La escalabilidad en la gestión de aplicaciones IoT basadas en servicios

UNIVERSIDAD
DE MÁLAGA

Tesis Doctoral
(Por compendio)

Nicolás Pozas García
Programa de Doctorado en Tecnologías Informáticas
ETS Ingeniería Informática
Universidad de Málaga

Dirigida por
Francisco Javier Durán Muñoz

Junio 2024

AUTOR: Nicolás Pozas García

https://orcid.org/0000-0002-2761-5836

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

Agradecimientos

Antes de ponernos técnicos, y comenzar a explicar conceptos y términos informáticos poco entendibles para los no-informáticos, me gustaría agradecer a todas las personas que han contribuido ya sea directa o indirectamente, en el desarrollo de esta tesis.

Aunque la tesis comenzó formalmente en 2021, hace unos tres años, su desarrollo comenzó mucho antes. Concretamente durante la E.S.O., donde mi madre insistía en que su hijo estudiase y se dejase de tonterías diciendo que no quería ni terminar la E.S.O., y quería ponerse a trabajar ya. Gracias, mamá, por haber seguido insistiendo, al final hemos superado el objetivo y un poco más allá.

Por supuesto, gracias al resto de mi familia por su apoyo constante (y financiar gran parte de este viaje sin objeciones). A mi padre, quien, aunque no entendía del todo lo que estaba haciendo con el ordenador siempre me ha apoyado y ha querido que siga estudiando; a mi hermano, posible ingeniero informático pero que tiene que terminar antes bachiller, y a mi hermana, quien ha actuado como asesora en más de una ocasión.

Un agradecimiento especial a la persona que más de cerca ha vivido todo este proceso, a quien un día le decía que quería hacer una cosa, y al día siguiente otra. Cuando le preguntan que a qué se dedica su pareja, no sabe muy bien que decir más allá de "es ingeniero informático", y que hago muchos dibujitos en papel con números, grafos y esas cosas. Pero que siempre ha impedido que abandone, y también ha insistido en terminar esta tesis.

A mis amigos, porque más de una chapa sobre inteligencia artificial, sobre algoritmos y grafos se han comido. Más para aclarar mis propias ideas que para comprender realmente cómo funciona un perceptrón, y por qué el *accuracy* no es lo único importante. Algunos muy aburridos, pero otros no hacían nada más que preguntar y encima gratis, p*** Juan.

A las personas que me acogieron durante mi estancia de investigación en Colombia, así como a mis compañeros del ITIS, porque a pesar de ser de los pocos que no es teleco, y encima de otro grupo de investigación, no me han mirado del todo mal. Gracias a Javi, Pablo, Rafa, Bea, Delia y a mi excompañera Katia, que ha participado activamente en parte de esta tesis y a quien debo una barbacoa.

A mis compañeros del grupo de investigación Carlos, Javier Troya, Javier Cámara, Paula, Loli, Ernesto, Rafa, porque alguna que otra presentación me han corregido. Y, sobre todo, un agradecimiento especial a la persona de quien más depende esta tesis (o al menos la segunda), mi director Paco. Solo puedo ofrecerle mi más sincero agradecimiento por hacer tan "sencilla" la realización de la tesis, por darme la oportunidad de ir a Colombia, por darme la libertad de enfocar los problemas a mi manera, por discutir libremente, aunque sepa que él tiene razón, sobre la mejor solución posible. Por buscar siempre la calidad antes que la cantidad, y las publicaciones que avalan esta tesis lo demuestra. Realmente ha sido un privilegio contar con un director y compañero con tanta experiencia, gracias.

Por último, pero no menos importante... ¡Gracias, Jorge!, que no se me ha olvidado.

Índice

1. Introducción

Hoy en día, la proliferación de dispositivos conectados a través del Internet de las Cosas (IoT) ha dado lugar a un ecosistema digital en constante expansión. En este contexto, las aplicaciones diseñadas para facilitar la interconexión y la gestión de una amplia variedad de servicios desempeñan un papel crucial en la transformación de nuestra vida cotidiana. Estas aplicaciones no solo orquestan los datos generados por dispositivos IoT, sino que también ofrecen un conjunto amplio y variado de servicios, desde la monitorización de la salud en centros hospitalarios hasta la automatización del hogar.

No obstante, a medida que estas aplicaciones evolucionan para abordar las crecientes demandas de la sociedad, surge un desafío esencial: la escalabilidad. Tecnologías como la nube, el internet de las cosas (IoT) o el continuo cloud-fog-edge han introducido nuevos desafíos en el ámbito:

1. En primer lugar, se están considerando aplicaciones de cientos, incluso miles de servicios que refleja la evolución exponencial del ecosistema digital. La complejidad de esta gestión manual se ha vuelto poco práctica, y no solo obstaculiza la eficiencia operativa, sino que también subraya la urgencia de desarrollar soluciones automatizadas y escalables para garantizar la viabilidad continua de estas aplicaciones. Esto plantea desafíos que van más allá de la complejidad arquitectónica, extendiéndose hacia la gestión práctica, la optimización de la calidad del servicio y el despliegue eficiente teniendo en cuenta a usuarios dispersos globalmente.

2. En segundo lugar, la optimización de los atributos de calidad de servicio (QoS), se convierte en una tarea compleja cuando servicios funcionalmente equivalentes, pero con distintos atributos de QoS, pueden intercambiarse en repositorios de servicios para dar respuesta a diferentes eventos. La motivación detrás de este problema radica en la necesidad de tomar decisiones informadas y estratégicas sobre la implementación de servicios maximizando la eficacia y minimizando los costes asociados, donde la oferta disponible dificulta la decisión de dónde implementar cada servicio.

3. El tercer problema abordado destaca la importancia de las distancias y la calidad de los canales de comunicación en aplicaciones distribuidas globalmente. Las distancias entre los componentes de una aplicación, y entre los servicios de front-end y los usuarios de las aplicaciones, así como la calidad de los canales de comunicación utilizados, son clave para proporcionar los tiempos de respuesta apropiados a los usuarios, lo que en un contexto en el que tenemos proveedores de servicios ubicados en todo el mundo puede suponer una diferencia significativa. La motivación aquí reside en ofrecer tiempos de respuesta adecuados a los usuarios, reconociendo que la geografía de los proveedores de servicios puede impactar significativamente en la experiencia de usuario.

4. El cuarto aspecto, la predicción del comportamiento de los servicios, se convierte en un elemento crucial para anticiparse a las necesidades, permitiendo evitar pérdidas de tiempo si es posible prever los eventos próximos, optimizando así la eficiencia y la capacidad de respuesta del sistema al anticiparse a las necesidades futuras de los servicios y del sistema en su conjunto.

5. Finalmente, el quinto problema, la verificación sistemática de los servicios, emerge como una piedra angular, para garantizar el correcto funcionamiento y la integridad de cada componente, así como la interacción entre los mismos, y contribuyendo así a la confiabilidad y la seguridad general del sistema.

La incorporación de una alta cantidad de servicios dentro de una aplicación no solo intensifica la complejidad de su arquitectura, sino que también presenta obstáculos significativos en términos de mantenimiento, actualización y verificación de todos sus componentes. La escalabilidad se convierte así en un aspecto crítico para garantizar la eficiencia y la sostenibilidad a largo plazo de estas aplicaciones.

En esta tesis exploramos la importancia de la escalabilidad en aplicaciones orientadas a IoT con una amplia variedad de servicios, haciendo énfasis en cada uno de los puntos comentados anteriormente. Analizamos cómo la capacidad de escalar de manera efectiva se convierte en un factor determinante para la adaptabilidad y la viabilidad continua de estas aplicaciones en entornos dinámicos. Además, examinamos casos de uso específicos en el día a día, donde la escalabilidad se vuelve crucial, ya sea para implementar modificaciones, añadir nuevos servicios o verificar el correcto funcionamiento de todos los elementos involucrados.

La tesis se sumerge en este desafiante escenario, explorando cada faceta de este para comprender la importancia de estrategias eficientes de escalabilidad. A través de casos de uso concretos, la investigación se propone ofrecer soluciones y perspectivas valiosas para el desarrollo futuro de sistemas tecnológicos más resilientes y adaptables.

En las siguientes secciones abordaremos cada uno de los puntos anteriores de forma individual, pero atendiendo al contexto del problema global, dando sentido a las preguntas que han guiado el trabajo que se desarrolla en esta tesis:

- *¿Es posible conseguir una composición de servicios con miles de nodos en un tiempo razonable atendiendo a las necesidades y restricciones del usuario y de la propia aplicación?*

- *¿Cómo afectan los canales de comunicación a la calidad de las soluciones? ¿Se puede establecer algún método que nos ofrezca alternativas entre los diferentes proveedores que tenemos en cuenta para poder reemplazarlos rápidamente en caso de que sea necesario?*

- *¿Se podría predecir el comportamiento de estos nodos para obtener una estrategia que nos permita adelantarnos a los acontecimientos y así aumentar la eficiencia de nuestras aplicaciones según sus necesidades?*

- *¿Se podría verificar que el comportamiento de estas aplicaciones formadas por miles de nodos es el que cabría esperar, y además de una forma escalable?*

En concreto, en la sección 1.1 presentaremos varias técnicas, tecnologías y lenguajes de modelado e implementación que serán usados en el resto de la tesis. Más tarde, en la sección 1.2, se explorarán los diversos aspectos clave relacionados con la composición de servicios. Continuando con la sección 1.3 donde se analizará a fondo la necesidad de una gestión de recursos eficiente, mientras que la sección 1.4 se adentrará en la verificación y el análisis estadístico de aplicaciones. Además, en la sección 1.5, destacaremos los trabajos que respaldan esta tesis. Finalmente, en la sección 1.6, detallaremos cómo se han aplicado las diversas tecnologías en los experimentos realizados.

1.1. Preliminares

En esta sección introducimos brevemente algunas nociones básicas sobre algunas de las tecnologías que usamos en esta tesis para comprender los elementos claves que sustentan la investigación y el desarrollo presentado en este trabajo. Se abordarán diversos temas que constituyen los pilares de la investigación, incluyendo los algoritmos genéticos, las redes neuronales recurrentes, donde nos centraremos sobre todo en las redes LSTM (*Long Short-Term Memory*), en los procesos de negocio modelados con BPMN (*Bussiness Process Model and Notation*), así como herramientas y lenguajes de programación específicos como P o Maude; con P modelaremos aplicaciones como conjuntos de *statecharts* y Maude lo usaremos para modelar procesos de negocio. Aunque existen un amplio abanico de técnicas y herramientas para verificar sistemas especificados en P y en Maude, en concreto usaremos *statistical model checking* para verificar propiedades cuantitativas de programas P y usaremos Maude para modelar y realizar análisis cuantitativos basados en simulaciones de asignaciones de recursos a procesos de negocio. Estos elementos forman la base teórica y práctica que sustentan el desarrollo y la contribución original de esta tesis.

1.1.1. Algoritmos genéticos

Los algoritmos genéticos (*Genetic Algorithms*, GA) [66] son una técnica de optimización inspirada en la evolución natural y en la genética. Funcionan mediante la creación y manipulación de una población de soluciones candidatas para encontrar la mejor solución posible a un problema dado.

El proceso comienza con la generación aleatoria de una población inicial de individuos, donde cada individuo representa una solución potencial al problema. Estos individuos están codificados en una representación que refleja las características relevantes del problema, lo que permite su evaluación y comparación.

Durante cada iteración del algoritmo, llamada *generación*, se evalúa cada individuo de la población utilizando una función objetivo predefinida, a esta función se la conoce como función de *fitness*. Esta función asigna un valor numérico que indica qué tan buena es cada solución en términos de satisfacer los objetivos del problema. En nuestro caso, esta función está acotada en el rango $[0, 1]$, donde 0 representa la peor solución y 1 representa la mejor solución.

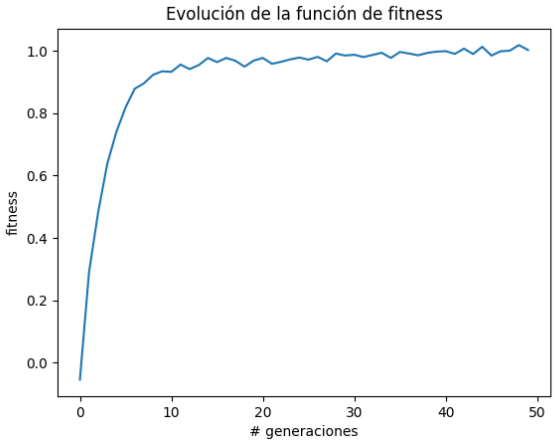


Figura 1.1: Evolución de la función de *fitness*

A continuación se seleccionan individuos para reproducirse, dando preferencia a aquellos con mayor valor de función de *fitness*. Este proceso de selección puede ser determinista, como la selección *por torneo*, donde se seleccionan los individuos con los valores de función de *fitness* más altos, o estocástico, como la selección *por ruleta*, donde se le da una probabilidad en función del valor de la función de *fitness* del individuo (mayor valor, mayor probabilidad de ser seleccionado).

Una vez seleccionados los padres, se aplican operadores de mutación y entrecruzamiento para generar su descendencia. La mutación consiste en realizar pequeños cambios aleatorios en la información genética de un individuo, lo que introduce diversidad en la población y ayuda a evitar la convergencia prematura hacia soluciones subóptimas (máximos locales) fomentando la exploración. Por otro lado, el entrecruzamiento implica intercambiar información genética entre dos padres seleccionados, produciendo descendientes que combinan características de ambos padres. Esto permite explorar nuevas áreas del espacio de búsqueda y combinar características beneficiosas de diferentes soluciones, fomentando la explotación de las soluciones prometedoras.

Finalmente, los nuevos individuos, tanto los descendientes como los padres seleccionados, forman la población para la siguiente generación. Este proceso de selección, reproducción y aplicación de operadores genéticos se repite durante un número predeterminado de generaciones o hasta que se cumpla un criterio de terminación, como alcanzar una solución satisfactoria o que en un número determinados de generaciones no se haya encontrado una mejor solución. Este último método es el utilizado a lo largo de la tesis. En la figura 1.1 se puede observar cómo la evolución del *fitness* durante la ejecución del algoritmo genético suele seguir una función asintótica. Se observa que en las primeras generaciones se logra un buen valor de función de *fitness* rápidamente, pero luego el proceso tarda considerablemente más en alcanzar el valor óptimo o pseudo-óptimo. Esto suele permitir obtener una solución “aceptable” en relativamente poco tiempo.

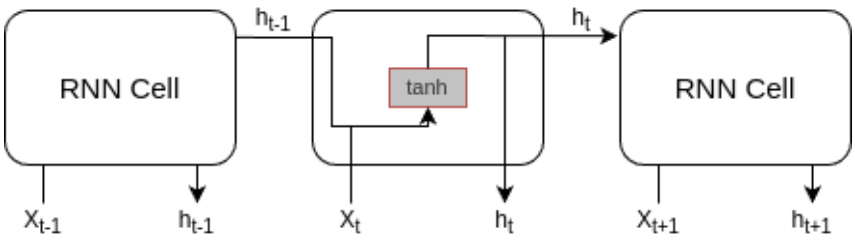


Figura 1.2: Neuronas recurrentes

El solucionador de algoritmos genéticos usado a lo largo de esta tesis ha sido implementado usando la biblioteca Jenetics[1]. Los hiper-parámetros usados en nuestros experimentos pueden ser encontrados en la sección 1.6.1. La biblioteca Jenetics ofrece dos condiciones de parada: un tiempo de ejecución dado, o un criterio de convergencia que hace que la evolución termine cuando el valor objetivo sea considerado convergente, es decir, no se consiga un valor mejor durante las últimas generaciones ejecutadas.

Cada uno de los experimentos realizados con GA usados en esta tesis y en los trabajos que la avalan se ha realizado 10 veces, y los valores mostrados en las gráficas de los distintos experimentos son la media de estas ejecuciones para suavizarlas.

1.1.2. Redes recurrentes

Las redes neuronales recurrentes (*Recurrent Neural Network*, RNN) [88] son un tipo de arquitectura de red neuronal diseñada para modelar secuencias de datos, donde la salida en cada paso de tiempo depende de la entrada actual y de la información procesada en pasos de tiempo anteriores. Este tipo de redes permite capturar dependencias temporales en los datos, lo que las hace especialmente adecuadas para tareas como el procesamiento del lenguaje natural, la predicción de series temporales y la traducción automática.

Las RNN funcionan propagando la información a través de conexiones recurrentes que forman bucles en la arquitectura de la red. En cada paso de tiempo, la red toma una entrada y calcula una salida junto con un estado oculto, que encapsula la información relevante de pasos de tiempo anteriores. Este estado oculto se actualiza en cada paso utilizando la entrada actual y el estado oculto anterior, lo que permite a la red mantener una representación dinámica de la historia de la secuencia.

En la figura 1.2 se pueden observar tres pasos de tiempo, donde en el paso de tiempo actual (t), se recibe la información actual (x_t) y la historia de la información pasada (h_{t-1}) dando lugar a una nueva historia actualizada (h_t).

Sin embargo, las RNN tradicionales tienen dificultades para manejar dependencias a largo plazo debido al problema de desvanecimiento o explosión del gradiente [78]. Este problema ocurre cuando los gradientes que se propagan a través de los pasos de tiempo se vuelven demasiado pequeños o demasiado grandes, lo que dificulta el aprendizaje de dependencias temporales a largo plazo.

[1]La biblioteca Jenetics está disponible en `https://jenetics.io/`.

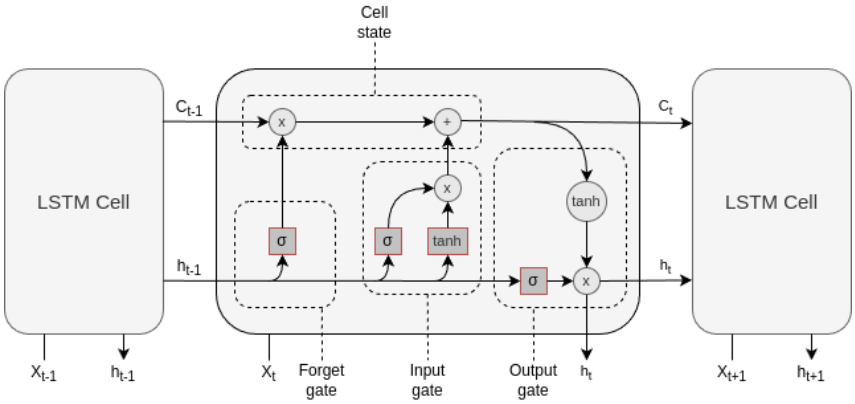


Figura 1.3: Neuronas recurrentes LSTM

Para abordar este problema, surgieron las redes neuronales de memoria a largo plazo (LSTM) [106], una variante de las RNN diseñada para capturar dependencias a largo plazo en los datos de la secuencia, como se propone en [79]. Las LSTM incorporan unidades de memoria adicionales llamadas celdas de memoria, que actúan como un mecanismo de almacenamiento de la información a largo plazo. Estas celdas de memoria están controladas por compuertas, como las compuertas del olvido (*Forget gate*), de entrada (*Input gate*) y de salida (*Output gate*), que regulan el flujo de información dentro de la red.

En cada paso de tiempo, las LSTM deciden qué información recordar, olvidar y actualizar utilizando las compuertas y la entrada actual. Esto permite que las LSTM aprendan dependencias a largo plazo al tiempo que controlan el flujo de información, evitando el problema de desvanecimiento o explosión del gradiente. En la figura 1.3 se pueden encontrar estas compuertas.

En resumen, las redes neuronales recurrentes, especialmente las LSTM, son arquitecturas de redes neuronales diseñadas para modelar secuencias de datos, capturando dependencias temporales a través de conexiones recurrentes y utilizando mecanismos como las compuertas para aprender y recordar información a largo plazo. Esto las hace especialmente útiles en tareas que implican procesamiento de secuencias, donde la información contextual es crucial.

1.1.3. Procesos de negocio

BPMN 2.0 [12] es un estándar ISO/IEC [56] que se utiliza ampliamente para modelar procesos de negocio. BPMN es una notación gráfica para modelar procesos de negocios como colecciones de tareas relacionadas que producen servicios o productos específicos. En BPMN, los procesos se modelan utilizando representaciones gráficas para tareas y compuertas, que están conectadas a través de flujos y eventos. Aunque la especificación de BPMN incluye varios diagramas, en este trabajo nos centramos en los elementos de BPMN relacionados con el modelado del flujo de control y los aspectos conductuales. Por lo tanto, se consideran dos tipos principales de diagramas de BPMN, los diagramas de actividad y los diagramas de colaboración. Para estos diagramas, consideramos las tareas, los eventos y las compuertas, que

son los elementos más comunes. Además de estos elementos, la especificación de BPMN se extiende para proporcionar información sobre recursos, así como sobre duraciones y retrasos de las tareas y los flujos. Esta información adicional permitirá un análisis en tiempo real de los procesos especificados.

El proceso de la figura 1.4 se utiliza para introducir e ilustrar el uso de los elementos de BPMN admitidos. Aunque se trata de un proceso de negocio muy simple, el proceso ayudará a guiar la discusión.

