <name>\ASP.NET Applications (*)\Request Execution
Time</name>
                                <identifier>[68]</identifier>
                                <extractionRate>3</extractionRate>
                                <extractionType>0</extractionType>
                            </operationalization>
                        </Metric>
                    </metrics>
                </Attribute>
            </attributes>
            <subCharacteristics />
        </Characteristic>
    </subCharacteristics>

```

```
</Characteristic>
</Characteristics>
<NFRs>
  <NFR>
    <name>GuaranteeOfReliability</name>
    <nFRReference>0</nFRReference>
    <attributes />
  </NFR>
  <NFR>
    <name>Latency</name>
    <nFRReference>2</nFRReference>
    <attributes />
  </NFR>
</NFRs>
</RuntimeModel>
```