El flujo de trabajo de la figura 1.4 comienza con una compuerta exclusiva de apertura que presenta tres ramas. Las compuertas, representadas con diamantes, $<>$, se utilizan para controlar la divergencia y convergencia de los flujos de ejecución. En ese trabajo, se admiten compuertas *exclusivas* y *paralelas*, representadas, respectivamente, con un símbolo $+$ o $\times$, es decir, <+>y <×>. Las bifurcaciones, compuertas con un flujo de entrada y varios de salida, las llamaremos *splits*, mientras que a las que tienen varios de entrada y uno de salida, las llamaremos compuertas *merge*. Cada rama de un *split* exclusivo está asociada a un valor que indica la probabilidad de que el sistema continúe a través de esa rama específica, excluyendo las demás. En la figura, la rama superior tiene una probabilidad 0.3, la rama intermedia del 0.2 y la rama inferior del 0.5. La ramas superior e inferior contienen una sola tarea cada una, representadas con rectángulos, y designadas como S_1 y S_4 respectivamente, donde una tarea representa una actividad atómica que tiene exactamente un flujo de entrada y un flujo de salida. Mientras tanto, en la rama central se encuentra un *merge* exclusivo que une dos ramas: una de ellas representa un bucle y otra la conexión con la compuerta anterior. Posteriormente, se presentan dos tareas en secuencia, S_2 y S_3, que finalizan en la compuerta de inicio del bucle. Estas dos tareas están conectadas mediante un flujo de secuencia que se describe como dos nodos ejecutados uno después del otro, es decir, impone un orden de ejecución entre estos dos nodos. Prosiguiendo con el proceso, nos encontramos con la compuerta exclusiva de cierre, seguida por la tarea S_5 y una compuerta paralela de apertura. En estas compuertas paralelas, todas las ramas se ejecutan concurrentemente, siendo necesario que la ejecución de cada una de las ramas converja en una compuerta paralela de cierre para continuar con el proceso global. Entre estas compuertas paralelas de apertura y cierre, se encuentran las tareas S_6 y S_7. El proceso descrito finaliza después de la compuerta paralela de cierre. La lectura intuitiva del diagrama consistiría en que el proceso empieza con la ejecución de una de las tres ramas, cada una con la probabilidad indicada, ejecutándose S_1, S_4 o repetidamente S_2 seguido de S_3. Al terminar la ejecución de la rama seleccionada, se ejecutaría S_5 y acabaría tras ejecutar concurrentemente S_6 y S_7.

En estos diagramas, los recursos son definidos explícitamente en el nivel de tarea. Por lo tanto, una tarea que requiere recursos para su ejecución debe incluir, como parte de su especificación, los recursos requeridos. Para representarlos, se asocian símbolos a cada tipo de recursos, y estos símbolos se incluyen dentro de las tareas correspondientes. Nótese que los recursos pueden referirse a humanos (por ejemplo, empleado, cajero, ejecutivo), así como a recursos no humanos (por ejemplo, robot, máquina virtual, dron, herramienta). En general, además de múltiples recursos, se puede especificar un número de ellos. Por ejemplo, cierta cantidad de harina para

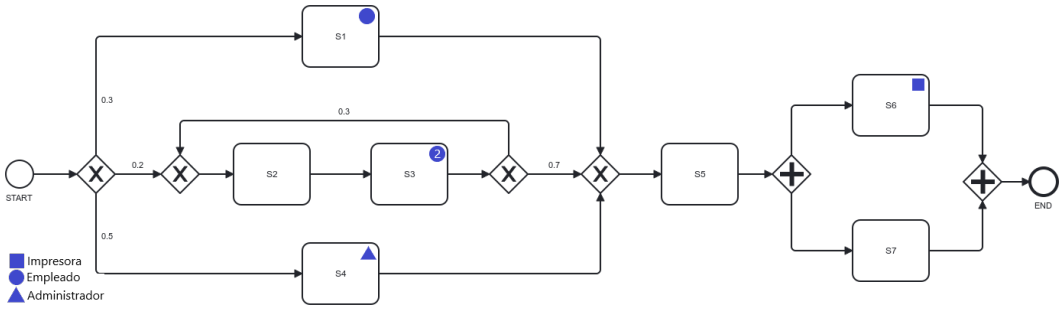


Figura 1.4: Ejemplo de proceso de negocio BPMN

preparar una receta o cierta cantidad de dinero para comprar algún producto podrían ser especificados. Para evitar tratar con múltiples unidades de medida, los recursos se cuentan como instancias o réplicas, y si se requiere más de una instancia de un cierto tipo de recursos, se representan como un número de iconos en la tarea, por ejemplo, volviendo a la figura 1.4, podemos ver que la tarea S_1 necesita de un empleado para completarse, mientras que la tarea S_3 necesita dos empleados para ser completada, y las S_4 y S_6, respectivamente, necesitan de un administrador y de una impresora.

La especificación de recursos puede, sin embargo, incluir información, por ejemplo, sobre los rangos a considerar, por ejemplo, puede darse el caso en el que no se puedan asignar más de cinco empleados en las instalaciones de la empresa, o sobre el tiempo de asignación, pueden ser necesarios algunos días para asignar un nuevo automóvil, varios minutos para asignar una máquina virtual, o varias semanas para contratar a un nuevo ingeniero.

El proceso evoluciona ejecutando sus tareas sucesivamente. Sin embargo, la ejecución de una tarea requiere las cantidades especificadas de recursos, lo que puede llevar a una competencia por dichos recursos. Además, es habitual que un mismo proceso se ejecute múltiples veces concurrentemente, por ejemplo, en una tienda online atendiendo pedidos. Es decir, múltiples instancias del proceso también pueden ejecutarse concurrentemente y múltiples tareas en la misma ejecución pueden requerir los mismos recursos finitos.

1.1.4. El lenguaje P

P [25] es un lenguaje de programación específico del dominio, asíncrono y basado en eventos. Un programa P define una colección de máquinas de estado que interactúan entre sí mediante eventos. En general, proporciona primitivas que permiten capturar la comunicación entre los diferentes componentes descritos. P se ha utilizado, por ejemplo, en Amazon (AWS) para el análisis de sistemas distribuidos complejos y protocolos, o en Microsoft para implementar y validar la pila de controladores de dispositivos USB [31].

La semántica de P está basada en el modelo de actores [2, 53]. El modelo de actores es un paradigma de programación concurrente donde las unidades de ejecución, llamadas *actores*, interactúan entre sí mediante el intercambio de mensajes. Cada actor tiene su propio estado interno y puede realizar acciones como respuesta

a mensajes que recibe. En P, los actores se definen como entidades independientes que puede comunicarse entre sí enviando mensajes asíncronos. Este enfoque permite construir sistemas concurrentes de manera modular y escalable, donde los actores pueden ejecutarse en paralelo sin necesidad de una sincronización explícita.

Esto es posible porque el comportamiento no determinista habitual, debido a la computación concurrente y todas las demás fuentes de no determinismo en P (como aquellas vinculadas a construcciones específicas del lenguaje), se vuelven puramente probabilístico. Esta transición se efectúa garantizando que el planificador de eventos ofrezca todas las posibles opciones siguiendo una distribución de probabilidad, generalmente uniforme, para seleccionar probabilísticamente la opción a elegir [6].

Una ventaja de P radica en sus capacidades de prueba y validación. Estas capacidades pueden extenderse para admitir varios motores de análisis con el fin de verificar que el sistema distribuido programado en el lenguaje cumple con las especificaciones de corrección deseadas. Como se indica en su página oficial [31], P ayuda a los desarrolladores de tres formas fundamentalmente:

- Escribiendo especificaciones formales en P que obligan a los desarrolladores a pensar en el diseño de sus sistema de forma rigurosa, y a su vez les ayuda a comprender mejor el sistema. Gran parte de los errores pueden eliminarse en el propio proceso de redacción de las especificaciones.
- La comprobación de los modelos ayuda a encontrar errores puntuales en el diseño del sistema que no se detectaron en las pruebas de estrés e integración.
- Tras la sobrecarga inicial que supone la creación de modelos formales, las futuras actualizaciones y ampliaciones de funciones pueden llevarse a cabo con mayor rapidez, ya que estos cambios se validan rigurosamente antes de su implementación.

P nos permite hacer algunas verificaciones de seguridad, como comprobar que un servicio bancario no pueda tener un flujo de caja negativo, y verificaciones de viveza, que son aquellas en las que se asegura que los componentes del sistema se están mandando eventos entre sí, es decir, verificando que el sistema se encuentra activo. La realización de este tipo de verificaciones es importante, dado que existe la posibilidad de que todas las medidas de seguridad estén siendo cumplidas únicamente porque el sistema ha quedado atrapado en un bucle que impide llevar a cabo otras operaciones.

1.1.5. Maude

Maude [22] es un lenguaje de especificación y programación diseñado para modelar y analizar sistemas concurrentes, así como sistemas basados en reglas. Es conocido por su capacidad para representar de manera precisa y formalizar sistema complejos, permitiendo la especificación de reglas de inferencia, operaciones y comportamientos de sistemas computaciones de una manera modular y matemáticamente rigurosa.

Este lenguaje se basa en la lógica de reescritura [71-73], con la lógica ecuacional de pertenencia como sublógica, y utiliza un enfoque de reescritura para definir la semántica de sistemas. Ofrece características como tipos de datos algebraicos, módulos, operaciones de reescritura y módulos de búsqueda, lo que facilita la especificación y análisis de sistemas formales.

Maude se ha utilizado en una amplia gama de aplicaciones, incluyendo la verificación de programas, la especificación de protocolos de comunicación, la simulación de sistemas concurrentes, la ingeniería del software y la inteligencia artificial. Su enfoque formal y su capacidad para representar sistemas complejos lo hacen especialmente útil en áreas donde se requiere precisión y confiabilidad en el modelo y análisis de sistemas computacionales.

Los procesos BPMN usados en esta tesis se modelan en Maude con dos objetivos diferentes. En el primer caso, la ejecución de un proceso está guiada por un registro de eventos. en el segundo caso, el proceso es *autónomo*, en el sentido de que su ejecución procede aleatoriamente, teniendo en cuenta expresiones estocásticas que modelan las diferentes decisiones que ocurren: decisiones sobre qué ramas tomar, duraciones de actividades y retrasos en flujos. Esta información sobre el comportamiento del sistema puede ser proporcionada por expertos reales o aprendida de los mismos registros de eventos.

El proceso autónomo es responsable de la simulación del sistema, la gestión y el análisis de los recursos. También se encarga de comunicarse con el proceso Python que realiza las predicciones que veremos más adelante en la sección 1.3 de este mismo capítulo. El proceso Maude enviará el registro de eventos de las actividades realizadas y, periódicamente, solicitará predicciones. En este momento, el proceso BPMN representado en Maude crea una réplica de sí mismo, que será guiada por la secuencia de eventos enviada por el predictor y evaluará cómo de buena ha sido la predicción.

1.2. Composición de servicios

En esta sección, exploraremos diversos aspectos clave relacionados con la composición de servicios. Discutiremos los patrones arquitectónicos utilizados para esta tarea, así como la función de *fitness* que guía la selección y composición de servicios. También abordaremos los diferentes algoritmos de composición empleados para resolver el problema de composición.

Además, examinaremos los canales de comunicación utilizados para la interacción entre los servicios compuestos, así como la reemplazabilidad de los servicios mediante métodos de utilidad. Esta sección proporcionará al lector un análisis integral sobre los enfoques y técnicas involucradas en la composición de servicios.

1.2.1. Patrones arquitectónicos y función de *fitness*

En las distintas soluciones al problema de la composición que podemos encontrar en la literatura se pueden encontrar diferentes enfoques sobre cómo trabajar con estas composiciones. Uno de ellos es ver cada servicio como un servicio abstracto

(por ejemplo, un servicio de búsqueda) para el cual se debe seleccionar un servicio concreto (por ejemplo, Algolia, CloudSearch, etc.) con métricas específicas para los atributos de QoS, como pueden ser el coste, la potencia de computación, la fiabilidad, la disponibilidad, etc. Un problema alternativo es ver los servicios como servicios específicos que deben desplegarse en proveedores específicos (servicios PaaS de diferentes proveedores, diferentes máquinas virtuales de diferentes proveedores, una máquina Linux local, un dispositivo Arduino, etc.), que nuevamente ofrecen métricas específicas de QoS. Aunque ambos plantean un problema muy similar, nos centraremos en el segundo caso a lo largo de esta tesis. El problema específico que abordamos está ejemplificado por la aplicación representada en la figura 1.5, la cual coincide con el diagrama BPMN definido previamente en la figura 1.4 de la sección 1.1.3. En este nuevo diagrama, cada tarea es implementada por un servicio correspondiente y cada servicio tiene una oferta de proveedores asociados. En ella, podemos ver la arquitectura de una aplicación en la que S_1 (servicio 1) puede desplegarse tanto en P_1 (proveedor 1) como en P_2, S_4 puede desplegarse en P_3 y P_4, S_2 en P_7 y P_5, S_3 en P_6 y P_3, S_5 solo en P_1, S_6 en P_3 y P_5, y S_7 en P_1, P_8 y P_3. Como se puede observar, varios servicios pueden estar desplegados en el mismo proveedor (siempre que el proveedor lo permita). A su vez, vemos cómo estos servicios están conectados entre sí mediante diferentes canales de comunicación que forman diferentes patrones arquitectónicos. Si miramos de nuevo la figura 1.5, vemos que S_2 está conectado de manera *secuencial* con S_3, es decir, cuando S_2 termine su ejecución comenzará la ejecución de S_3. A su vez, S_3 está conectado mediante un patrón *iterativo* de nuevo a S_2, es decir, cuando el S_3 termine su ejecución habrá una probabilidad p (0.3), de que la ejecución vuelva al comienzo del patrón (S_2) o por el contrario, con una probabilidad $1-p$ (0.7) de que la aplicación continúe con el cierre el patrón *condicional*. Vemos cómo S_1, el patrón iterativo (S_2 y S_3) y S_4 están conectados entre sí mediante un patrón condicional, donde cada rama tiene asociada una probabilidad p_i de ser ejecutada por la aplicación (0.3, 0.2 y 0.5), donde $\sum_i p_i = 1$. Por último, tenemos S_6 y S_7 conectados entre sí mediante un patrón *paralelo*, que es aquel en el que cada una de sus ramas se ejecuta simultáneamente y tendremos que esperar que terminen todas y cada una de las ramas para poder continuar con la ejecución de la aplicación. Las probabilidades mostradas en la figura son estimaciones obtenidas en base a la experiencia e información previa que tenemos de la aplicación en cuestión.

La complejidad de la composición de servicios centrada en la calidad de servicio (QoS) radica en los muchos factores que deben tenerse en cuenta simultáneamente. En primer lugar, los servicios combinan sus entradas y salidas según lo dicta la arquitectura de la aplicación, lo cual tiene un impacto directo en sus atributos de calidad. Esto se vuelve más complicado cuando, además de patrones secuenciales, la arquitectura incluye patrones condicionales, iterativos o paralelos. El segundo factor es el problema de optimización: la combinación de resultados debe lograr la mejor calidad de servicio global. Cuanto mayor sea el espacio de búsqueda, más costosa será la resolución del problema de optimización, y cómo crece el tiempo de cálculo se vuelve crucial para la escalabilidad. Por ejemplo, una aplicación con 100 componentes y 4 proveedores alternativos para cada componente tiene $4^{100} = 1.606938\text{e}{+60}$

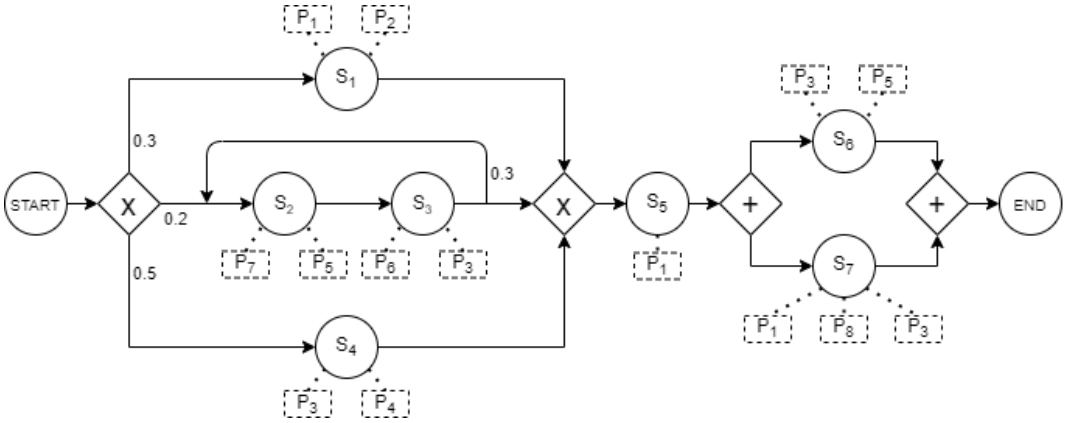


Figura 1.5: Ejemplo de aplicación típica

posibles combinaciones, y aunque este número puede reducirse aplicando restricciones estructurales (por ejemplo, componentes que deben implementarse en el mismo proveedor o que no pueden alojarse en algunos proveedores), y restricciones en los valores de los atributos QoS impuestas por el usuario, la cantidad de alternativas a analizar es demasiado alta, cuando solo unas pocas cumplirán con los requisitos del usuario.

En este trabajo, además permitimos a los usuarios que indiquen restricciones sobre los atributos QoS, que pueden ser restricciones *duras* (*hard constraints*) y restricciones *blandas* (*soft contraints*). Cuando una restricción dura es incumplida la solución es descartada directamente. Sin embargo, cuando una restricción blanda es incumplida, la solución no se descarta, sino que será penalizada con un peso (W_c) asignado por el usuario, con el que se indica la importancia de la penalización. Este peso estará en el rango $[0,1]$, donde 0 no penalizaría en absoluto el incumplimiento de las restricciones, y 1 indicaría que la solución se descartaría en caso de no cumplirse alguna de las restricciones, como si de una restricción dura se tratase.

Si volvemos a la aplicación descrita en la figura 1.5, que contiene 7 servicios y 2 proveedores para cada servicio, con la excepción de un servicio que cuenta con un solo proveedor y un servicio que cuenta con 3 proveedores, tenemos un total de $2^5 \times 1 \times 3 = 96$ posibles combinaciones: todas los servicios tienen asignado su primer proveedor $[0,0,0,0,0,0,0]$ (de su lista particular), todas las aplicaciones tienen asignado su primer proveedor excepto la última $[0,0,0,0,0,0,1]$, etc. Si incrementamos el número de posibles proveedores en solo una unidad el espacio de búsqueda pasaría a incrementarse hasta las $3^5 \times 2 \times 4 = 1944$ posibles combinaciones.

Para afrontar esta explosión combinatoria, existe un cierto consenso sobre el uso de algoritmos genéticos (GA, véase sección 1.1.1) como la mejor opción para resolver el problema de la composición. En general, las soluciones basadas en GA son capaces de encontrar una solución aceptable bastante rápido, aunque puede llevar bastante tiempo encontrar una solución pseudo-óptima para espacios de búsqueda enormes, como los que manejamos en este caso.

Adelantándonos un poco a los resultados presentados en el capítulo 4, la figura 1.6 muestra resultados de algunos experimentos que ilustran cómo se comportan

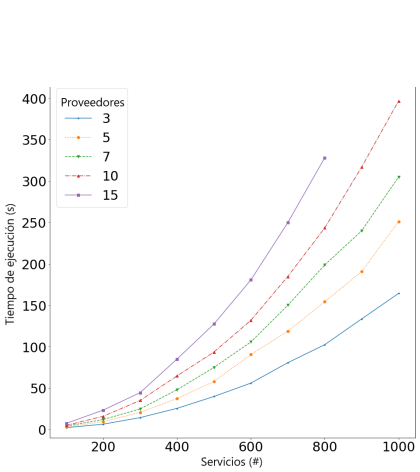
las soluciones basadas en GA a medida que crece el tamaño de la composición. En la figura 1.6a se muestra el tiempo promedio de ejecución para encontrar una implementación adecuada para aplicaciones con hasta 1000 servicios y diferentes números de proveedores por servicio sin un límite de tiempo hasta encontrar una solución que no se ha conseguido mejorar durante las últimas generaciones.

Podemos observar que el tiempo de ejecución aumenta exponencialmente a medida que aumenta el número de servicios o de proveedores. Por supuesto, al establecer un límite de tiempo, detendremos el algoritmo antes de alcanzar un máximo local. Sin embargo, hay un equilibrio entre el tiempo y la calidad de la solución encontrada, ya que los GA tienden a alcanzar un buen resultado rápidamente, pero tardan bastante más en mejorarlo. Cada una de las soluciones obtenidas por el GA viene acompañada por un valor que nos da información sobre cómo de buena ha sido dicha solución. Este valor es otorgado por una función objetivo (véase [81] y la sección 3.1). En el caso de los GA dicha función es llamada función de *fitness*. El valor de esta función está acotado en el rango $[0,1]$, donde 0 simboliza el peor valor teóricamente posible, y 1 simboliza el mejor valor teóricamente posible. Aquí decimos "teóricamente" ya que ambos valores son prácticamente imposibles de alcanzar por cualquier algoritmo.

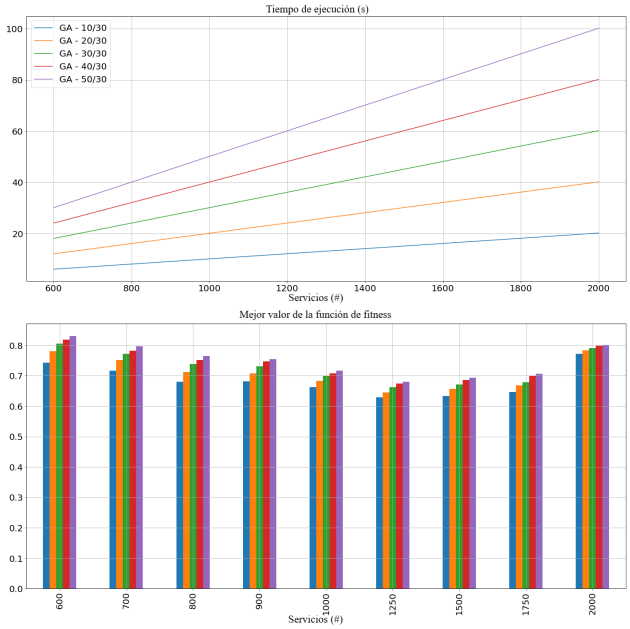
Obsérvese que no tendría sentido tener un límite de tiempo fijo, ya que cuanto mayor sea el espacio de búsqueda o en general más complejo el problema, peor sería la calidad de la solución. Para superar este problema, hemos considerado un límite de tiempo que dependa del número de servicios. Para la figura 1.6b, hemos creado una función $cx+y$ para darle un tiempo límite de ejecución a los diferentes algoritmos, donde c representa la pendiente, x el número de servicios e y el desplazamiento. Así pues, GA - 20/30 representa la función con pendiente 20ms y desplazamiento 30ms, cuyo límite de tiempo irá variando en función del número de servicios que estemos considerando. Los gráficos en esta figura ilustran cómo este límite de tiempo está relacionado con la calidad de la solución. En la parte superior de la figura 1.6b podemos ver diferentes funciones que definen los límites de tiempo utilizados y las soluciones respectivas en la parte inferior. Obsérvese que se ha utilizado el mismo código de colores en ambas gráficas de la figura 1.6b, es decir, la línea de un color en la gráfica superior se corresponde con las barras del mismo color para cada uno de los números de servicios de la gráfica en la parte inferior. Aunque la función de *fitness* está explicada en detalle en [81] y la sección 3.1, podemos observar en estos gráficos que cuanto menor sea el límite de tiempo, mayor será la pérdida en la precisión de la solución. De hecho, establecer un límite de tiempo demasiado bajo puede llevar a soluciones inaceptables.

1.2.2. Canales de comunicación y reemplazabilidad

Aunque el problema de composición se puede ver desde diferentes perspectivas, con diferentes nomenclaturas y en diferentes dominios, para nosotros el problema de composición de servicios es el proceso de asignar recursos a servicios de entre un conjunto de servicios disponibles. De todas las asignaciones posibles, nos interesa encontrar la mejor solución posible, lo que significa que el QoS general es óptimo, o



(a) Tiempos de ejecución variando el número de servicios para diferentes números de proveedores para cada servicio



(b) Tiempo de ejecución y mejor valor de la función de *fitness* por servicios

Figura 1.6: Escalabilidad de las soluciones basadas en algoritmos genéticos

pseudo-óptimo, en el menor tiempo posible y satisfaciendo todas las restricciones de los usuarios (tanto blandas como duras). Dado el escenario mencionado anteriormente, proporcionamos soluciones para el problema de composición teniendo en cuenta su escalabilidad, las ubicaciones de los servicios y las restricciones de los usuarios.

En la computación orientada a servicios es práctica común organizar los servicios existentes en flujos de trabajo [105]. La arquitectura de la aplicación basada en servicios que se va a componer se proporcionará como entrada, de modo que se consideren no solo las descripciones y restricciones de los servicios, sino también las dependencias funcionales explícitas entre los servicios. El objetivo principal de la gestión de dichos procesos en este área es optimizar las procesos para alcanzar los objetivos de negocio de manera óptima. Suponemos un formalismo de descripción de flujo de trabajo como WS-BPEL [7] (Web Services - Business Process Execution Language) o BPMN [1] (véase sección 1.1.3). Ambos son formatos muy comunes en la descripción de servicios y procesos de negocios. Nuestras técnicas son aplicables a otras notaciones para especificar flujos de trabajo, como Crusoe et al. [24] o Garijo et al. [50], de las que podemos extraer dependencias funcionales.

En nuestro enfoque suponemos que ha tenido lugar un emparejamiento funcional previo. Es decir, para cada servicio, habrá una oferta de proveedores de servicio candidatos, cada uno con una estimación de sus atributos de QoS, especificados en su acuerdo de nivel de servicio (*Service Level Agreement*, SLA), que se tendrá en cuenta para calcular los valores de la función de *fitness* de las posibles soluciones. Para poder considerar la calidad de las comunicaciones suponemos que también están disponibles la ubicación y la capacidad del canal de comunicación (*throughput*)

de los proveedores. Ambos atributos serán parte fundamental de nuestro trabajo, para minimizar los tiempos de latencia entre las comunicaciones de servicios y tener en cuenta posibles cuellos de botella.

La composición de servicios centrada en QoS es un problema NP-difícil bien conocido [95]. Por lo tanto, aunque el problema se puede resolver utilizando métodos exactos, como la programación entera [114], hemos visto anteriormente que existe un cierto consenso en el uso de algoritmos genéticos (GA) como una buena opción para el problema de composición gracias a su flexibilidad y fácil adaptación a casi cualquier problema de optimización (véanse, por ejemplo, [69], [87] y [57]). En general, los procedimientos basados en GA proporcionan una solución aceptable, principalmente porque en ellos se puede encontrar un equilibrio entre el tiempo y la precisión. En otras palabras, incluso limitando su tiempo de ejecución, aún pueden ser capaces de encontrar soluciones "aceptables" rápidamente, aunque después consuma más recursos mejorarla (véase la discusión anterior sobre la figura 1.6b en la sección 1.2).

También podemos encontrar en la literatura soluciones basadas en el uso de heurísticas (véanse, por ejemplo, [65] y [113]), la mayoría basadas en la *utilidad* de cada proveedor candidato para algún objetivo global. En general, podemos ver la noción de utilidad como una estimación asignada a cada proveedor para un servicio en particular, indicando la probabilidad de que dicho proveedor pueda contribuir a satisfacer las necesidades dadas para ese servicio específico y en qué medida [67]. En otras palabras, las medidas de utilidad intentan responder a la pregunta de cómo y cuánto pueden contribuir cada uno de los proveedores candidatos de un servicio, por ejemplo, a un valor global objetivo o unos requisitos dados. Por ejemplo, nuestro objetivo puede ser obtener un menor coste económico y una mayor confiabilidad. El uso de este tipo de heurísticas, con la posterior aplicación de un algoritmo genético para garantizar la satisfacción de las restricciones del usuario, podría dar resultados muy satisfactorios en un tiempo muy razonable (véanse, por ejemplo, [67] o [65]). De hecho, el método puede obtener no solo soluciones para un objetivo específico, sino también hacerlo en menos tiempo, ya que los proveedores de servicio pueden agruparse para reducir significativamente el espacio de búsqueda.

Para hacernos una idea de la complejidad del problema, comparemos experimentalmente varias de estas soluciones alternativas, por ahora sin restricciones. Consideremos una solución pura de GA (GA), una solución de GA-utility (U) y una solución que decida sobre el mejor candidato considerando individualmente, o localmente, cada servicio (Express). Consideremos una aplicación con 100 servicios, con 10 proveedores candidatos para cada servicio. Este problema de composición tiene un espacio de búsqueda de 10^{100} combinaciones posibles. La solución de GA comenzaría a mutar y recombinar individuos de su población inicial a través de este espacio de búsqueda hasta que se encuentre un resultado adecuado. Dependiendo de sus meta-parámetros, exploraría una parte más grande o más pequeña de este espacio de búsqueda, con una mayor o menor probabilidad de acercarse a un óptimo global. Un método basado en utilidad con cinco grados de utilidad tendría un espacio de búsqueda de tamaño 5^{100}, considerablemente más pequeño, aunque requeriría un análisis previo adicional. Finalmente, un método en el que el proveedor para cada

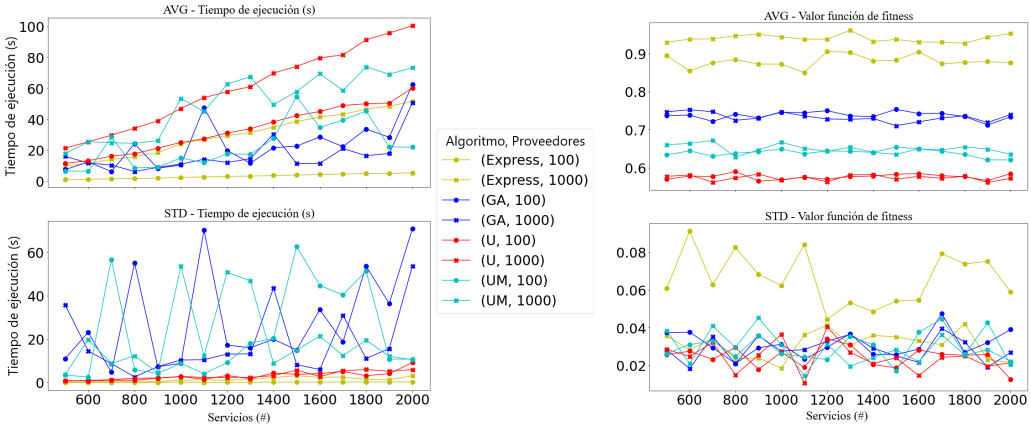
servicio se decide de forma aislada, simplemente tomando el mejor candidato, tendría una complejidad mucho menor: $100 \times 10 = 10^3$. La cuestión que nos planteamos entonces es: ¿cuál es la pérdida de calidad de la solución que podemos encontrar si disponemos de menos tiempo o de un tiempo limitado para encontrarla?

La figura 1.7 muestra los tiempos de ejecución y los valores de la función de *fitness* para cada una de estas tres alternativas (GA, U y Express), para diferentes números de servicios y proveedores candidatos. En estos gráficos, aparecen los algoritmos Express y GA como se mencionó anteriormente, y los algoritmos U y UM que utilizan dos nociones diferentes de utilidad, una en la que se consideran tanto la disponibilidad como el QoS global (U) y otra en la que solo se tiene en cuenta el QoS global (UM). Se han considerado aplicaciones de hasta 2000 servicios, con 100 y 1000 proveedores candidatos por servicio. Por ejemplo, para la solución U se denotan como (U, 100) y (U, 1000). Para llevar a cabo estos experimentos, se generaron aleatoriamente las arquitecturas de las aplicaciones y los valores de QoS de los proveedores candidatos. Dado que la ejecución de los diversos métodos puede variar considerablemente dependiendo ambos parámetros, cada experimento se repitió 10 veces. Se utilizó la misma aplicación en cada repetición para cada uno de los métodos distintos.

Aunque en [82] y en la sección 3.2 se discute con mucho más detalle la configuración y los resultados de experimentos como este, asumamos por ahora que los parámetros para los diferentes métodos están optimizados y centrémonos en las principales observaciones clave con respecto a nuestra discusión sobre escalabilidad.

El gráfico en la parte superior de la figura 1.7b muestra los valores de la función de *fitness* promedio para los experimentos utilizando los diferentes métodos. Podemos observar que los mejores valores de la función de *fitness* se obtienen con el método Express, seguido por el método basado en GA, y con los peores resultados para los basados en utilidad. También podemos observar que solo para el método Express el número de proveedores de servicio es significativo, mientras que para los demás métodos no se observa una diferencia significativa. El gráfico en la parte superior de la figura 1.7a, muestra los tiempos de ejecución para estos experimentos. Nuestra primera observación en este caso es que el crecimiento para todos ellos es más o menos lineal, con algunos picos para las soluciones basadas en algoritmos genéticos, es decir, GA, U y UM. Estos picos coinciden con la variabilidad observada en la gráfica de desviaciones estándar mostrada en la parte inferior de la figura 1.7a y son debidos a la influencia de la estructura de la aplicación en estos métodos. El método Express proporciona, como se esperaba, los mejores tiempos de ejecución, con una diferencia significativa sobre los resultados para los otros métodos a la que debemos añadir la estabilidad en los mismos, y con un crecimiento prácticamente insignificante a pesar del aumento en el número de servicios considerados. El segundo mejor es, en líneas generales, y a pesar de su variabilidad, GA.

Si tuviéramos que tomar una decisión sobre qué método usar al mirar estos gráficas, claramente elegiríamos el método Express por su aparente superioridad. Sin embargo, es importante destacar que cada método tiene sus propias ventajas y desventajas que pueden ser consideradas antes de tomar una decisión definitiva. Aunque los gráficos de las figuras presentadas sugieren que el método Express es



(a) Tiempo de ejecución (b) Valor de la función de fitness

Figura 1.7: Escalabilidad de los diferentes métodos

el claro ganador en términos de rendimiento y valor de la función de *fitness*, otros factores, como la reemplazabilidad, el cumplimiento de restricciones o no tener el tiempo como un elemento prioritario pueden influir en la elección del método más adecuado para un contexto particular. Estas consideraciones serán abordadas con más detalles tanto en [82] y la sección 3.2, y en el capítulo 4.

1.3. Gestión de recursos

La optimización de procesos de negocio es la práctica de aumentar la eficiencia organizativa mediante la mejora de los procesos. El objetivo principal de la gestión de procesos de negocio es optimizar los procesos para alcanzar los objetivos comerciales. Dado que la eficiencia es una de las principales herramientas cuantitativas en la toma de decisiones industriales, dos de los objetivos más comunes son maximizar el rendimiento y minimizar los costes, aunque hay otros parámetros, como podrían ser la sostenibilidad, impacto medioambiental, etc. que podrían ser considerados de forma similar. Por ejemplo, las empresas que utilizan un proceso de negocio, a lo largo del tiempo pueden beneficiarse enormemente al simplificar un flujo de trabajo y al aumentar el uso de recursos dentro de límites razonables de redundancia y costes.

Sin embargo, la optimización de procesos de negocio presenta un desafío complejo debido a su naturaleza multiobjetivo y al número de variables involucradas. Es crucial adaptar eficientemente los recursos disponibles a las demandas cambiantes, anticipándonos a los eventos del sistema. Esta anticipación nos permite no solo reducir los costes asociados a los recursos, sino también disminuir los tiempos de aprovisionamiento del sistema, lo que resulta fundamental para mejorar la eficiencia operativa y la competitividad empresarial. La integración de herramientas predictivas en las etapas de diseño de estrategias de optimización puede ayudarnos a alcanzar dichos objetivos. El uso de estas herramientas predictivas puede ayudarnos a comprender cómo gestionar dinámicamente los recursos, un aspecto esencial, ya que en la composición de servicios estos recursos permanecen estáticos, mientras que en entornos reales experimentan variaciones constantes a lo largo del tiempo, como

incrementos en la mano de obra, fallos en impresoras, adquisición de nuevos equipos, entre otros.

Cada vez son más los autores que usan estos modelos de aprendizaje profundo en aplicaciones empresariales, ya que sirven, entre muchas otras cosas, como base para monitorizar y predecir el comportamiento de los procesos (véanse, por ejemplo, [32, 70, 80]). Especialmente en ámbitos como la asignación de recursos, la optimización del tiempo y los costes, la monitorización de fallos, así como el descubrimiento de procesos, las aplicaciones hacen uso del registro de eventos del sistema. Estos registros, junto a otra información que pueda ser relevante, sirven como fuente de datos para entrenar los modelos de aprendizaje profundo que permiten predecir variables de interés. Por ejemplo, en la asignación de recursos, los registros de eventos pueden proporcionar información sobre el historial de utilización de los recursos. En el caso de la monitorización de fallos, el análisis de los registros de eventos puede identificar patrones que sugieran posibles problemas. Además, en el descubrimiento de procesos, los registros de eventos revelan la secuencia de actividades y decisiones tomadas en un proceso, lo que permite comprender mejor su funcionamiento y encontrar posibles áreas de mejora.

Los registros suelen contener información sobre los procesos en una organización en forma de listas de eventos. Cada evento está asociado a una instancia de proceso, que se identifica por un número de caso. Un caso se considera como una colección de actividades, es decir, tareas con atributos. Así, cada evento incluye un nombre o número de caso, una marca de tiempo y los recursos o costes asociados, entre otros atributos. Con esta información podríamos predecir el comportamiento de cada tarea individualmente, los movimientos de los tokens que simulan el comportamiento del proceso, o los tamaños de las colas de peticiones de recursos.

Existen muchas herramientas para construir modelos con la información de estos registros, como las regresiones logarítmicas, estrategias basadas en colas, Random Forest, etc. Sin embargo, las redes neuronales de memoria a largo plazo (LSTM) [106], explicadas en la sección 1.1.2, se han utilizado ampliamente para realizar predicciones de secuencias de datos (eventos, palabras, valores numéricos, etc.) con muy buenos resultados. En nuestro caso, las redes LSTM se entrenan para aprender de registros de eventos pasados y posteriormente predecir secuencias de "eventos próximos" que son más probables que ocurran según una ejecución parcial del proceso.

La simulación, llevada a cabo por un proceso Maude, se detendrá periódicamente en intervalos de tiempo específicos, los cuáles serán especificados mediante un parámetro denominado *Time Between Checks* (TBC), que analizaremos en el capítulo 4. Durante estos intervalos, se recopilará información sobre el porcentaje de uso de los recursos y la cantidad de réplicas. Esta información será monitorizada para proporcionar a la red neuronal una visión actualiza del estado del sistema y así ayudar en la predicción.

La figura 1.8 muestra el diseño lógico detrás del enfoque propuesto. Esta semántica se ha diseñado y construido para leer registros de eventos, por ejemplo, a través de sockets, mantener el estado del proceso e interactuar con heurísticas predictivas para mejorar la asignación de recursos. En nuestro enfoque se mantiene un mode-

Figura 1.8: Interacción de Maude entre una red neuronal (izquierda) y un tiempo de ejecución de proceso BPMN (derecha).

lo del sistema construido mediante una representación del proceso de negocio en Maude. Sobre este modelo del sistema bajo control, se utilizan distintas estrategias de asignación de recursos para tomar decisiones dinámicamente. Sin embargo, esta tesis se centra en la estrategia predictiva basada en redes neuronales para gestionar la asignación de recursos.

El proceso mediante el cual hemos identificado la necesidad de desarrollar estas estrategias, su desarrollo y cómo evolucionan a lo largo del tiempo se describe como una estrategia típica de *MAPE-K*. Estas estrategias están basadas en el aprendizaje autoadaptativo, que se componente de cuatro fases:

1. *Monitor* (Monitorización): En esta fase se recopila de forma continua los datos del entorno y del sistema en tiempo de ejecución. Estados datos, por lo general, incluyen métricas de rendimiento, estado del sistema, cambios en las condiciones del entorno, entre otros.

2. *Analyze* (Análisis): En esta etapa, los datos recopilados durante la fase de monitorización se analizan para identificar patrones, tendencias o anomalías. El objetivo es comprender el estado actual del sistema y su entorno, así como anticipar posibles problemas o necesidades futuras.

3. *Plan* (Planificación): En base al análisis realizado, se desarrollan planes o estrategias para adaptar el sistema de manera óptima a las condiciones cambiantes del entorno o a los requisitos del usuario. Esto podría impactar en la selección de nuevas configuraciones, ajuste de parámetros, cambios en las políticas de operación, entre otros.

4. *Execute* (Ejecución): En esta fase, se implementan las estrategias elaboradas durante la etapa de planificación. Esto puede incluir la aplicación de cambios en el sistema en tiempo real, la configuración de componentes, la asignación dinámica de recursos, entre otras acciones.

5. *Knowledge* (Conocimiento): Por último la K representa el conocimiento, que si bien no representa ninguna fase como tal, se acumula y se utiliza en cada una de las fases anteriores. Este conocimiento puede ser previamente definido por expertos, aprendiendo a partir de datos históricos o adquirido durante la ejecución del sistema. El conocimiento se utiliza para guiar y mejorar el proceso de autoadaptación en cada iteración de las fases *MAPE*.

Para la parte de predicción dentro de este trabajo, utilizamos un modelo estructurado en dos capas para la optimización de procesos. En la primera capa se utiliza la capacidad de predicción del aprendizaje profundo para anticipar, dada una ejecución parcial, los siguientes eventos en la traza. Con esta predicción, continuaremos en la segunda capa donde realizaremos una estimación de la cantidad adecuada de los recursos necesarios, realizando operaciones de aprovisionamiento o liberación de los recursos. Esta segunda capa es la mencionada en la sección 1.1.3, donde se ejecutan concurrentemente las múltiples instancias que puede competir por los mismos recursos finitos. Nuestro objetivo principal es la gestión dinámica de estos recursos.

La integración de estas dos capas da como resultado una heurística que permite analizar y optimizar la asignación de recursos de los procesos de negocio. Esta técnica permite un análisis cuantitativo, como calcular la mejor combinación de valores de parámetros que reduzcan los costes, los tiempos de procesamiento o el desperdicio de recursos.

Dado un proceso de negocio B y una traza parcial t de tareas en B, se utiliza una red LSTM, denominada $\mathscr{N}_B$, para predecir una extensión de t que cumple con B basándose en el entrenamiento previo de $\mathscr{N}_B$ con registros de eventos de B.

Necesitamos un modelo que refleje la evolución del flujo de trabajo ante los eventos que recibe y los recursos disponibles. La semántica expresada mediante lógica de reescritura simplifica este proceso de modelado, permitiendo una notación y representación adecuada de los procesos de negocio. Esta especificación en lógica de reescritura, que se implementa en Maude, tiene la capacidad de simular el comportamiento concurrente de las instancias de B bajo un conjunto específico de restricciones sobre los recursos, consultando a $\mathscr{N}_B$, y ajustando la cantidad de réplicas de los recursos en tiempo de ejecución.

Dada la ejecución concurrente parcial de un número de instancias de B en la semántica de Maude, se consulta a $\mathscr{N}_B$ con una ventana de tiempo. Devolviendo una secuencia de eventos que extienden las trazas de las instancias dadas, es decir, predice continuaciones para las trazas en ese período de tiempo particular. Estas predicciones se utilizan luego para ajustar la cantidad de réplicas por recurso en tiempo de ejecución en el modelo de Maude que representa B.

Para entrenar la red neuronal $\mathscr{N}_B$, el enfoque propuesto toma como entrada un proceso BPMN, denominado B, y un conjunto de trazas T_B de B. El proceso B tiene información sobre el tipo de recursos necesarios para completar las tareas. Las trazas T_B representan ejecuciones de B en, por ejemplo, un entorno de producción. Cuando la red neuronal haya terminado su entrenamiento, se introducirán nuevas trazas generadas que darán lugar a la predicción de los eventos siguientes.

Estas predicciones se utilizan en última instancia para predecir cómo se deben asignar/liberar recursos para optimizar su uso; por ejemplo, para minimizar el tiempo que un recurso no se está utilizando o, de manera similar, para maximizar el uso de recursos que cumplen con ciertas restricciones de presupuesto y tiempo asociadas a la ejecución del proceso.

La estrategia para asignar/liberar recursos automáticamente sigue un diseño introducido en [39]. En la sección 3.3 se puede encontrar la información completa

sobre el problema abordado en este apartado. Todas las especificaciones Maude y el código Python, junto con ejemplos, modelos y salidas, están disponibles en un repositorio público de GitHub [34].

1.4. Verificación y análisis estadístico de aplicaciones

La verificación formal de sistemas es otro pilar fundamental en la escalabilidad de las aplicaciones, especialmente en contextos de sistemas basados en tiempo real. La verificación de sistemas de tiempo real ha sido objeto de investigación y ha dado lugar al desarrollo de diversas herramientas, entre las que se encuentran *UPPAAL* [10], *PRISM* [83], *CADP* [49], *SPIN* [94] y otros model checkers, cada uno con sus propias capacidades y enfoques.

UPPAAL, por ejemplo, se centra en el modelado, validación y verificación de sistemas en tiempo real modelados como redes de autómatas temporizados. PRISM, por otro lado, está especializado en sistemas probabilísticos, permitiendo el análisis de propiedades cuantitativas y cualitativas bajo incertidumbre. CADP ofrece un enfoque más general, permitiendo la especificación y verificación de sistemas concurrentes mediante técnicas como el model checking y el testing. SPIN, por su parte, se destaca en la verificación de sistemas concurrentes mediante el model checking de sistemas concurrentes especificados en el lenguaje PROMELA.

A pesar de las ventajas y capacidades específicas de estas herramientas, se ha observado que la verificación forma de sistemas a menudo enfrenta desafíos relevantes en términos de escalabilidad. Esto se debe en parte a la complejidad de muchos problemas de verificación, así como a la necesidad de manejar sistemas cada vez más grandes y complejos. A medida que los sistemas crecen en tamaño y en complejidad, las herramientas de verificación mencionadas pueden encontrar dificultades para manejar de forma eficiente el espacio de estados generado durante la propia verificación. En consecuencia, aunque estas herramientas son comúnmente usadas para verificar propiedades específicas en sistemas de tamaño moderado, su utilidad puede verse limitada en entornos donde la escalabilidad es un factor relevante. Por otra parte, en sistemas reales, más que propiedades cualitativas, puede ser interesante el poder hacer afirmaciones sobre valores cuantitativos. Por ejemplo, es difícil construir un sistema que no falle nunca, y si asumimos que los sistemas van a fallar, lo que nos interesa es garantizar que el porcentaje de fallos estará por debajo de un umbral determinado, o que tras un fallo, el sistema se recuperará en menos de una cantidad determinada de tiempo, posiblemente en un porcentaje de veces superior a un umbral dado. Si nos marcamos como requisitos el tratar este tipo de propiedades, y además hacerlo de una forma escalable, el model checking estadístico [90] parece la técnica más recomendable.

Herramientas como PRISM, Storm [51] o VeStA [91]/PVeStA [5]/MultiVeStA [44] ofrecen la posibilidad de hacer model checking estadístico. Mientras que PRISM utiliza su propio lenguaje de entrada, y Storm utiliza el lenguaje de PRISM, MultiVeStA permite analizar ejecuciones de sistemas especificadas utilizando distintos simuladores de eventos discretos. En nuestro caso, hemos utilizado MultiVeStA con simulaciones realizadas en el simulador de P.

Una vez tenemos definida formalmente la aplicación, podemos realizar varias simulaciones sobre el mismo sistema, mediante un sistema de simulaciones de Monte Carlo. Estas simulaciones nos permiten hacer una inferencia estadística sobre los resultados obtenidos. Como ya hemos mencionado, existen numerosas herramientas que pueden realizar este tipo de simulaciones, pero MultiVeStA nos permite hacerlo de forma escalable lanzando las ejecuciones en diferentes servidores, además de permitir verificar sistemas modelados utilizando distintos generadores de eventos discretos.

Por otra parte, existe una gran variedad de herramientas disponibles para analizar flujos de trabajo y procesos de negocio, pero algunas tienen limitaciones en cuanto a expresividad del lenguaje. Una alternativa para modelar este tipo de sistemas son los diagramas de estados, los cuales son ampliamente utilizados. Una implementación robusta de estos diagramas es P [25]. P es lenguaje desarrollado por Microsoft junto con investigadores de la Universidad de Berkeley, California, y utilizado por gigantes de la industria como Microsoft y Amazon.

Dada la inherente complejidad de los sistemas distribuidos, P puede beneficiarse de otros enfoques de métodos formales para abordar una tarea tan compleja. Además, P actualmente no dispone de la opción de ejecutarse de forma distribuida, lo que limita su escalabilidad.

Esta limitación sobre la escalabilidad se puede solventar mediante la integración de MultiVeStA [89] dentro del entorno de ejecución y verificación de P. De esta forma, gracias a la integración de MultiVeStA y el simulador de P, ahora es posible verificar sobre programas P fórmulas cuantitativas de lógica temporal del tipo "¿cuál es el número esperado de veces que un recurso está disponible antes de dejar de funcionar?" o "¿cuál es la probabilidad de que un recurso esté disponible después de *x* usos?" expresadas en el lenguaje QuaTEx [3]: su corrección estadística se verifica con simulaciones de Monte Carlo utilizando muestreo probabilístico.

MultiVeStA es una herramienta para la verificación estadística distribuida de modelos que puede operar junto con diferentes simuladores de eventos discretos. Como parte de esta tesis, se ha integrado P con MultiVeStA. Los detalles de esta integración se encuentran en la sección 1.6.3, donde se describe cómo se implementaron funciones específicas para observar el estado de un programa y coordinar las simulaciones de Monte Carlo. Esto incluye una nueva funcionalidad en el compilador de P para generar automáticamente código para cualquier programa dado. El resultado general de este esfuerzo es que el simulador de P aún se puede ejecutar de manera independiente o bajo el control de MultiVeStA, que ahora puede establecer una semilla aleatoria para el muestreo, realizar una ejecución paso a paso y consultar los estados de las variables en programas P. Para sus verificaciones, MultiVeStA ejecutará tantas simulaciones independientes como sea necesario para respetar un intervalo de confianza especificado por el usuario. Además, MultiVeStA ofrece una arquitectura cliente-servidor para distribuir simulaciones, lo que permite su ejecución concurrente y la ampliación del análisis.

La integración de ambas herramientas se presenta en el trabajo [33] de la sección 3.4, donde se ilustra la integración con la ayuda de un ejemplo de un sistema de uso compartido de bicicletas. El ejemplo está inspirado en un caso de estudio

presentado originalmente en [9], basado en un sistema público real. En dicho trabajo, se analizan tres propiedades:

- ¿Cuántas veces una bicicleta será usada antes de ser desechada completamente?
- ¿Qué probabilidad existe de que una bicicleta sea usada más de un determinado número de ocasiones antes de que sea desechada?
- ¿Que porcentaje del tiempo está en uso una bicicleta antes de desecharse?

En el mismo trabajo, se realiza también un análisis sobre la escalabilidad de esta integración. Se presenta información sobre el rendimiento de la integración tanto en una máquina local como en una configuración de instancias de AWS. Un resumen de este análisis se encuentra en el capítulo 4. En él, se muestra cómo distribuir las simulaciones en varias instancias reduce significativamente los tiempos de ejecución mejorando considerablemente la escalabilidad del sistema.

1.5. Contribuciones

El reto de la escalabilidad en la gestión de aplicaciones basadas en servicios es crucial y se puede desglosar en tres aspectos fundamentales: la composición de servicios, la gestión dinámica de recursos y la verificación de aplicaciones basadas en servicios. Estos tres aspectos, constituyen los pilares para abordar eficazmente la escalabilidad y el análisis de las aplicaciones en entornos de servicios.

Esta tesis se apoya en una serie de trabajos previamente publicados que profundizan en el análisis y la resolución de los problemas relacionados con la escalabilidad en cada uno de estos ámbitos. A través de estos trabajos, se examinan detalladamente los desafíos específicos que surgen en la composición de servicios, la gestión dinámica de recursos y la verificación de aplicaciones, proporcionando un entendimiento más completo de cómo enfrentar estos problemas en diferentes contextos y escenarios de implementación. Los siguientes trabajos se encuentran anexados en las diferentes secciones del capítulo 3:

[81] Nicolás Pozas and Francisco Durán. “On the scalability of compositions of service-oriented applications”. Artículo presentado en International Conference on Service-Oriented Computing (ICSOC 2021). Lecture Notes in Computer Science, vol 13121. Springer, 2021. Anexado en la sección 3.1.

[82] Nicolás Pozas, Francisco Durán, Katia Moreno Berrocal and Ernesto Pimentel. “Location-Aware Scalable Service Composition”. Artículo publicado en Journal of Software: Practice and Experience (2023). Anexado en la sección 3.2.

[35] Francisco Durán, Nicolás Pozas and Camilo Rocha. “Business Processes Resource Management using Rewriting Logic and Deep-Learning-based Predictive Monitoring”. Artículo publicado en Journal of Logical and Algebraic Methods in Programming (2023). Anexado en la sección 3.3.

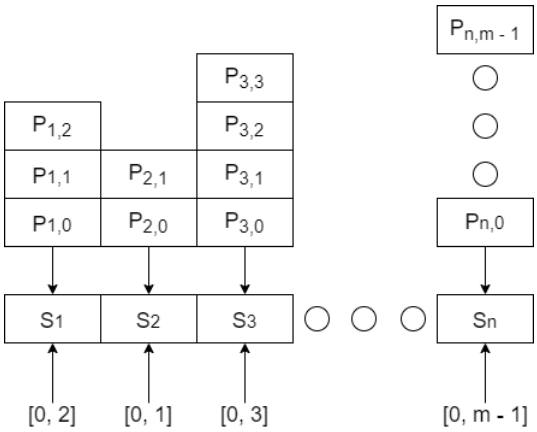


Figura 1.9: Codificación

[33] Francisco Durán, Nicolás Pozas, Carlos Ramírez and Camilo Rocha. "Statistical Model Checking for P". Artículo publicado en Formal Methods for Industrial Critical Systems, FMICS (2023). Lecture Notes in Computer Science, vol 14290. Springer, 2023. Anexado en la sección 3.4.

1.6. Implementación

En esta sección, describiremos en detalle la implementación de los algoritmos genéticos utilizados en nuestro trabajo, abordando aspectos como su codificación, evaluación y los híper parámetros empleados. Además, presentaremos la arquitectura de la red neuronal que hemos usado para la predicción de tareas. Por último, también revisaremos la integración entre la herramienta MultiVeStA y el lenguaje P para llevar a cabo el model checking estadístico.

Para facilitar el acceso a los resultados de nuestro trabajo, se proporcionan las siguientes direcciones web de los repositorios:

- Composición de servicios: `https://github.com/Pozas91/Locassc`.
- Gestión de procesos: `https://github.com/Pozas91/bpmn-resource-manager`.
- Verificación de procesos: `https://github.com/PST-P/p-pst`.
- Resultados de los experimentos de los diferentes algoritmos de composición de servicios: `http://services-composition-web.s3-website-us-east-1.amazonaws.com/`.

1.6.1. Algoritmos genéticos

Codificación y evaluación

En la figura 1.9 se puede ver un ejemplo de cómo se ha realizado la codificación de la aplicación. Como se ha mencionado anteriormente en la sección 1.1.1, la representación de esta codificación debe reflejar las características relevantes del

problema. Por tanto, hemos codificado el problema como una lista de longitud n_s, siendo n_s el número servicios ($S_1, S_2, S_3...$). Cada uno de estos servicios tiene disponible un catálogo de proveedores disponibles que pueden estar compartidos por los demás servicios. Estos proveedores ($P_{1,0}, P_{1,1}, P_{1,2}, P_{2,0}...$, siendo $P_{i,j}$ el proveedor j del servicio i) estarán identificados mediante un valor entero que está en el rango $[0, n_p)$, siendo n_p el número de proveedores total para ese servicio en concreto. A su vez, tenemos las fórmulas especificadas en los trabajos [81, 82] de las secciones 3.1 y 3.2, que nos devolverá un valor en el rango $[0, 1]$, como se explica en [81]. Con esta información, la codificación de las aplicaciones descritas serán vistas como una lista de tamaño n_s, donde cada elemento se corresponde con un cromosoma que contiene la información de cada uno de los servicios. Dentro de este cromosoma, se encuentra el gen donde se almacena la información del índice que identifica el proveedor asignado para ese servicio en ese individuo. Para explicar estos conceptos, podemos ayudarnos de la aplicación descrita por la figura 1.5 de la sección 1.2.1. En dicha aplicación, el número de servicios representado por n_s es igual a 7, lo que implica que contamos con una lista de tamaño 7. Los proveedores para cada servicio también estarían especificados mediante listas, dando lugar al siguiente catálogo:

$$
\begin{aligned}
P_{S_1} &= [P_1, P_2], \\
P_{S_2} &= [P_7, P_5], \\
P_{S_3} &= [P_6, P_3], \\
P_{S_4} &= [P_3, P_4], \\
P_{S_5} &= [P_1], \\
P_{S_6} &= [P_3, P_5], \\
P_{S_7} &= [P_1, P_8, P_3].
\end{aligned}
$$

Con todo esto, un posible ejemplo de codificación sería $[[1], [0], [1], [1], [0], [1], [2]]$, donde S_1 estaría desplegado en P_2, S_2 en P_7, S_3 en P_3, S_4 en P_4, S_5 en P_1 (su único proveedor disponible), S_6 en P_5 y S_7 en P_3. Sin embargo, una codificación incorrecta sería $[[1], [0], [2], [1], [1], [1], [2]]$, ya que ni S_3 cuenta con tres proveedores en su catálogo, ni S_5 cuenta con dos proveedores.

Híper parámetros

Para encontrar los híper parámetros correctos para nuestras pruebas se han realizado múltiples experimentos donde se cambian cada uno de los valores sistemáticamente hasta encontrar el siguiente conjunto de valores:

- Población: 100.
- Probabilidad para mutar cualquier cromosoma: 13 %.
- Probabilidad de entrecruzamiento: 70 %, que puede afectar a un total de 5 cromosomas diferentes al mismo tiempo (el entrecruzamiento multipunto se ha establecido en 5).

- Selección del superviviente: Elitista de 2 individuos. Este parámetro asume algo de riesgo, ya que es posible arrastrar individuos que son mínimos locales durante la ejecución del GA, sin embargo, si esos dos individuos son prometedores su mutación y entrecruzamiento es probable que dé como resultado mejores individuos.

- Selección de descendientes: Los individuos escogidos para generar nuevos descendientes son elegidos mediante el método *RouletteWheelSelector*, donde la probabilidad de que un individuo sea elegido será generada mediante la función $P(i) = \frac{f_i}{\Sigma_{j=0}^{N-1} f_j}$, donde N es la población total de individuos.

1.6.2. Redes LSTM

Las redes LSTM utilizadas en este trabajo se han implementado en Python con las librerías de Keras [58] y TensorFlow [99]. Existe una amplia variedad de tipos de redes neuronales entre los cuales elegir. Se ha optado por utilizar este tipo de redes neuronales debido a que son las que mejor se adaptan a las estructuras de datos que estamos manejando para predecir las trazas de los procesos BPMN. Esto se debe a su capacidad para predecir secuencias de datos a largo plazo, coincidiendo con la naturaleza de las trazas BPMN, las cuales están organizadas precisamente de esta manera.

En la figura 1.10 se puede observar el modelo completo con las diferentes capas de la red. El modelo comienza con tres entradas, donde la red recibe información sobre las actividades, un identificador del token y la información sobre el uso de recursos actual del sistema. Estas tres entradas son concatenadas como una sola entrada, y se normaliza para que todos los valores tengan la misma relevancia dentro de la red neuronal. Estos valores normalizados van atravesando cada una de las tres capas LSTM de las que se compone la red, se normaliza el resultado de la última capa y se divide de nuevo en tres capas de salida. La primera ofrece información sobre la siguiente actividad de la traza, la segunda ofrece información sobre cuando va a llegar esa actividad, y la tercera, ofrece información sobre el uso de recursos que tendrá en ese momento.

La integración de la capa de predicción con Maude se realiza mediante la comunicación por sockets: la semántica en lógica de reescritura implementada en Maude se comunica con el programa ejecutado en Python para intercambiar mensajes. En estos mensajes, Maude solicitará a Python una predicción utilizando la información que le proporciona.

1.6.3. Integración entre MultiVeStA y P

En esta sección, describiremos la interacción entre la herramienta MultiVeStA y el lenguaje de programación P. Esta integración se representa mediante el diagrama de la figura 1.11. En dicho diagrama observamos dos bloques principales: *MultiVeStA Core* y *P Simulator*.

MultiVeStA Core recibe las propiedades descritas en el pseudo-lenguaje Multi-QuaTEx (*QuaTEx Properties*), y establece una comunicación continua con el bloque

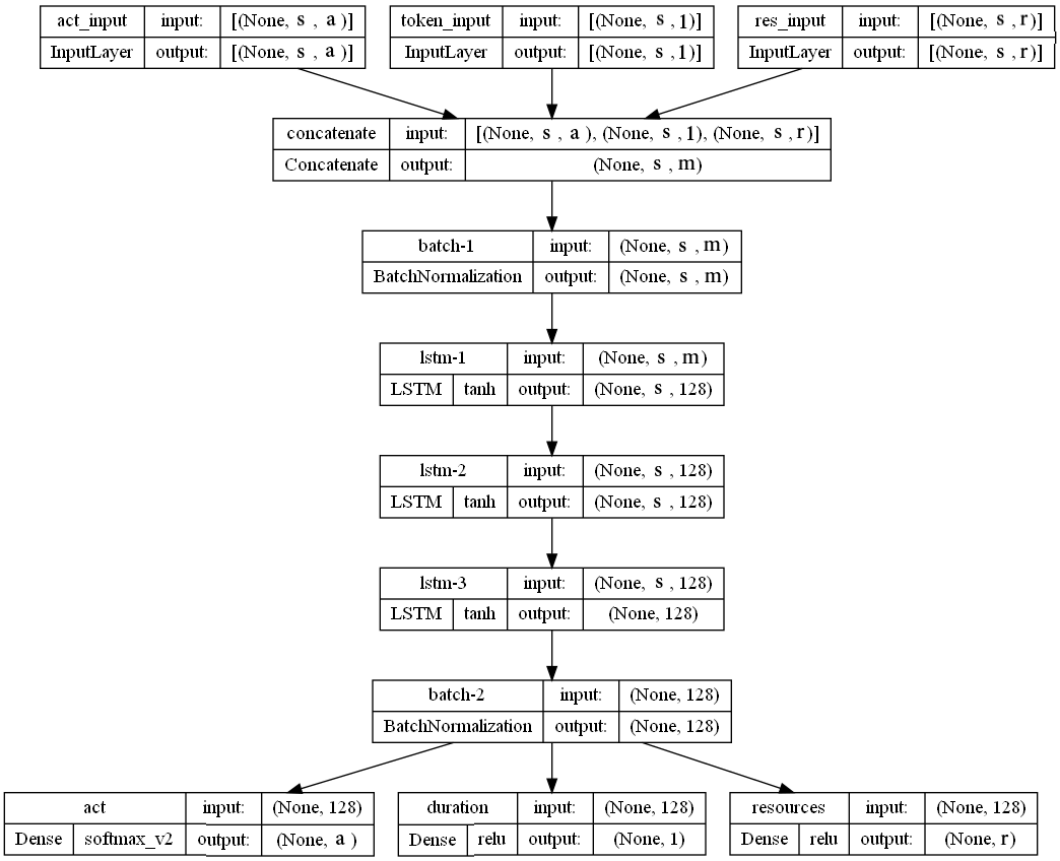


Figura 1.10: Modelo LSTM

P Simulator mediante intercambio de mensajes haciendo uso de la clase escrita en Java *PState*. Esta clase, tal y como aconsejan los desarrolladores de MultiVeStA, extiende de la clase predefinida *NewState* para facilitar la interacción entre los diversos lenguajes de programación con los que queremos que interactúe su herramienta. Dentro de esta clase, se emplea la librería *ExpectJ*, que toma el control de una consola del sistema para ejecutar los diferentes comandos de la interacción. En esta clase *PState* se deben completar varios métodos incluyendo *setSimulatorForNewSimulation*, el cual utiliza la librería *ExpectJ* para ejecutar una instancia del simulador de P (*P Simulator*). Además, existe la función *performOneStepOfSimulation*, encargada de ejecutar un paso en la simulación de MultiVeStA y comunicarlo al simulador de P. Asimismo, una función llamada *performWholeSimulation* ejecuta toda la simulación de forma continuada hasta que se complete la simulación. Una función crucial es *rval*, que puede recibir un valor entero o una cadena de texto como parámetro. y se encarga de consultar al simulador de P sobre el valor de la variable indicada. Entre las variables más comunes y útiles se encuentran: Variables que responde a las siguientes preguntas, ¿ha finalizado la simulación?, ¿cuántos pasos ha dado el simulador? o ¿cuánto tiempo ha transcurrido desde el inicio de la simulación?, entre otras. Por último, la función *getIsSimulationFinished* pregunta al simulador de P en cada paso de ejecución si ha concluido la simulación.

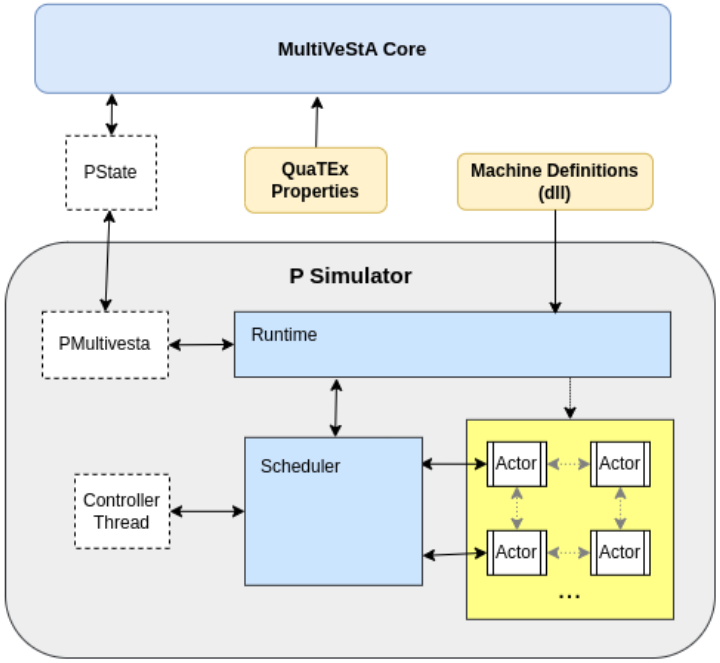


Figura 1.11: Arquitectura de la integración entre MultiVeStA y P

La mayoría de estas funciones se comunican con la clase *PMultivesta*, dentro del bloque del simulador de P. Esta clase recibe los mensajes mandados por la clase *PState*, consulta el valor correspondiente y lo devuelve a la clase *PState*. Además, *PMultivesta* también actúa como una especie de semáforo para el *Runtime* de P, deteniéndolo en cada paso de ejecución para asegurar que tanto el estado en MultiVeStA como el estado en el simulador de P estén sincronizados. Este control se logra interactuando con un proceso (*ControllerThread*) que se ejecuta en un nuevo hilo y actúa como un candado para el planificador de P (*Scheduler*).

Antes de llevar a cabo la simulación, P requiere compilar el código escrito en P a código C#. Al utilizar esta herramienta de compilación a C#, podemos agregar automáticamente algunas funciones al código C#, lo que nos permiten consultar el estado en el que se encuentra una determinada máquina, el valor de las variables numéricas de una máquina, y determinar si una máquina ha sido ya lanzada. Las funciones añadidas nos permiten consultar las siguientes variables mediante el uso de la función *rval*:

- alive.*MACHINE.INDEX*: Evalúa a 1 si la máquina *MACHINE* con índice *INDEX* ha sido creada y 0 en caso contrario.
- state.*MACHINE.INDEX.STATE.EVENT*: Evalúa a 1 si el evento *EVENT* dentro del estado *STATE* ha sido el último evento consumido por la máquina *MACHINE* con índice *INDEX*, y a 0 en caso contrario. Si no se especifica el evento, entonces se comprobará que el estado *STATE* ha sido el último estado visitado por la máquina.

- *MACHINE.INDEX.PROPERTY*: Devuelve el valor de la propiedad *PROPERTY* en la máquina *MACHINE* con el índice *INDEX*. Estas propiedades solo pueden ser valores numéricos. Los valores booleanos se representan como 1 si el valor es *true* y como 0 si el valor es *false*.

De esta forma, hemos logrado enriquecer el pseudo-lenguaje MultiQuaTEx y realizar consultas más interesantes.

2. Trabajos relacionados

La gestión eficiente de aplicaciones basadas en micro servicios IoT constituye un desafío complejo que implica abordar varios subproblemas clave. Durante el desarrollo de esta tesis, hemos recopilado una cantidad considerable de literatura relevante [1-116], la cual proporciona una amplia gama de conocimientos sobre las ventajas y desventajas de las soluciones propuestas. En esta etapa de la investigación, hemos llevado a cabo una revisión exhaustiva de la literatura existente, centrándonos en tres subapartados que abordan los aspectos críticos de la escalabilidad en la gestión de aplicaciones basadas en servicios IoT:

1. La composición de servicios.
2. La gestión de recursos.
3. La verificación y análisis de la aplicación.

2.1. Composición de servicios, canales de comunicación y reemplazabilidad

La revisión sistemática de la literatura de Strunk [95] resume, clasifica y evalúa los principales esfuerzos de investigación sobre la composición de servicios enfocados a los atributos de calidad de servicio (QoS). Aunque el trabajo es de 2010, la mayor parte del análisis creemos que sigue bastante vigente. La revisión concluye que, dado el tiempo y los costes exponenciales del problema de composición, la mayoría de los enfoques intentan simplificar el problema ya sea linealizando la función objetivo, considerando la maximización local de los QoS, no considerando restricciones de QoS, considerando la optimización de un solo objetivo o seleccionando una solución subóptima al problema.

Más recientemente, Arellanes y Lau [8] evalúan diferentes mecanismos de composición de servicios para sistemas IoT, con un enfoque en su escalabilidad. En lugar de la escalabilidad horizontal/vertical usual, [93] se centra, como nosotros, en la escalabilidad funcional, donde la escalabilidad se estudia en términos del número de servicios compuestos. Aunque Arellanes y Lau proporcionan una interesante evaluación de los mecanismos de composición, admitiendo flujos de datos distribuidos, transparencia en la ubicación de servicios y otras características, proponen un modelo algebraico capaz de emparejar servicios existentes. En nuestro caso, asumimos que esta coincidencia ha resultado previamente en una oferta de proveedores de servicios candidatos para cada uno de los servicios de una aplicación. En la misma línea, Chen et al. [19] proponen el uso de filtrado colaborativo distribuido para seleccionar retroalimentación utilizando la calificación de similitud de relaciones de amistad, contacto social e intereses de la comunidad. Utilizando estas calificaciones, proporcionan una técnica de filtrado adaptativo para determinar la mejor manera de combinar confianza directa e indirecta para minimizar el tiempo de convergencia y el sesgo de estimación de confianza. La escalabilidad horizontal/vertical ha sido

estudiada extensamente, por ejemplo, en trabajos de Calinescu et al. [14], Coutinho et al. [23], Xu y Helal [108], Vakili y Navimipour [102], y Cabre et al. [13].

En su artículo de 2022, Razian et al. [85] presentan una revisión de la literatura sobre los estudios existentes en la composición de servicios en entornos dinámicos, con un enfoque en la consideración de la incertidumbre. Dada la variabilidad de los valores de QoS en entornos dinámicos del mundo real, la estimación/predicción de QoS en la composición de servicios se ha vuelto un reto mayor, ya que la información de QoS disponible puede ser incompleta o desconocida. El estudio revela que los métodos utilizados por los trabajos incluidos en el estudio para resolver el problema de composición bajo incertidumbre son probabilísticos, de aprendizaje automático, de sistemas difusos y de sistemas de recomendaciones. Según este estudio, aquellos trabajos preocupados por la escalabilidad utilizan algoritmos basados en meta-heurísticas. Sin embargo, el estudio se enfoca en atributos específicos del servicio, sin prestar atención a sus ubicaciones y latencias: el 61 % de los trabajos analizados considera el tiempo de respuesta; la disponibilidad, confiabilidad y throughput es considerado por entre el 20 % y el 28 % de ellos; el coste, la reputación, la seguridad, la energía y otros atributos se consideran en porcentajes más pequeños. Uno de los trabajos más prometedores, en cuanto al tiempo de respuesta, podría ser el de Parejo et al. [77], donde se utiliza un enfoque híbrido que combina GRASP con Path Relinking [47].

Es práctica común tomar como entrada del problema de composición una descripción de la arquitectura de la aplicación, donde se hace explícito el orden de los nodos y sus dependencias. Aunque hoy en día otras notaciones pueden ser más activas, las notaciones más utilizadas en la literatura para proporcionar estas descripciones son WS-BPEL [7] y BPMN [12] —véanse, por ejemplo, los trabajos de Siddiqui et al. [92], Le et al. [64], Ouyang et al. [76], Durán y Salaün [42], Kheldoun et al. [59] o Festa et al. [47]—. En ellos, las estructuras de control se definen como entidades que regulan el flujo de ejecución, y dependiendo del trabajo, se usarán más o menos estructuras de control, como condiciones, secuenciales, iterativas, paralelas, etc.

Se han propuesto una amplia gama de soluciones para el problema de composición. Los métodos de resolución suelen dividirse en métodos exactos y métodos basados en heurísticas. El uso de otros métodos, como, por ejemplo, colonias de hormigas [116], programación dinámica [111] o técnicas de divide y vencerás [81], también han sido explorados por diferentes autores. Un resumen más amplio de las diferentes alternativas se puede encontrar en el estudio de Strunk [95]. Entre los métodos exactos, las soluciones más exitosas se basan en la programación entera —véanse, por ejemplo, los trabajos de Zeng et al. [115] o Tao et al. [111]—. Los algoritmos basados en métodos exactos buscan todas las composiciones posibles, y, para cada una de las composiciones, se calcula la función objetivo. Se selecciona entonces la composición con el mejor valor para la función objetivo como resultado. Típicamente, los enfoques de selección pueden basarse en decisiones locales o globales. Dependiendo de qué enfoque se siga, se pueden considerar por tanto restricciones locales o globales. Por ejemplo, Yu et al. [111] proponen una solución que utiliza un modelo combinatorio y un modelo de grafos. En él, el modelo combinatorio define el problema como un problema de mochila 0-1 multidimensional y de múltiples

opciones; el modelo de grafos define el problema como un problema de camino óptimo de múltiples restricciones. En su trabajo se puede observar que el tiempo de ejecución de los métodos exactos crece exponencialmente a medida que aumenta la entrada, volviéndose intratable [111, 115]. El mayor beneficio de estos métodos es que proporcionan soluciones exactas, garantizando soluciones óptimas globales.

En [107] se utiliza un enfoque basado en restricciones para eliminar las restricciones especificadas por el usuario que pueden ser redundantes, reduciendo la complejidad del plan de composición y mejorándolo. Varios autores [45, 46, 115] han intentado reducir el espacio de búsqueda minimizando los servicios dentro del plan de composición. Aunque el problema y la solución son muy diferentes, se centran, al igual que nosotros, en la complejidad del problema para aplicaciones con un gran número de servicios.

Como alternativa a los métodos exactos, se han propuesto algunas soluciones que buscan un equilibrio entre el tiempo de ejecución y la precisión. Al seleccionar proveedores de servicios a nivel de servicio [81, 115] se pueden obtener buenos aumentos de velocidad. Una de esas técnicas es el uso del algoritmo de divide y vencerás. Sin embargo, como señalan Berbner et al. [11], a pesar de los grandes aumentos de velocidad proporcionados por el método de divide y vencerás, el uso de este método no brinda la mejor solución. De hecho, al analizar el problema globalmente, cuanto mayor sea el número de servicios, más alejado estará del óptimo. Otro problema con los métodos exactos es su dificultad para establecer restricciones globales. Yu y Lin [110] proponen un método basado en divide y vencerás en el que, para tener en cuenta las restricciones globales, proponen hacer varias ejecuciones del algoritmo, perdiendo la velocidad que conlleva el método de divide y vencerás.

En [30], Deshpande y Sharm proponen el uso de un clasificador de aprendizaje automático para elegir entre los métodos exactos, los métodos basados en colonias de hormigas y los métodos basados en algoritmos genéticos dependiendo de la complejidad del problema. Aunque la decisión depende de múltiples factores, en general, para problemas con pocos servicios, el clasificador recomendaba los métodos exactos, mientras que para problemas más complejos recomendaba el uso de la solución basada en algoritmos genéticos. De hecho, los algoritmos genéticos son ampliamente aceptados para resolver problemas de composición, ya que obtienen resultados suficientemente buenos en tiempos pequeños. Sin embargo, aunque son una de las mejores alternativas actuales, presentan serios problemas de escalabilidad. Este trabajo no compara todos los algoritmos revisados mencionados aquí, pero es un trabajo interesante donde se presenta el uso de un clasificador para elegir entre las diversas opciones dependiendo de las características de la aplicación.

La mayoría de las propuestas que utilizan heurísticas se basan en algoritmos genéticos [17, 18, 45, 46, 74]. En lugar de explorar todo el espacio de soluciones candidatas, como hacen los métodos exactos, los algoritmos genéticos exploran solo una parte de este espacio, y esta exploración se basa en la evolución de las soluciones. La debilidad principal de los algoritmos genéticos es que se basan principalmente en estadísticas, y puede haber casos en los que estos algoritmos caigan en un mínimo local. Cuando el algoritmo genético está en esta situación puede ser difícil salir de ella. Una buena selección de los híper parámetros que controlan la ejecución del algoritmo

es clave para obtener los mejores resultados en los tiempos más cortos. Estos híper parámetros controlan básicamente el tipo de mutaciones a las que están sujetos los genes y los criterios para decidir cuándo la solución es lo suficientemente buena, es decir, para detenerse. Estos algoritmos reducen significativamente el espacio de búsqueda, y por tanto la complejidad, ofreciendo un equilibrio entre tiempo y calidad de solución, y debido a esto es utilizado por varios autores. El mayor problema de las soluciones basadas en algoritmos genéticos, como todos los métodos basados en heurísticas, es que no es posible saber qué tan cerca se encuentran de la solución óptima, ni siquiera si ya tenemos la mejor solución. Además, incluso con los mejores híper parámetros, y aunque se comprometa algo de calidad en las soluciones, el crecimiento de los tiempos de ejecución de los algoritmos genéticos (sin limitaciones) es exponencial, al igual que el espacio de búsqueda. Esto significa que, al aumentar el tamaño del problema, la calidad de las soluciones puede verse comprometida si se ejecutan dentro de plazos estrictos. Con buenos híper parámetros, como se pueden ver en los experimentos realizados en los trabajos [81, 82] y las secciones 3.1 y 3.2, este crecimiento puede ser casi lineal. Una ventaja adicional de los algoritmos genéticos es que son lo suficientemente flexibles como para considerar atributos de naturaleza muy diferente. Como se puede ver en [82] y la sección 3.2, esta flexibilidad nos ha permitido considerar atributos dependientes del canal, como la latencia y el throughput, de una manera bastante eficiente.

Tanto los algoritmos de un solo objetivo como los multiobjetivo son alternativas comúnmente utilizadas en la literatura. Mientras que un enfoque de un solo objetivo devuelve una sola solución, el enfoque multiobjetivo devuelve un conjunto de soluciones, cada una logrando un compromiso diferente entre los diferentes objetivos. Aunque los algoritmos multiobjetivo requieren un análisis posterior, el enfoque de un solo objetivo se basa en una suma ponderada. La idoneidad de los algoritmos evolutivos de muchos objetivos para el problema de composición ha sido explorada por diferentes autores [75, 84, 96, 100, 104, 112]. Suciu et al. [96] combinan heurísticas adaptativas y el algoritmo multiobjetivo. Moustafa y Zhang [75] utilizan el aprendizaje por refuerzo para lidiar con la incertidumbre característica inherente en entornos abiertos y descentralizados. Trummer et al. [100] analizan formalmente las garantías de complejidad y precisión. Ramírez et al. [84] consideran nueve propiedades de QoS, a saber, tiempo de respuesta, disponibilidad, confiabilidad, throughput, latencia, capacidad de éxito, cumplimiento, mejores prácticas y documentación. Yu, Ma y Zhang [112] proponen un enfoque basado en una estrategia de búsqueda de espacio reducido.

Se han propuesto métodos heurísticos alternativos basados en la noción de *utilidad*. Por ejemplo, Mabrouk et al. [65] proponen un algoritmo que utiliza una heurística tipo utilidad basada en agrupamientos. Calculan la utilidad de los proveedores y luego la agrupan para buscar la composición. Yuan et al. [113] utilizan el método de utilidad con un enfoque de lógica difusa. Su trabajo incluye experimentos con hasta 1000 servicios, obteniendo buenos tiempos de ejecución, lo que muestra la escalabilidad de su propuesta. Sin embargo, aunque estos trabajos utilizan como atributos de calidad el coste, el tiempo de ejecución, la disponibilidad, la precisión y

el throughput, no está claro cómo extender su propuesta a otros atributos como la latencia.

Los enfoques basados en utilidad traen consigo dos problemas principales, el primero de ellos es que, como este es un método basado en heurísticas, al igual que GA, no podemos saber con certeza si hemos alcanzado la mejor solución. El segundo problema es que todos los atributos tenidos en cuenta son dependientes del servicio y del proveedor; no tienen en cuenta atributos que dependen de los canales de comunicación entre los servicios.

En la mayoría de los trabajos donde se abordan la latencia y el throughput (véanse, por ejemplo, [4, 46, 65]), estos atributos se ven como valores numéricos asociados a los proveedores de servicios, que deben minimizarse o maximizarse, de la misma manera que se suelen considerar los atributos de coste, tiempo de respuesta, etc. Klein et al. [60, 61] y Martini et al. [68] tienen en cuenta los canales de comunicación utilizando un modelo de red, lo que les permite calcular los QoS de la red, incluida la latencia y la tasa de transferencia. Klein et al. utilizan modelos de red basados en tablas hash [61] y árboles kd [60]. Aunque consideran varios otros atributos de QoS, la optimización se realiza para los QoS de la red. En su artículo de 2014, Klein et al. [60] logran una latencia casi óptima para sus composiciones de servicios, trabajando bajo condiciones de red realistas.

2.2. Gestión de recursos

La gestión eficiente de recursos en aplicaciones IoT es un área de investigación que está atrayendo interés recientemente, dada la creciente complejidad de estos sistemas. La necesidad de una gestión escalable se ha abordado mediante diversas aproximaciones, donde se destaca el enfoque hacia la inteligencia artificial y la lógica de reescritura. La IA ofrece capacidades predictivas, permitiendo a las aplicaciones anticipar y adaptarse a los cambios en la demanda de recursos. Por otro lado, la lógica de reescritura proporciona una herramienta muy interesante para la gestión dinámica de reglas y políticas de asignación de recursos.

En el amplio espectro de la inteligencia artificial se han utilizado enfoques basados tanto en aprendizaje profundo como en aprendizaje automático para abordar tres clases de problemas de monitorización predictiva en procesos de negocio. Es decir, predicción del siguiente paso (por ejemplo, la próxima tarea que una instancia de proceso probablemente ejecutará), predicción de resultados (por ejemplo, qué tan probable es que un cliente presente una queja dado un estado actual de una instancia de proceso), y rendimiento cuantitativo del proceso (por ejemplo, el tiempo de ejecución restante de cada caso en curso de un proceso).

En esta tesis hemos utilizado la predicción del siguiente paso y la traza para anticipar la disponibilidad y la necesidad de gestionar las réplicas de recursos. Como sugiere la literatura para este tipo de casos [55, 103], empleamos técnicas de aprendizaje profundo (DL); en particular, se utilizan redes (LSTM) [106] para la predicción de trazas.

Se remite al lector a [97] para una comparación interdisciplinar de métodos de modelado de secuencias para predicción del próximo elemento, incluidos modelos

de procesos de negocio y problemas similares en otros dominios. Kratsch et al. [62] comparan el rendimiento de técnicas de DL (es decir, redes neuronales profundas feedforward simples y redes LSTM) y técnicas de ML (es decir, Random Forest y máquinas de vectores de soporte) para la predicción de resultados en procesos de negocio. Además, Teinemaa et al. [98] y Verenich et al. [103] presentan estudios, respectivamente, sobre métodos de predicción orientados a resultados y tiempo restante en la monitorización de procesos de negocio.

Camargo et al. [16] comparan varias arquitecturas de redes neuronales para predecir las actividades, roles y tiempos de las trazas BPMN. En esta tesis se ha implementado una arquitectura similar a la que ellos denominan como "compartida completa", donde la red neuronal recibe información sobre el estado de los recursos utilizados en el proceso BPMN para predecir un próximo estado.

Camargo et al. [15] utilizan arquitecturas compartidas completas para crear redes neuronales generativas basadas en redes recurrentes (GAN-LSTM), junto con conjuntos de datos de trazas BPMN, para crear un simulador capaz de inferir un proceso BPMN completo, incluida la generación de actividades, tiempos de inicio y duración de estas actividades.

Hinkka et al. [54] utilizan un preprocesamiento basado en agrupaciones para simplificar el tamaño de entrada de las redes neuronales, así como para abstraer ciertos atributos. Luego añaden la etiqueta generada por estos grupos como entrada a la red neuronal.

Dado que la monitorización predictiva de procesos ha ganado popularidad en las empresas, Di Francescomarino et al. [48] desarrollaron un marco orientado al valor para clasificar el trabajo existente sobre monitorización predictiva de procesos. Su revisión puede ser utilizada para ayudar a las organizaciones a navegar en el campo de monitorización predictiva de procesos encontrando valor en las oportunidades habilitadas por las técnicas de aprendizaje.

Se han propuesto varias técnicas y enfoques sobre la base de la lógica de reescritura como marco semántico, y se han presentado resultados. En [43] se propone una codificación de procesos de negocio temporizados en el lenguaje Maude [21], que permite la verificación automática de varias propiedades de interés en los procesos, como el tiempo de ejecución máximo/mínimo/promedio. El mismo entorno se utilizó en [38] para analizar el grado de paralelismo temporizado.

En [40] se habla de la especificación estocástica de construcciones temporales y de bifurcación, y la comprobación estadística de los modelos que se soportan en una extensión temporal y probabilística en BPMN. En ese trabajo, los tiempos de duración y las demoras para tareas y flujos pueden especificarse como expresiones estocásticas, mientras que las probabilidades están asociadas a diversas formas de comportamiento de bifurcación en puertas. En este trabajo se utilizó una extensión probabilística de la lógica de reescritura (y Maude) y el verificador de modelos estadísticos PVeStA [3], respectivamente, para la simulación de eventos discretos y la verificación estocástica automática de propiedades. En su análisis se incluyen el tiempo de procesamiento esperado, el tiempo de sincronización esperado en puertas de unión y afirmaciones cuantitativas específicas del dominio.

La ejecución y verificación simbólicas se propusieron en [41] con la ayuda de la reescritura módulo SMT [86]. La ejecución es impulsada por axiomas de reescritura módulo y por consultas a procedimientos de decisión SMT para condiciones de datos. Se pueden especificar y verificar automáticamente propiedades de alcanzabilidad, como la ausencia de bloqueo y la detección de estados no alcanzables con datos que exhiben ciertos valores, con la ayuda de Maude, gracias a su soporte para la reescritura módulo SMT.

Como parte de este esfuerzo, en [37] se propuso un análisis inicial de la asignación de recursos que fue posteriormente extendida en [36]. En este trabajo, se habilitó el cálculo automático de medidas para identificar y optimizar la asignación de recursos en procesos de negocio, incluido el uso de recursos a lo largo del tiempo. Este enfoque tiene la ventaja de capturar todo el comportamiento concurrentes de los procesos y, por lo tanto, se utiliza para simular la evolución concurrente de cualquier proceso comercial con un número dado de recursos y réplicas.

Las simulaciones proporcionan la información esencial para entrenar nuestro modelo LSTM, permitiéndole predecir el comportamiento de la aplicación y anticipar las siguientes tareas a ejecutar. Esta red, complementada con las heurísticas previamente implementadas en [39], ofrecerá una variedad de estrategias para contrastar con la estrategia predictiva desarrollada en el ámbito de esta tesis.

En [35] y la sección 3.3, llevamos a cabo una comparativa de todas las estrategias mencionadas anteriormente, además de la estrategia *predictive-ml-usage*. Esta estrategia utilizará una red neuronal LSTM para predecir la secuencia de actividades BPMN de la aplicación que se ejecutarán en el futuro, permitiendo así una gestión dinámica de los recursos.

2.3. Verificación de la aplicación

En [53], Hewitt, Bishop y Steiger introducen el concepto de *actores* como unidades fundamentales de computación concurrente y establecen las bases teóricas del modelo de actores. Más tarde, en [2], Agha proporciona una revisión exhaustiva del modelo de actores, discutiendo su semántica, características y ventajas sobre otros modelos de concurrencia.

Clarke et al. hablan en [20] de la importancia de la verificación formal de la siguiente forma:

> La verificación formal de programas depende del uso de la lógica matemática. Un programa es un objeto matemático con un comportamiento bien definido, aunque posiblemente complejo e intuitivamente insondable. La lógica matemática puede utilizarse para describir precisamente lo que constituye un comportamiento correcto. Esto hace posible contemplar el establecimiento matemático de que el comportamiento del programa se ajusta a la especificación de corrección.

En la mayoría de los primeros trabajos, la verificación formal de programas implicaba construir una prueba formal de corrección. En contraposición, la verificación

de modelos evita las pruebas. A su vez, en [20] presentan el problema de la explosión de estados en la verificación de modelos. Clarke et al. comentan que existen varias opciones para tratar con este problema en el futuro, entre ellas la verificación de modelos probabilísticos y estadísticos, y la necesidad de escalar este tipo de verificaciones. Hewitt, en [52], destaca el potencial de escalabilidad del modelo de actores gracias a la comunicación mediante mensajes asíncronos. Señala que, con la proliferación de la concurrencia masiva proporcionada por la computación cliente-servidor y las arquitecturas de múltiples núcleos, el interés en el modelo de actores ha crecido aún más.

Existen numerosos programas para verificación de sistemas, que permiten analizar y verificar propiedades de sistemas en diferentes dominios, como sistemas de software, sistemas concurrentes, sistemas embebidos y sistemas en tiempo real, entre otros. Estos programas pueden variar en sus capacidades y enfoques, pero a continuación daremos un repaso por algunos de los más usados.

SPIN [94], además de ser un verificador de modelos, también se puede considerar como una herramienta para verificar sistemas, ya que se utiliza para la verificación de sistemas concurrentes y software concurrente. Permite analizar propiedades de corrección como la ausencia de condiciones de carrera (*race conditions*) y la seguridad en sistemas concurrentes. PRISM [83] es una herramienta de verificación de sistemas probabilísticos que se utiliza para analizar sistemas estocásticos y sistemas con comportamiento probabilístico. Es especialmente útil para la verificación de protocolos de comunicación, sistemas biomédicos y otros sistemas donde las distribuciones de probabilidad sean un factor importante. UPPAAL [10] se utiliza para la verificación de sistemas en tiempo real. Permite analizar propiedades de sistemas en tiempo real y sistemas embebidos, utilizando una combinación de modelos de comportamiento discreto y continuo. Storm [51] es una herramienta para el análisis de sistemas que involucran fenómenos aleatorios o probabilísticos. Dado un modelo de entrada y una especificación cuantitativa, puede determinar si el modelo de entrada se ajusta a la especificación.

En Abhishek et al. [101] se examina la eficiencia de limitar la profundidad de los espacios de estados explorados por los verificadores del modelo. Dado un parámetro k, sus algoritmos garantizan encontrar cualquier violación de una invariante que pueda evidenciarse utilizando un contraejemplo de longitud k o menor desde el estado inicial. De esta forma, exploran formas eficientes de realizar la delimitación de la profundidad, demostrando la solidez de sus algoritmos en el sentido de que exploran exactamente todos los estados alcanzables dentro de un límite de profundidad, y mostrando su eficacia en modelos de productos de Microsoft de gran escala. Más tarde, en el trabajo de Desai et al. [25] se describe el diseño e implementación de P, un lenguaje específico del dominio para modelar sistemas asíncronos dirigidos por eventos. P permite al programador especificar el sistema como una colección de máquinas de estados que interactúan entre sí utilizando eventos. Además, P cuenta con un verificador de seguridad que busca la presencia de eventos no manejados y proporciona comprobaciones de viveza, que detectan cuando un evento puede ser potencialmente diferido indefinidamente. En sus trabajos posteriores [26, 29], Desai et al. añaden un compilador en C para el lenguaje y una herramienta para ejecutar

sistemáticamente programas escritos en P. Según los autores, ambas herramientas permiten analizar en cuestión de minutos ejecuciones que anteriormente podrían tardar meses o incluso años en analizarse en sistemas distribuidos en vivo.

Kwiatkowska et al. [63] presentan técnicas eficientes para la verificación de modelos probabilísticos utilizando una técnica híbrida que combina aspectos de los enfoques simbólicos y explícitos para superar los problemas de rendimiento que sufren los verificadores de modelos comúnmente. En [91], Koushik et al. presentan una herramienta para el análisis estadístico de sistemas probabilísticos llamada VeStA. Esta herramienta admite la verificación estadística de modelos [90, 109], y la evaluación estadística de valores esperados de expresiones temporales. Tomando este trabajo como referencia, Alturki y Meseguer, en [5] tratan de aumentar la escalabilidad de la verificación estadística de modelos y hacer que esta escalabilidad esté disponible para herramientas como Maude, donde los sistemas probabilísticos pueden especificarse a un alto nivel como teorías de reescritura probabilística. Así, en este trabajo presentan la paralelización de VeStA, con una herramienta llamada PVeStA. En su trabajo [44] Jonas et al. presentan un modelo basado en actores compuestos para la especificación de entidades complejas, como Internet. Este modelo ofrece la posibilidad de realizar verificaciones estadísticas mediante un enfoque novedoso que garantiza la ausencia de no determinismo no cuantificado. Para la implementación de este modelo, se emplea el lenguaje Maude junto con la herramienta de verificación de modelos estadísticos PVeStA. Por último, en [89], Sebastio y Vandin se basan en PVeStA para presentar MultiVeStA, una herramienta de análisis estadístico que puede integrarse fácilmente con simuladores de eventos discretos como P [28], enriqueciéndolos con capacidades eficientes de análisis distribuido y paralelizable como una solución para mitigar la explosión de estados producida en la verificación de modelos.

3. Publicaciones

3.1. On the Scalability of Compositions of Service-Oriented Applications

Referencia bibliográfica

[81] Nicolás Pozas and Francisco Durán. "On the scalability of compositions of service-oriented applications". Artículo presentado en International Conference on Service-Oriented Computing (ICSOC 2021). Lecture Notes in Computer Science, vol 13121. Springer, 2021. Anexado en la sección 3.1.

Resumen

One of the current challenges in the context of service-oriented applications is the generation of composition plans for applications that optimize their QoS attributes by taking advantage of the resources offered by different providers, and generate them as efficiently as possible. Scalability is indeed a major issue, and the problem becomes a real challenge for applications with big numbers of services. In this work, we propose a divide-and-conquer algorithm that exploits the architecture of applications, to reduce the size of the search space. With it, we are able to recompose the solution for the global problem, with a significant gain in execution time. A variant of the algorithm—in which, when a sub-problem cannot be further divided without loosing information, a solver is used to find the optimal solution for it—allows us to trade execution time and precision. We report on the extensive experimentation carried out, where applications with up to services are considered, and which includes a comparison with the results delivered by GA solvers.

DOI

https://doi.org/10.1007/978-3-030-91431-8_28

3.2. Location-Aware Scalable Service Composition

Referencia bibliográfica

[82] Nicolás Pozas, Francisco Durán, Katia Moreno Berrocal and Ernesto Pimentel. "Location-Aware Scalable Service Composition". Artículo publicado en Journal of Software: Practice and Experience (2023). Anexado en la sección 3.2

Resumen

The problem of service composition is the process of assigning resources to services from a pool of available ones in the shortest possible time so that the overall quality of service is maximized. This article provides solutions for the composition problem that takes into account its scalability, services' locations, and users' restrictions, which are key for the management of applications using state-of-the-art technologies. The provided solutions use different techniques, including genetic algorithms and heuristics. We provide an extensive experimental evaluation, which shows the pros and cons of each of them, and allows us to characterize the preferred option for each specific problem. Since no solution dominates the others, we propose a decision tree, based on our results, to select the best composition algorithm in each situation.

DOI

https://doi.org/10.1002/spe.3260

3.3. Business Processes Resource Management using Rewriting Logic and Deep-Learning-based Predictive Monitoring

Referencia bibliográfica

[35] Francisco Durán, Nicolás Pozas and Camilo Rocha. "Business Processes Resource Management using Rewriting Logic and Deep-Learning-based Predictive Monitoring". Artículo publicado en Journal of Logical and Algebraic Methods in Programming (2023). Anexado en la sección 3.3

Resumen

A significant task in business process optimization is concerned with streamlining the allocation and sharing of resources. This paper presents an approach for analyzing business process provisioning under a resource prediction strategy based on deep learning. A timed and probabilistic rewrite theory specification formalizes the semantics of business processes. It is integrated with an external oracle in the form of a long short-term memory neural network that can be queried to predict how traces of the process may advance within a time frame. Comparison of execution time and resource occupancy under different parameters is included for several case studies, as well as details on the construction of the deep learning model and its integration with Maude.

DOI

https://doi.org/10.1016/j.jlamp.2023.100928

3.4. Statistical Model Checking for P

Referencia bibliográfica

[33] Francisco Durán, Nicolás Pozas, Carlos Ramírez and Camilo Rocha. "Statistical Model Checking for P". Artículo publicado en Formal Methods for Industrial Critical Systems, FMICS (2023). Lecture Notes in Computer Science, vol 14290. Springer, 2023. Anexado en la sección 3.4.

Resumen

P is a programming language equipped with a unified framework for modeling, specifying, implementing, testing, and verifying complex distributed systems. This language is based on the actor model, and its framework includes a compiler toolchain for code generation, bounded randomized testing, and reachability analysis. This paper presents an extension of P 's framework to include statistical model checking of quantititive temporal logic formulas. The distributed statistical model checking for discrete event simulators has been integrated into the P framework for Monte Carlo validation of P programs against QuaTEx quantitative temporal logic formulas. For this, P 's compiler has been modified to generate instrumentation code enabling the observation of a program's attributes without direct, manual intervention of the code. As a result, distributed incremental statistical model checking is now available for P via probabilistic sampling. As the experiments show, some of them reported

here, these new verification capabilities scale up and complement the ones already available for P.

DOI
https://doi.org/10.1007/978-3-031-43681-9_3

4. Conclusiones y trabajos futuros

Durante el desarrollo de esta tesis hemos profundizado en la complejidad referente a la gestión de servicios IoT. A través de un análisis detallado que se enfoca en tres objetivos principales, la composición de servicios, la predicción de servicios y la verificación de aplicaciones, hemos evaluado cómo podemos dar solución de manera efectiva a cada uno de estos apartados, maximizando la eficiencia y minimizando la carga de trabajo del sistema.

Hemos evaluado una serie de algoritmos de composición de servicios, en concreto nos hemos centrado en los algoritmos genéticos, algoritmos de utilidad y un algoritmo llamado Express, analizando las fortalezas y debilidades de cada uno de ellos. El resultado de esta evaluación ha dado lugar al árbol de decisión representando en la figura 4.1. En este árbol de decisión se pueden observar todos los métodos implementados: *U*, *UM*, *GA* y *Express* (E), cada uno de ellos con sus ventajas y desventajas. Al analizar el árbol de decisión, podemos inferir que si nuestro principal objetivo es obtener la composición con mejor calidad posible sin que el tiempo de ejecución sea relevante, nuestra mejor opción sería hacer uso de *GA* sin establecer un límite de tiempo y utilizando los parámetros adecuados. Por otro lado, si nuestro objetivo es tener una composición en el menor tiempo posible y contamos con una arquitectura de aplicación y proveedores fijos, entonces una buena opción sería el uso de *UM* o *U*. En este caso, una vez calculada la utilidad de cada proveedor, podríamos intercambiarlos rápidamente en caso de fallos. Sin embargo, si nuestra aplicación hace uso de restricciones que debemos cumplir de manera estricta, sería apropiado emplear el algoritmo *GA* con un límite de tiempo para obtener una solución adecuada en un tiempo razonable. Finalmente, si no tenemos restricciones o podemos ser más flexibles con ellas, el algoritmo *E* sería nuestra opción recomendada, ya que cada servicio es considerado de forma aislada y se asigna el mejor proveedor para cada servicio sin tener en cuenta la arquitectura de la aplicación ni las restricciones establecidas. Para tener una visión global más informada, nos gustaría también ejecutar los diferentes algoritmos en procesos reales de la industria. Hemos estado buscando en repositorios de servicios para realizar estas pruebas, pero la mayoría de estos repositorios corresponden a procesos con poca cantidad de servicios y proveedores.

Por otro lado, hemos hecho uso de las predicciones en trazas BPMN para la planificación de estrategias basadas en recursos. Al anticipar la demanda de recursos, podemos asignar de manera pro activa los recursos necesarios, evitando así costes innecesarios por exceso de recursos disponibles o escasez de recursos, lo que podría afectar negativamente a la calidad del servicio, aumentar los costes operativos y provocar retrasos innecesarios. Utilizando la predicción proporcionada por la red neuronal descrita en la figura 1.10 de la sección 1.6.2, hemos implementado la estrategia *predictive-ml-usage*, que ha sido comparada con las estrategias ya implementadas en el trabajo de referencia [39].

La figura 4.2 presenta una comparativa entre las diferentes estrategias aplicadas a un caso de uso denominado “visa”, tal como se describe en el mismo trabajo. Estas estrategias son:

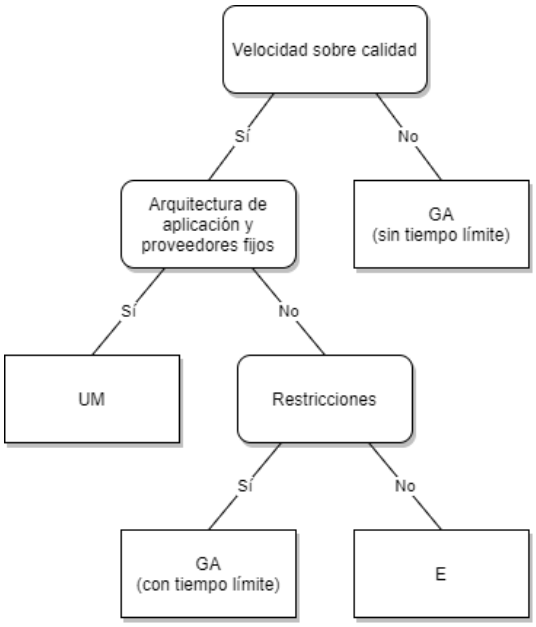


Figura 4.1: Árbol de decisión para elegir algoritmos de composición de servicios según la presencia de restricciones, disponibilidad de arquitectura, tipo de arquitectura (fija o variable) y existencia de un límite de tiempo

- *predictive-ml-usage*: Esta estrategia utiliza la información de la traza actual de las actividades ejecutadas para alimentar una red neuronal. Esta red neuronal predice las siguientes actividades que se van a ejecutar y el estado futuro del sistemas. Con esta información, podemos gestionar los recursos de acuerdo con la situación prevista del sistema.
- *predictive-usage*: Similar a la estrategia anterior, pero en lugar de una red neuronal, hace uso de la semántica probabilística del proceso de negocio para predecir las siguientes actividades a ejecutar.
- *queues*: Esta estrategia se basa en la información recopilada de una ventana de tiempo pasado y considera el tamaño de las colas de peticiones de recursos para gestionarlos.
- *usage*: Similar a la estrategia anterior, pero en lugar de considerar el tamaño de las colas, tiene en cuenta el porcentaje de uso de los recursos.
- *usage-queues*: Esta estrategia combina elementos de las dos estrategias anteriores.

El propósito de esta figura no es entrar en detalles sobre cada una de las columnas y barras mostradas, sino observar el comportamiento de las diferentes estrategias conforme se modifican tres parámetros principalmente:

- **Time Between Checks** (TBC): Es el tiempo entre comprobaciones con la que se realiza la comprobación del sistema. Este parámetro es común a todas las estrategias presentadas.

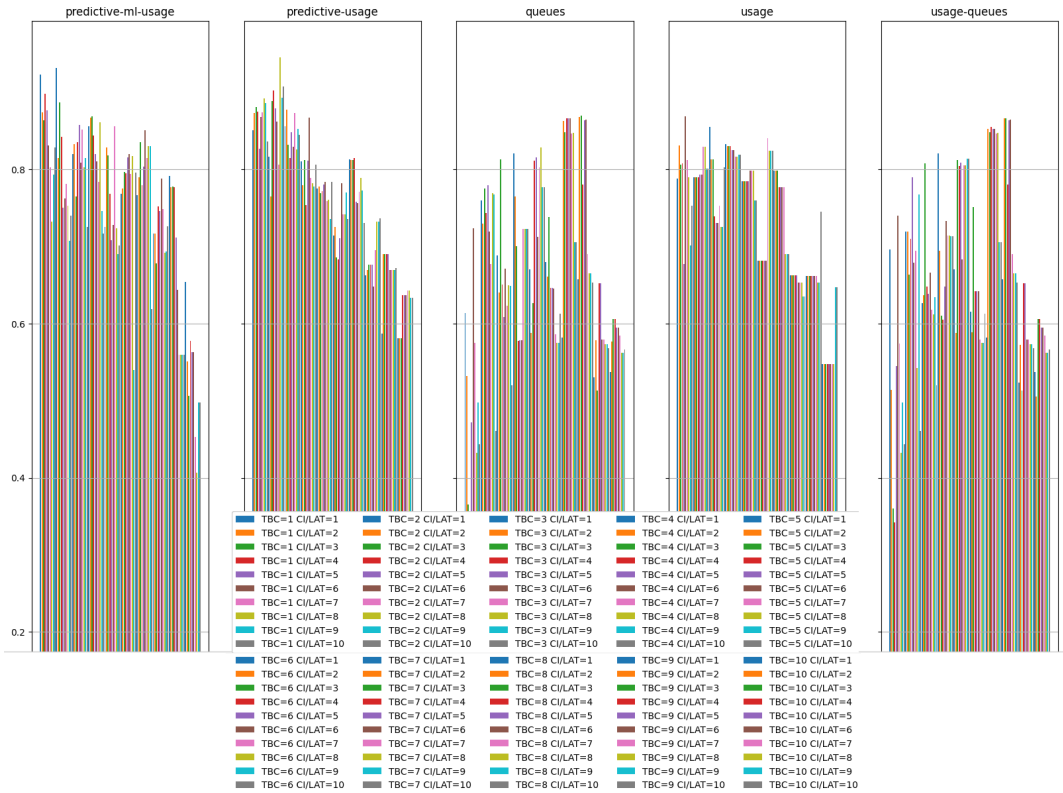


Figura 4.2: Comparación de estrategias

- **Check Interval** (CI): Representa la ventana de tiempo considerada para calcular las medias, es decir, el tamaño del historial tenido en cuenta. Este parámetro es usado en estrategias que miran al pasado, como *queues*, *usages* y *queues-usage*.

- **Look-Ahead Time** (LAT): Indica el tamaño de la ventana de tiempo en que queremos predecir eventos y se utiliza en estrategias predictivas como *predictive-ml-usage* y *predictive-usage*.

La distribución de estos parámetros en el eje x se realiza de menor a mayor, comenzando desde la izquierda, y de 1 al 10. Por ejemplo, la columna 13 tendría los parámetros *TBC=2, CI/LAT=3*, mientras que la columna 25 tendría los parámetros *TBC=3, CI/LAT=5*. El eje y representa el valor de la función de fitness, en una escala entre 0 y 1, donde 0 representa el peor resultado posible y 1 el mejor resultado posible. Observando la figura 4.2 vemos que las estrategias predictivas tienen en general un mejor resultado que las estrategias basadas en *queues*. Además, estas estrategias muestran un mejor comportamiento cuando la ventana de tiempo a predecir (LAT) es menor (1-3 unidades de tiempo) y el tiempo entre comprobaciones (TBC) tiene un valor medio (5 unidades de tiempo).

En la tercera parte de esta tesis, hemos explorado cómo la verificación de aplicaciones mediante el model checking estadístico puede ofrecer una capa adicional de garantía en la calidad de nuestras aplicaciones. Esta metodología nos permite

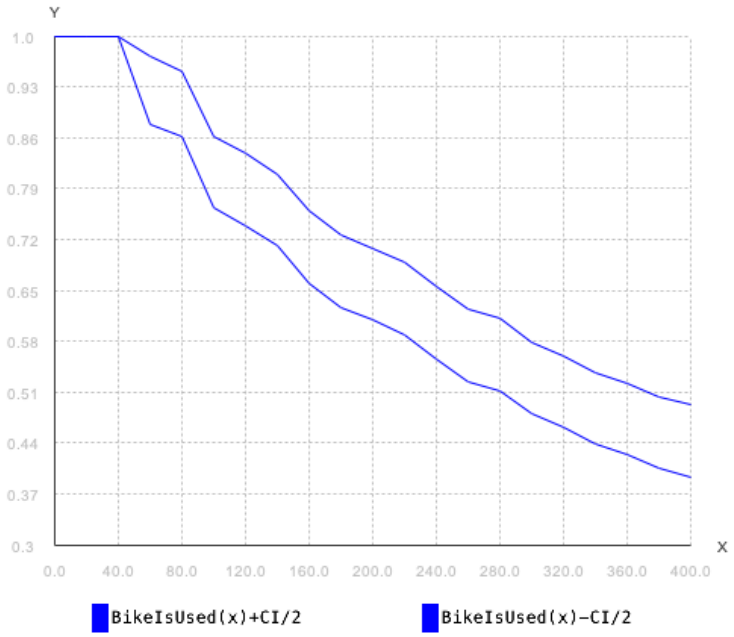


Figura 4.3: Resultados de la consulta MultiQuaTEx

identificar posibles fallos o anomalías en el comportamiento de las aplicaciones IoT, lo que contribuye significativamente a la fiabilidad y seguridad del sistema en general.

La utilización de model checking estadístico nos permite realizar un análisis cuantitativo de los sistemas especificados. Además, este tipo de análisis supone dos mejoras considerables en cuanto a la escalabilidad y la eficiencia del análisis. En primer lugar, permite obtener una respuesta aproximada especificando un tiempo límite de análisis o un margen de error mínimo, de forma que al alcanzar este valor el análisis termina dando el valor calculado hasta ese momento como resultado. Por otra parte, las simulaciones pueden ejecutarse concurrentemente. Hay pues varios parámetros con los que podemos adaptarnos a estas necesidades específicas: modificando el margen de error aceptado, modificando el tamaño de los bloques de simulaciones, el número de procesadores y la potencia de los mismos.

El resultado de la consulta MultiQuaTEx utilizada para llevar a cabo el experimento anterior se muestra en la figura 4.3, cuya pregunta subyacente es "¿Qué probabilidad existe de que una bicicleta sea utilizada más de x veces antes de romperse definitivamente?". En esta figura, las probabilidades se representan en el eje y, mientras que el número de usos se encuentran en el eje x. En la figura 4.3 se observan dos líneas que representan un intervalo de confianza para la solución del problema. De este modo, podemos observar que la probabilidad de que una bicicleta sea usada más de 40 veces antes de romperse es prácticamente del 100 %, mientras que para 200 veces este valor oscila aproximadamente entre un 72 % y un 60 %.

En cuanto al rendimiento y a la escalabilidad de la consulta, que es otro de los factores a tener en cuenta, el resultado puede verse en la figura 4.4. En esta figura se muestran los resultados de un experimento realizado haciendo uso de diferentes

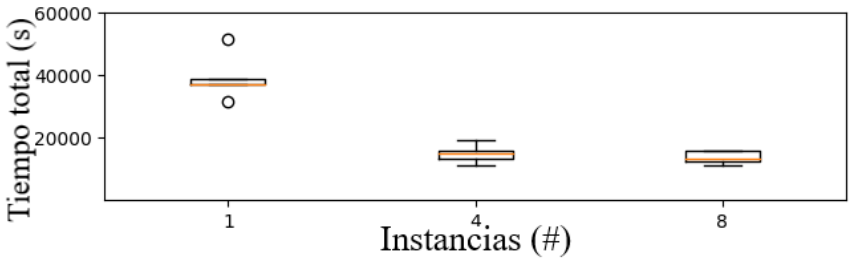


Figura 4.4: Rendimiento en instancias t2.small de AWS

instancias de AWS para evaluar su comportamiento en cuanto a escalabilidad usando la misma consulta "¿Qué probabilidad existe de que una bicicleta sea utilizada más de x veces antes de romperse definitivamente?". En el eje x de esta figura, se muestra el número de instancias *t2.small*[2] en las que se ha lanzado las simulaciones. En el eje y se representa el tiempo total, en segundos, necesario para completar las ejecuciones. Con una instancia, el tiempo total de ejecución es de 40000 segundos, para cuatro instancias es ligeramente inferior a 20000 segundos, y para ocho instancias el tiempo es aproximadamente igual que para cuatro instancias. Esto se debe a que, en este problema en particular, el número de simulaciones requeridas es bajo. Con el tamaño de lote fijado, se observa una notable mejora de rendimiento al pasar a cuatro instancias, pero agregar más de cuatro instancias no resulta beneficioso. Sin embargo, en otros experimentos, la situación podría ser distinta.

La arquitectura utilizada para nuestras simulaciones ya ha sido descrita en sección 1.6.3. Como se menciona en dicha sección, al ejecutar la herramienta MultiVeStA, lanzamos instancias individuales del simulador P. Para distribuir nuestras simulaciones hemos aprovechado este concepto y la facilidad que ofrece MultiVeStA, para hacerlo de manera sencilla.

La consulta mencionada se ejecuta en lotes de tamaño 20 con una probabilidad objetivo $\alpha = 0.05$ y un intervalo de confianza $\delta = 0.1$. Esto significa que cuando se alcance un valor medio $\bar{x}$ mediante la simulación, con una probabilidad $1-\alpha$ y esta se encuentre dentro del rango de confianza $[\bar{x}-\frac{\delta}{2}, \bar{x}+\frac{\delta}{2}]$, la simulación finalizará. Antes de comenzar, se decide el número de servidores que vamos a usar (1, 4 u 8), ejecutamos el cliente MultiVeStA con la consulta e informamos al cliente de las direcciones IP de cada uno de los servidores. Cada servidor cuenta con una instancia de MultiVeStA que se encuentra escuchando las peticiones. Al recibir la petición del cliente, cada uno de los servidores utilizados ejecutará 20 simulaciones de la consulta.

Estos servidores lanzarán sus respectivos simuladores P, ejecutarán las 20 simulaciones y mandarán de devuelta los resultados al cliente MultiVeStA. Cuando el cliente reciba todas y cada una de estas respuestas, evaluará si la solución se

[2]Las instancias *t2.small*, son un tipo de instancias de AWS de uso general. Estas instancias cuentan cada una con una CPU virtual (como las denomina AWS) y 2GiB de memoria RAM. Estas instancias se encuentran dentro de los productos *Amazon Elastic Compute Cloud* (EC2) que ofrece la plataforma.

encuentra dentro de los márgenes de error indicados (α y δ). En caso afirmativo, la simulación termina, proporcionando como resultado el valor medio correspondiente con su intervalo de confianza como se muestra en la figura 4.3. En caso contrario, el cliente volverá a lanzar otro lote de 20 simulaciones a cada uno de los servidores hasta que los márgenes de error estén dentro de los valores deseados o hasta que se alcance el número máximo de simulaciones, lo que ocurra primero.

En conjunto, estas investigaciones proporcionan un buen enfoque para abordar los desafíos de escalabilidad en la gestión de servicios IoT, ofreciendo una base de conocimientos sólida para el diseño e implementación de sistemas robustos y eficientes que puedan adaptarse dinámicamente a las demandas del entorno. Este trabajo representa un paso significativo en el campo de la gestión de servicios IoT, allanando el camino para futuras innovaciones y avances en este área crítica.

La amplitud de los temas tratados en esta tesis ha dejado muchos problemas por resolver. Por ejemplo, en cuanto a la composición de servicios, además de los atributos de calidad de servicio considerados, se podrían considerar otros atributos como la sostenibilidad, el impacto medioambiental o la eficiencia energética. Además, sería interesante explorar el uso algoritmos multiobjetivo, que nos permitan escoger entre distintas estrategias basándonos en diferentes criterios, como la minimización del coste, la eco-eficiencia, entre otros. Se podrían considerar modelos de redes para tratar con mayor precisión los canales de comunicación. En cuanto a la optimización de la asignación de recursos, una primera línea de investigación consistiría en considerar un mejor modelo de los recursos. En el trabajo realizado se considera un recurso con un número de instancias disponibles y en uso. El número de opciones es enorme, se podrían considerar desde recursos consumibles como la gasolina o el dinero, hasta elementos no cuantificables o con distintas unidades de medida, como la harina o la cantidad de personas. Además, se podría considerar la disponibilidad de los recursos, como el horario de los trabajadores o tiempo necesario para fabricar una pieza, así como todo tipo de restricciones operativas, como las capacidades de carga de un repartidor o las rutas de distribución disponibles.

Asimismo, también se pueden mejorar los modelos y su entrenamiento, así como aumentar la eficiencia de este proceso. En cuanto a la verificación, se pueden explorar métodos automatizados para generar los parámetros del análisis automáticamente y hacer uso de otras herramientas de verificación, por ejemplo, podríamos utilizar una semántica de P (o de otros lenguajes de modelado de aplicaciones) para explorar otras formas de verificar aplicaciones de gran tamaño. Por ejemplo, podría ser útil emplear una semántica P de lógica de reescritura para utilizar abstracciones de los modelos y realizar verificaciones, o bien utilizar análisis simbólico.

En cuanto a la optimización de procesos de negocio, se podría mejorar la optimización de la asignación estática y dinámica de los recursos de distintas formas, pero también se podría abordar la mejora de la estructura misma de los procesos. Para ello, existen distintas técnicas de aprendizaje computacional que podrían ayudarnos. Una idea que nos gustaría explorar consiste en construir modelos que nos sugieran operaciones de refactorización de nuestros procesos de negocio. Esto significa que, al identificar las operaciones de refactorización de procesos de negocio, podemos plantear el problema de optimización como un proceso de exploración. Aunque la

optimización en la asignación de recursos se plantea como una línea independiente, ambas tienen un aspecto en común: proporcionar un mecanismo de análisis que nos permite establecer una relación de orden sobre los procesos de negocio con sus recursos asignados. En otras palabras, podemos explorar el espacio de búsqueda proporcionado por las operaciones de refactorización para encontrar la estructura óptima para el proceso.

Dado el enorme tamaño del espacio de búsqueda, una exploración exhaustiva se anticipa computacionalmente impracticable. Para acelerar la convergencia, proponemos guiar la búsqueda utilizando meta-heurísticas, específicamente, alguna variante de la escalada de colinas (*hill climbing*) y la búsqueda tabú. Este procedimiento nos permitirá analizar no solo factores como el coste y el tiempo de ejecución, sino también otros aspectos como el consumo y el desperdicio de energía, que son claves para la preocupación actual por los problemas climáticos.

Todas las ideas planteadas hasta ahora son sin embargo pequeños incrementos en líneas de investigación que ya estamos explorando. Sin embargo, la razón principal detrás de esta propuesta es otra: explorar el potencial de los algoritmos de Machine Learning y las redes neuronales orientadas a grafos (*Graph Neural Networks*, GNN) para la optimización. De hecho, creemos que contamos con todos los componentes necesarios: uno de los principales obstáculos al implementar técnicas de aprendizaje y modelado basados en redes neuronales es la necesidad de las grandes cantidades de datos necesarios para entrenar el modelo. Sin embargo, el disponer de técnicas alternativas, aunque ineficientes, nos permite considerar la posibilidad de generar casos artificiales que pseudo-optimizamos utilizando heurísticos. Esto nos proporciona un número ilimitado de casos con los que entrenar nuestros modelos.

Uno de los principales problemas de los modelos generativos es su propensión a "inventar" soluciones que pueden no ser válidas ni mejores que la solución inicial. Pedirle al modelo generativo una versión mejorada del proceso podría dar como resultado un proceso que no ha mejorado, pero incluso podría dar como resultado un proceso que no sea válido. Sin embargo, la combinación de GNNs y operaciones de transformación puede mejorar esta situación.

La propuesta consiste en utilizar como entrada del modelo un grafo que represente el proceso de negocio en su totalidad, recogiendo la información estructural del proceso, la asignación de recursos y otros parámetros relevantes (como recursos disponibles y cargas de trabajo). El objetivo es que el modelo GNN produzca como salida la operación de refactorización necesaria para mejorar el proceso. Al alimentar recurrentemente la red neuronal con el proceso modificado que estamos obteniendo como salida, acabaremos alcanzando un proceso no mejorable, que devolveremos como óptimo.

Además, la información recopilada a partir de los diferentes análisis descritos anteriormente, nos permitirá comprender mejor el funcionamiento general del proceso de negocio. El siguiente paso, implicará utilizar dicha información para resolver o reducir problemas identificados. Investigaremos la identificación de "olores" (como cuellos de botella, retrasos en la sincronización y uso excesivo de recursos) para sugerir acciones de refactorización del modelo.

Una vez desarrollada esta primera parte, creemos que el uso de modelos generativos puede ser de gran utilidad para proporcionar al responsable del modelado del proceso, una explicación clara y concisa del motivo detrás de las acciones de refactorización propuestas. Esto facilitará al responsable la comprensión de cada uno de los pasos de refactorización dados, verificar su coherencia y validez, y finalmente aceptando los cambios propuestos con confianza.

Referencias

[1] Gustav Aagesen y John Krogstie. «BPMN 2.0 for Modeling Business Processes». En: *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems* (abr. de 2015), págs. 219-250. DOI: 10.1007/978-3-642-45100-3_10.

[2] Gul Agha. «Actors: a Model of Concurrent Computation in Distributed Systems (Parallel Processing, Semantics, Open, Programming Languages, Artificial Intelligence)». Tesis doct. University of Michigan, USA, 1985. URL: http://hdl.handle.net/2027.42/160629.

[3] Gul Agha, José Meseguer y Koushik Sen. «PMaude: Rewrite-based Specification Language for Probabilistic Object Systems». En: *Electronic Notes in Theoretical Computer Science* 153.2 (2006). Proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005), págs. 213-239.

[4] Mohammad Alrifai y Thomas Risse. «Combining Global Optimization with Local Selection for Efficient QoS-Aware Service Composition». En: *Intl. Conf. on World Wide Web (WWW)*. 2009, págs. 881-890. DOI: 10.1145/1526709.1526828.

[5] Musab AlTurki y José Meseguer. «PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool». En: *Proc. of CALCO*. Vol. 6859. LNCS. Springer, 2011, págs. 386-392.

[6] Musab AlTurki, José Meseguer y Carl A. Gunter. «Probabilistic Modeling and Analysis of DoS Protection for the ASV Protocol». En: *SecReT@LICS/CSF 2008*. Vol. 234. Electronic Notes in Theoretical Computer Science. Elsevier, 2008, págs. 3-18. URL: https://doi.org/10.1016/j.entcs.2009.02.069.

[7] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Systinet Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guiar, Neelakantan Kartha et al. *WS-BPEL: Web Services Business Process Execution Language Version 2.0*. Standard. OASIS, abr. de 2007.

[8] Damian Arellanes y Kung-Kiu Lau. «Evaluating IoT service composition mechanisms for the scalability of IoT systems». En: *Future Gener. Comput. Syst.* 108 (2020), págs. 827-848. DOI: 10.1016/j.future.2020.02.073.

[9] Maurice H. ter Beek, Axel Legay, Alberto Lluch-Lafuente y Andrea Vandin. «Statistical analysis of probabilistic models of software product lines with quantitative constraints». En: *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*. ACM, 2015, págs. 11-15. DOI: 10.1145/2791060.2791087.

[10] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson y Wang Yi. «UPPAAL — a tool suite for automatic verification of real-time systems». En: *Hybrid Systems III*. Ed. por Rajeev Alur, Thomas A. Henzinger y Eduardo D. Sontag. Berlin, Heidelberg: Springer, 1996, págs. 232-243.

[11] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann y Ralf Steinmetz. «Heuristics for QoS-aware Web Service Composition». En: *IEEE Intl. Conf. on Web Services (ICWS)*. 2006, págs. 72-82. DOI: 10.1109/ICWS.2006.69.

[12] *Business Process Model and Notation (BPMN) - V 2.0*. Jan 2011. Object Manag. Group.

[13] Jesús Alejandro Cárdenes Cabré, Doina Precup y Ricardo Sanz. «Horizontal and Vertical Self-Adaptive Cloud Controller with Reward Optimization for Resource Allocation». En: *Cloud and Autonomic Computing (ICCAC)*. 2017, págs. 184-185. DOI: 10.1109/ICCAC.2017.25.

[14] Radu Calinescu, Lars Grunske, Marta Z. Kwiatkowska y Raffaela. «Dynamic QoS Management and Optimization in Service-Based Systems». En: *IEEE Trans. Software Eng.* 37.3 (2011), págs. 387-409. DOI: 10.1109/TSE.2010.92.

[15] Manuel Camargo, Marlon Dumas y Oscar González-Rojas. «Learning Accurate Business Process Simulation Models from Event Logs via Automated Process Discovery and Deep Learning». En: *Advanced Information Systems Engineering*. Ed. por Xavier Franch, Geert Poels, Frederik Gailly y Monique Snoeck. Springer, 2022, págs. 55-71.

[16] Manuel Camargo, Marlon Dumas y Oscar González-Rojas. «Learning Accurate LSTM Models of Business Processes». En: *Business Process Management*. Ed. por Thomas Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger y Jan Mendling. Springer, 2019, págs. 286-302.

[17] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito y Maria Luisa Villani. «An Approach for QoS-Aware Service Composition Based on Genetic Algorithms». En: *Conf. on Genetic and Evolutionary Computation (GECCO)*. ACM, 2005, págs. 1069-1075. DOI: 10.1145/1068009.1068189.

[18] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold y Krys Kochut. «Quality of service for workflows and web service processes». En: *J. of Web Semantics* 1.3 (2004), págs. 281-308. DOI: https://doi.org/10.1016/j.websem.2004.03.001.

[19] Ing-Ray Chen, Jia Guo y Fenye Bao. «Trust Management for SOA-Based IoT and Its Application to Service Composition». En: *IEEE Trans. Serv. Comput.* 9.3 (2016), págs. 482-495. DOI: 10.1109/TSC.2014.2365797.

[20] Edmund M. Clarke, E. Allen Emerson y Joseph Sifakis. «Model checking: algorithmic verification and debugging». En: 52.11 (nov. de 2009), págs. 74-84. DOI: 10.1145/1592761.1592781.

[21] Manuel Clavel, Francisco Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer y C.L. Talcott. *All About Maude*. Vol. 4350. LNCS. Springer, 2007.

[22] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer y José F. Quesada. «Maude: specification and programming in rewriting logic». En: *Theoretical Computer Science* 285.2 (2002). Rewriting Logic and its Applications, págs. 187-243. DOI: https://doi.org/10.1016/S0304-3975(01)00359-0.

[23] Emanuel Ferreira Coutinho, Flávio Rubens Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes y José Neuman Souza. «Elasticity in cloud computing: a survey». En: *Ann. des Télécommunications* 70.7-8 (2015), págs. 289-309. DOI: 10.1007/s12243-014-0450-7.

[24] Michael R. Crusoe Michael R. Crusoe, Sanne Abeln, Alexandru Iosup, Peter Amstutz, John Chilton, Nebojša Tijanić, Hervé Ménager, Stian Soiland-Reyes, Bogdan Gavrilović, Carole Goble y The CWL Community. «Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language». En: *Commun. ACM* 65.6 (2022), págs. 54-63. DOI: 10.1145/3486897.

[25] Ankush Desai, Vivek Gupta, Ethan Jackson, Shaz Qadeer, S. Rajamani y D. Zufferey. «P: safe asynchronous event-driven programming». En: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '13. Seattle, Washington, USA: ACM, 2013, págs. 321-332. DOI: 10.1145/2491956.2462184.

[26] Ankush Desai, Ethan Jackson, Amar Phanishayee, Shaz Qadeer y Sanjit A. Seshia. *Building Reliable Distributed Systems With P*. Inf. téc. UCB/EECS-2015-198. EECS Department, University of California, Berkeley, sep. de 2015. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-198.html.

[27] Ankush Desai, Amar Phanishayee, Shaz Qadeer y Sanjit A. Seshia. «Compositional programming and testing of dynamic distributed systems». En: *Proc. ACM Program. Lang.* 2.OOPSLA (2018), 159:1-159:30. DOI: 10.1145/3276529.

[28] Ankush Desai y Shaz Qadeer. «P: Modular and Safe Asynchronous Programming». En: *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*. Ed. por Shuvendu K. Lahiri y Giles Reger. Vol. 10548. Lecture Notes in Computer Science. Springer, 2017, págs. 3-7. DOI: 10.1007/978-3-319-67531-2_1.

[29] Ankush Desai, Shaz Qadeer y Sanjit A. Seshia. «Systematic testing of asynchronous reactive systems». En: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: ACM, 2015, págs. 73-83. DOI: 10.1145/2786805.2786861.

[30] Niranjana Deshpande y Naveen Sharma. «Composition Algorithm Adaptation in Service Oriented Systems». En: *Software Architecture*. Springer, 2020, págs. 170-179.

[31] P Developers. *Página web de P*. https://p-org.github.io/P/.

[32] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, Giulio Petrucci y Anton Yeshchenko. «An eye into the future: leveraging a-priori knowledge in predictive business process monitoring». En: *International Conference on Business Process Management*. Springer. 2017, págs. 252-268.

[33] Francisco Durán, Nicolás Pozas, Carlos Ramírez y Camilo Rocha. «Statistical Model Checking for P». En: *Formal Methods for Industrial Critical Systems*. Ed. por Alessandro Cimatti y Laura Titolo. Cham: Springer, 2023, págs. 40-56.

[34] Francisco Durán, Nicolás Pozas y Camilo Rocha. *BPMN Resource Manager*. URL: https://github.com/Pozas91/bpmn-resource-manager.

[35] Francisco Durán, Nicolás Pozas y Camilo Rocha. «Business processes resource management using rewriting logic and deep-learning-based predictive monitoring». En: *Journal of Logical and Algebraic Methods in Programming* 136 (2024), pág. 100928. DOI: https://doi.org/10.1016/j.jlamp.2023.100928.

[36] Francisco Durán, Camilo Rocha y Gwen Salaün. «A rewriting logic approach to resource allocation analysis in business process models». En: *Sci. Comput. Program.* 183 (2019). DOI: 10.1016/j.scico.2019.102303.

[37] Francisco Durán, Camilo Rocha y Gwen Salaün. «Analysis of Resource Allocation of BPMN Processes». En: *17th International Conference on Service-Oriented Computing -ICSOC 2019*. Vol. 11895. Lecture Notes in Computer Science. Springer, 2019, págs. 452-457. DOI: 10.1007/978-3-030-33702-5_35.

[38] Francisco Durán, Camilo Rocha y Gwen Salaün. «Computing the Parallelism Degree of Timed BPMN Processes». En: *Software Technologies: Applications and Foundations - STAF 2018*. Vol. 11176. Lecture Notes in Computer Science. Springer, 2018, págs. 320-335. DOI: 10.1007/978-3-030-04771-9_24.

[39] Francisco Durán, Camilo Rocha y Gwen Salaün. «Resource provisioning strategies for BPMN processes: Specification and analysis using Maude». En: *J. Log. Algebraic Methods Program.* 123 (2021), pág. 100711. DOI: 10.1016/j.jlamp.2021.100711.

[40] Francisco Durán, Camilo Rocha y Gwen Salaün. «Stochastic Analysis of BPMN with Time in Rewriting Logic». En: *Science of Computer Programming* 168 (ago. de 2018). DOI: 10.1016/j.scico.2018.08.007.

[41] Francisco Durán, Camilo Rocha y Gwen Salaün. «Symbolic Specification and Verification of Data-Aware BPMN Processes Using Rewriting Modulo SMT». En: *12th International Workshop on Rewriting Logic and Its Applications - WRLA 2018*. Vol. 11152. Lecture Notes in Computer Science. Springer, 2018, págs. 76-97. DOI: 10.1007/978-3-319-99840-4_5.

[42] Francisco Durán y Gwen Salaün. «Optimization of BPMN Processes via Automated Refactoring». En: *Service-Oriented Computing - 20th International Conference, ICSOC 2022, Seville, Spain, November 29 - December 2, 2022, Proceedings*. Ed. por Javier Troya, Brahim Medjahed, Mario Piattini, Lina Yao, Pablo Fernández y Antonio Ruiz-Cortés. Vol. 13740. Lecture Notes in Computer Science. Springer, 2022, págs. 3-18. DOI: 10.1007/978-3-031-20984-0_1.

[43] Francisco Durán y Gwen Salaün. «Verifying Timed BPMN Processes using Maude». En: *Proc. of COORDINATION*. Vol. 10319. LNCS. Springer, 2017, págs. 219-236.

[44] Jonas Eckhardt, Tobias Mühlbauer, José Meseguer y Martin Wirsing. «Statistical Model Checking for Composite Actor Systems». En: *WADT 2012*. Vol. 7841. Lecture Notes in Computer Science. Springer, 2012, págs. 143-160. URL: https://doi.org/10.1007/978-3-642-37635-1_9.

[45] Shi-Liang Fan, Feng Ding, Chenghao Guo y Yu-Bin Yang. «Supervised Web Service Composition Integrating Multi-objective QoS Optimization and Service Quantity Minimization». En: *25th International Conference on Web Services (ICWS)*. Vol. 10966. Lecture Notes in Computer Science. Springer, 2018, págs. 215-230. DOI: 10.1007/978-3-319-94289-6_14.

[46] Shi-Liang Fan, Kai-Yu Peng y Yu-Bin Yang. «Large-Scale QoS-Aware Service Composition Integrating Chained Dynamic Programming and Hybrid Pruning». En: *25th International Conference on Web Services (ICWS)*. Vol. 10966. Lecture Notes in Computer Science. Springer, 2018, págs. 196-211. DOI: 10.1007/978-3-319-94289-6_13.

[47] Paola Festa y Mauricio G.C. Resende. «Grasp: An Annotated Bibliography». En: *Essays and Surveys in Metaheuristics*. Springer, 2002, págs. 325-367. DOI: 10.1007/978-1-4615-1507-4_15.

[48] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi y Fredrik Milani. «Predictive Process Monitoring Methods: Which One Suits Me Best?» En: *Business Process Management - 16th International Conference, BPM, Proceedings*. Ed. por Mathias Weske, Marco Montali, Ingo Weber y Jan vom Brocke. Vol. 11080. Lecture Notes in Computer Science. Springer, 2018, págs. 462-479. DOI: 10.1007/978-3-319-98648-7_27.

[49] Hubert Garavel, Frédéric Lang, Radu Mateescu y Wendelin Serwe. «CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes». En: *International Journal on Software Tools for Technology Transfer* 15 (mar. de 2013), págs. 89-107. DOI: 10.1007/s10009-012-0244-z.

[50] Daniel Garijo, Yolanda Gil y Oscar Corcho. «Abstract, link, publish, exploit: An end to end framework for workflow sharing». En: *Future Generation Computer Systems* (2017). DOI: 10.1016/j.future.2017.01.008.

[51] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann y Matthias Volk. «The Probabilistic Model Checker Storm». En: *CoRR* abs/2002.07080 (2020). arXiv: 2002.07080. URL: https://arxiv.org/abs/2002.07080.

[52] Carl Hewitt. «Actor Model of Computation: Scalable Robust Information Systems». En: arXiv (ago. de 2010).

[53] Carl Hewitt, Peter Bishop y Richard Steiger. «A Universal Modular ACTOR Formalism for Artificial Intelligence». En: *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*. 1973, págs. 235-245. URL: http://ijcai.org/Proceedings/73/Papers/027B.pdf.

[54] Markku Hinkka, Teemu Lehto y Keijo Heljanko. «Exploiting Event Log Event Attributes in RNN Based Prediction». En: *New Trends in Databases and Information Systems*. Ed. por Tatjana Welzer, Johann Eder, Vili Podgorelec, Robert Wrembel, Mirjana Ivanović, Johann Gamper, Mikołaj Morzy, Theodoros Tzouramanis, Jérôme Darmont y Aida Kamišalić Latifić. Springer, 2019, págs. 405-416.

[55] Markku Hinkka, Teemu Lehto, Keijo Heljanko y Alexander Jung. «Classifying Process Instances Using Recurrent Neural Networks». En: *Business Process Management Workshops*. Ed. por Florian Daniel, Quan Z. Sheng y Hamid Motahari. Springer, 2019, págs. 313-324.

[56] ISO/IEC. «International Standard 19510, Information technology – Business Process Model and Notation». En: 2013.

[57] Sourabh Katoch, Sumit Singh Chauhan y Vijay Kumar. «A review on genetic algorithm: past, present, and future». En: *Multim. Tools Appl.* 80.5 (2021), págs. 8091-8126. DOI: 10.1007/s11042-020-10139-6.

[58] *Keras*. 2015. URL: https://keras.io.

[59] Ahmed Kheldoun, Kamel Barkaoui y Malika Ioualalen. «Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach». En: *Proc. of BPMN*. Vol. 9253. LNCS. Springer, 2015, págs. 55-71.

[60] Adrian Klein, Fuyuki Ishikawa y Shinichi Honiden. «SanGA: A Self-Adaptive Network-Aware Approach to Service Composition». En: *IEEE Trans. Serv. Comput.* 7.3 (2014), págs. 452-464. DOI: 10.1109/TSC.2013.2.

[61] Adrian Klein, Fuyuki Ishikawa y Shinichi Honiden. «Towards network-aware service composition in the cloud». En: *World Wide Web Conf. (WWW)*. ACM, 2012, págs. 959-968. DOI: 10.1145/2187836.2187965.

[62] Wolfgang Kratsch, Jonas Manderscheid, Maximilian Röglinger y Johannes Seyfried. «Machine Learning in Business Process Monitoring: A Comparison of Deep Learning and Classical Approaches Used for Outcome Prediction». En: *Bus. Inf. Syst. Eng.* 63.3 (2021), págs. 261-276. DOI: 10.1007/s12599-020-00645-0.

[63] Marta Kwiatkowska, Gethin Norman y David Parker. «Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach». En: *International Journal on Software Tools for Technology Transfer* 6 (nov. de 2001). DOI: 10.1007/s10009-004-0140-2.

[64] Duy Ngan Le, Ngoc Son Nguyen, Karel Mous, Ryan Kok Leong Ko y Angela Eck Soong Goh. «Generating Request Web Services from Annotated BPEL». En: *Intl. Conf. on Computing and Communication Technologies (IEEE-RIVF)*. 2009, págs. 1-8. DOI: 10.1109/RIVF.2009.5174641.

[65] Nebil Ben Mabrouk, Sandrine Beauche, Elena Kuznetsova, Nikolaos Georgantas y Valérie Issarny. «QoS-Aware Service Composition in Dynamic Service Oriented Environments». En: *Middleware*. Springer, 2009, págs. 123-142.

[66] Kim F. Man, Kit Sang Tang y Sam Kwong. «Genetic algorithms: concepts and applications [in engineering design]». En: *IEEE Transactions on Industrial Electronics* 43.5 (1996), págs. 519-534. DOI: 10.1109/41.538609.

[67] Farhad Mardukhi, Naser NematBakhsh, Kamran Zamanifar y Asghar Barati. «QoS decomposition for service composition using genetic algorithm». En: *Applied Soft Computing* 13.7 (2013), págs. 3409-3421. DOI: https://doi.org/10.1016/j.asoc.2012.12.033.

[68] Barbara Martini, Federica Paganelli, Paola Cappanera, Stefano Turchi y Piero Castoldi. «Latency-aware composition of Virtual Functions in 5G». En: *Conf. on Network Softwarization (NetSoft)*. 2015, págs. 1-6. DOI: 10.1109/NETSOFT.2015.7116188.

[69] John McCall. «Genetic algorithms for modelling and optimisation». En: *Journal of Computational and Applied Mathematics* 184.1 (2005). Special Issue on Mathematics Applied to Immunology, págs. 205-222. DOI: https://doi.org/10.1016/j.cam.2004.07.034.

[70] Nijat Mehdiyev, Joerg Evermann y Peter Fettke. «A Novel Business Process Prediction Model Using a Deep Learning Method». En: *Bus. Inf. Syst. Eng.* 62.2 (2020), págs. 143-157.

[71] José Meseguer. «A logical theory of concurrent objects». En: *Proceedings of the European Conference on Object-Oriented Programming on Object-Oriented Programming Systems, Languages, and Applications*. OOPSLA/ECOOP '90. Ottawa, Canada: ACM, 1990, págs. 101-115. DOI: 10.1145/97945.97958.

[72] José Meseguer. «Conditional rewriting logic as a unified model of concurrency». En: *Theoretical Computer Science* 96.1 (1992), págs. 73-155. DOI: https://doi.org/10.1016/0304-3975(92)90182-F.

[73] José Meseguer. «Rewriting as a unified model of concurrency». En: *CONCUR 90 Theories of Concurrency: Unification and Extension*. Ed. por J. C. M. Baeten y J. W. Klop. Berlin, Heidelberg: Springer, 1990, págs. 384-400.

[74] Seyedali Mirjalili. «Genetic Algorithm». En: *Evolutionary Algorithms and Neural Networks: Theory and Applications*. Springer, 2019, págs. 43-55. DOI: 10.1007/978-3-319-93025-1_4.

[75] Ahmed Moustafa y Minjie Zhang. «Multi-Objective Service Composition Using Reinforcement Learning». En: Service-Oriented Computing. Springer, 2013, págs. 298-312.

[76] Chun Ouyang, Eric Verbeek, Wil M.P. van der Aalst, Stephan Breutel, Marlon Dumas y Arthur H.M. ter Hofstede. «Formal semantics and analysis of control flow in WS-BPEL». En: *Sci. Comput. Program.* 67.2 (2007), págs. 162-198. DOI: https://doi.org/10.1016/j.scico.2007.03.002.

[77] José Antonio Parejo, Sergio Segura, Pablo Fernández y Antonio Ruiz-Cortés. «QoS-aware web services composition using GRASP with Path Relinking». En: *Expert Systems with Applications* 41 (9 2014), págs. 4211-4223. DOI: 10.1016/j.eswa.2013.12.036.

[78] Razvan Pascanu, Tomas Mikolov y Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG].

[79] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano y Donato Malerba. «Using Convolutional Neural Networks for Predictive Process Analytics». En: *2019 International Conference on Process Mining (ICPM)*. 2019, págs. 129-136.

[80] Peter Pfeiffer, Johannes Lahann y Peter Fettke. «Multivariate Business Process Representation Learning Utilizing Gramian Angular Fields and Convolutional Neural Networks». En: *19th International Conference on Business Process Management - BPM 2021*. Vol. 12875. Lecture Notes in Computer Science. Springer, 2021, págs. 327-344.

[81] Nicolás Pozas y Francisco Durán. «On the Scalability of Compositions of Service-Oriented Applications». En: *Intl. Conf. on Service-Oriented Computing (ICSOC)*. 2021, págs. 449-463.

[82] Nicolás Pozas García, Francisco Durán, Katia Moreno Berrocal y Ernesto Pimentel. «Location-aware scalable service composition». En: *Software: Practice and Experience* 53.12 (2023), págs. 2408-2429. DOI: https://doi.org/10.1002/spe.3260.

[83] PRISM. *Comprehensive Analysis and Powerful Statistics, Simplified*. 22 de mar. de 2024. URL: https://www.graphpad.com/features.

[84] Aurora Ramírez, José Antonio Parejo, José Raúl Romero, Sergio Segura y Antonio Ruiz-Cortés. «Evolutionary composition of QoS-aware web services: A many-objective perspective». En: *Expert Systems with Applications* 72 (2017), págs. 357-370. DOI: https://doi.org/10.1016/j.eswa.2016.10.047.

[85] Mohammad Reza Razian, Mohammad Fathian, Rami Bahsoon, Adel Nadjaran Toosi y Rajkumar Buyya. «Service composition in dynamic environments: A systematic review and future directions». En: *J. Syst. Softw.* 188 (2022), pág. 111290. DOI: 10.1016/j.jss.2022.111290.

[86] Camilo Rocha, José Meseguer y César. Muñoz. «Rewriting Modulo SMT and Open System Analysis». En: *Journal of Logical and Algebraic Methods in Programming* 86.1 (2017), págs. 269-297. DOI: http://doi.org/10.1016/j.jlamp.2016.10.001.

[87] Kumara Sastry, David Goldberg y Graham Kendall. «Genetic Algorithms». En: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, 2005, págs. 97-125. DOI: 10.1007/0-387-28356-0_4.

[88] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG].

[89] Stefano Sebastio y Andrea Vandin. «MultiVeStA: Statistical Model Checking for Discrete Event Simulators». En: *7th International Conference on Performance Evaluation Methodologies and Tools*. ValueTools '13. Torino, Italy: ICST, 2013, págs. 310-315. DOI: 10.4108/icst.valuetools.2013.254377.

[90] Koushik Sen, Mahesh Viswanathan y Gul Agha. «On Statistical Model Checking of Stochastic Systems». En: *Computer Aided Verification*. Ed. por Kousha Etessami y Sriram K. Rajamani. Berlin, Heidelberg: Springer, 2005, págs. 266-280.

[91] Koushik Sen, Mahesh Viswanathan y Gul Agha. «VeStA: A Statistical Modelchecker and Analyzer for Probabilistic Systems». En: *Second International Conference on the Quantitative Evaluation of Systems (QEST 2005), 19-22 September 2005, Torino, Italy*. IEEE Computer Society, 2005, págs. 251-252.

[92] Habiba Siddiqui, Imran Sarwar Bajwa, Dur-e-sameen Tabassum y Shabana Ramzan. «Producing BPEL models from text». En: *Intl. Conf. on Innovative Computing Technology (INTECH)*. 2016, págs. 471-477. DOI: 10.1109/INTECH.2016.7845066.

[93] Stelios Sotiriadis, Nik Bessis, Cristiana Amza y Rajkumar Buyya. «Elastic Load Balancing for Dynamic Virtual Machine Reconfiguration Based on Vertical and Horizontal Scaling». En: *IEEE Trans. Serv. Comput.* 12.2 (2019), págs. 319-334. DOI: 10.1109/TSC.2016.2634024.

[94] SPIN. *Verifying Multi-threaded Software with Spin*. 22 de mar. de 2024. URL: https://spinroot.com/spin/whatispin.html.

[95] Anja Strunk. «QoS-Aware Service Composition: A Survey». En: *8th IEEE European Conf. on Web Services (ECOWS)*. 2010, págs. 67-74. DOI: 10.1109/ECOWS.2010.16.

[96] Mihai Suciu, Denis Pallez, Marcel Cremene y Dumitru Dumitrescu. «Adaptive MOEA/D for QoS-Based Web Service Composition». En: Evolutionary Computation in Combinatorial Optimization. Springer, 2013, págs. 73-84.

[97] Niek Tax, Irene Teinemaa y Sebastiaan J. van Zelst. «An interdisciplinary comparison of sequence modeling methods for next-element prediction». En: *Softw. Syst. Model.* 19.6 (2020), págs. 1345-1365. DOI: 10.1007/s10270-020-00789-3.

[98] Irene Teinemaa, Marlon Dumas, Marcello La Rosa y Fabrizio Maria Maggi. «Outcome-Oriented Predictive Process Monitoring: Review and Benchmark». En: *ACM Trans. Knowl. Discov. Data* 13.2 (2019), 17:1-17:57. DOI: 10.1145/3301300.

[99] *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[100] Immanuel Trummer, Boi Faltings y Walter Binder. «Multi-Objective Quality-Driven Service Selection—A Fully Polynomial Time Approximation Scheme». En: *IEEE Transactions on Software Engineering* 40.2 (2014), págs. 167-191. DOI: 10.1109/TSE.2013.61.

[101] Abhishek Udupa, Ankush Desai y Sriram Rajamani. «Depth Bounded Explicit-State Model Checking». En: *Model Checking Software*. Ed. por Alex Groce y Madanlal Musuvathi. Berlin, Heidelberg: Springer, 2011, págs. 57-74.

[102] Asrin Vakili y Nima Jafari Navimipour. «Comprehensive and systematic review of the service composition mechanisms in the cloud environments». En: *J. Netw. Comput. Appl.* 81 (2017), págs. 24-36. DOI: 10.1016/j.jnca.2017.01.005.

[103] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi e Irene Teinemaa. «Survey and Cross-benchmark Comparison of Remaining Time Prediction Methods in Business Process Monitoring». En: *ACM Trans. Intell. Syst. Technol.* 10.4 (2019), 34:1-34:34. DOI: 10.1145/3331449.

[104] Hiroshi Wada, Junichi Suzuki, Yuji Yamano y Katsuya Oba. «E³: A Multiobjective Optimization Framework for SLA-Aware Service Composition». En: *IEEE Transactions on Services Computing* 5.3 (2012), págs. 358-372. DOI: 10.1109/TSC.2011.6.

[105] PengWei Wang, ZhiJun Ding, ChangJun Jiang, MengChu Zhou y YuWei Zheng. «Automatic Web Service Composition Based on Uncertainty Execution Effects». En: *IEEE Trans. Serv. Comput.* 9.4 (2016), págs. 551-565. DOI: 10.1109/TSC.2015.2412943.

[106] Paul Wilmott. *Machine Learning: An applied mathematics introduction*. Panda Ohana, 2019.

[107] Zhihui Wu, Piyuan Lin, Peijie Huang, Huachong Peng, Yihui He y Junan Chen. «A User Constraint Awareness Approach for QoS-Based Service Composition». En: *Intl. Conf. on Web Services (ICWS)*. Springer, 2019, págs. 48-62.

[108] Yi Xu y Abdelsalam Helal. «Scalable Cloud-Sensor Architecture for the Internet of Things». En: *IEEE Internet Things J.* 3.3 (2016), págs. 285-298. DOI: 10.1109/JIOT.2015.2455555.

[109] Håkan L. S. Younes y Reid G. Simmons. «Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling». En: *Computer Aided Verification*. Ed. por Ed Brinksma y Kim Guldstrand Larsen. Berlin, Heidelberg: Springer, 2002, págs. 223-235.

[110] Tao Yu y Kwei-Jay Lin. «Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints». En: *Intl. Conf. on Service-Oriented Computing (ICSOC)*. 2005, págs. 130-143.

[111] Tao Yu, Yue Zhang y Kwei-Jay Lin. «Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints». En: *ACM Trans. Web* 1.1 (2007), pág. 6. DOI: 10.1145/1232722.1232728.

[112] Yang Yu, Hui Ma y Mengjie Zhang. «F-MOGP: A novel many-objective evolutionary approach to QoS-aware data intensive web service composition». En: Congress on Evolutionary Computation (CEC). IEEE, 2015, págs. 2843-2850. DOI: 10.1109/CEC.2015.7257242.

[113] Yuan Yuan, Weishi Zhang, Xiuguo Zhang y Huawei Zhai. «Dynamic Service Selection Based on Adaptive Global QoS Constraints Decomposition». En: *Symmetry* 11.3 (2019). DOI: 10.3390/sym11030403.

[114] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam y Quan Z. Sheng. «Quality driven web services composition». En: *Intl. World Wide Web Conf. (WWW)*. ACM, 2003, págs. 411-421. DOI: 10.1145/775152.775211.

[115] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam y Henry Chang. «QoS-aware middleware for Web services composition». En: *IEEE Transactions on Software Engineering* 30.5 (2004), págs. 311-327. DOI: 10.1109/TSE.2004.11.

[116] Wei Zhang, Carl Chang, Taiming Feng y Hsinyi Jiang. «QoS-Based Dynamic Web Service Composition with Ant Colony Optimization». En: *IEEE 34th Annual Computer Software and Applications Conference*. Jul. de 2010, págs. 493-502. DOI: 10.1109/COMPSAC.2010.76.