UNIVERSITY OF MÁLAGA

DOCTORAL THESIS

A Requirements-Driven Approach for Building Architecture Projects of Self-Adaptive Systems

Author:
Patrícia ARAÚJO DE OLIVEIRA

Supervisor:
Dr. Ernesto PIMENTEL
Dr. Francisco DURÁN

A thesis submitted in fulfillment of the requirements for the degree of Doctor from the University of Malaga

in the

ITIS Software
Languages and Computer Science Department

June 15, 2022

AUTOR: Patrícia Araújo de Oliveira

https://orcid.org/0000-0002-2233-9609

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña PATRÍCIA ARAUJO DE OLIVEIRA
Estudiante del programa de doctorado TECNOLOGÍAS INFORMÁTICAS de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: A REQUIREMENTS-DRIVEN APPROACH FOR BUILDING ARCHITECTURE PROJECTS OF SELF-ADAPTIVE SYSTEMS

Realizada bajo la tutorización de ERNESTO PIMENTEL y dirección de Ernesto Pimentel y Francisco Duran (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 19 de febrero de 2022

Fdo.: Doctorando/a	Fdo.: Tutor/a

Fdo.:
Director/es de tesis

Sobre la tesis doctoral con título "A REQUIREMENTS DRIVEN APPROACH FOR BUILDING ARCHITECTURE PROJECTS OF SELF ADAPTIVE SYSTEMS", realizada por Patricia Araújo de Oliveira, los profesores Ernesto Pimentel Sánchez y Francisco Javier Durán Muñoz, del Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga:

- Confirman la idoneidad de la tesis para su presentación,
- Afirman que las publicaciones que avalan la tesis no han sido utilizadas en tesis anteriores, y
- Autorizan su lectura.

Ernesto Pimentel Sánchez
(tutor y co-director de la tesis)

Francisco Javier Durán Muñoz
(co-director de la tesis)

Acknowledgements

This work has been partially supported by Ministry of Science and Innovation and FEDER projects PGC2018-094905-B-100, UMA18-FEDERJA-180, and UMA-CEIATECH-08 by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech and by funding agency CNPq of the Ministry of Science and Technology (MCT), Brazil.

List of Figures

List of Tables

Contents

Resumen (Abstract in Spanish)

Motivación

Un sistema adaptativo puede modificar su configuración en tiempo de ejecución como respuesta a los cambios en su entorno operativo. Analizar este tipo de sistemas en tiempo de diseño es una tarea difícil que requiere considerar el sistema junto con las operaciones de adaptación, y tener en cuenta cómo actúan dichas adaptaciones sobre el sistema.

Para utilizar técnicas basadas en simulación para el análisis de tales sistemas, no solo necesitamos modelos ejecutables precisos de los sistemas a analizar, sino también capturar la semántica de sus mecanismos de adaptación. Dada la amplia gama y flexibilidad de las operaciones de adaptación, necesitamos formas que permitan la definición de nuevas operaciones.

Esta tesis tiene como objetivo abordar el problema de la construcción de proyectos de arquitectura para sistemas autoadaptativos que consideren la estructura general del sistema, la asignación, la funcionalidad, la comunicación, la gestión del dinamismo y las posibles modificaciones. Necesitamos considerar los cambios que pueden ocurrir a lo largo de su proceso de ejecución a partir de los requisitos definidos en el proyecto. Para reproducir estos cambios en la arquitectura en el momento del diseño a partir del análisis de los requisitos establecidos, consideramos el uso de la transformación de gráficos y un conjunto de otras herramientas para ayudar al proceso.

Palladio para la construción de proyectos de sistemas autoadaptativos

Palladio es una herramienta para la predicción de las propiedades de calidad de servicio (*Quality of Service* - QoS) de las arquitecturas de software basadas en componentes y puede usarse para modelar escenarios de diferentes dominios.

Hay cuatro aspectos importantes que se deben considerar en Palladio: (1) su proceso de desarrollo de software basado en componentes; (2) su metamodelo detallado de arquitecturas de software basadas en componentes, el PCM (*Palladio Componente Model*); (3) su simulador de arquitectura de software; y (4) y el Palladio-Bench. Con respecto a su proceso de desarrollo de software basado en componentes, el enfoque permite contribuciones en el modelado de escenarios de desarrollo de software basado en componentes. Este proceso de modelado usa el metamodelo PCM, lo que permite la creación de modelos de Palladio que pueden usar su simulador de arquitectura de software para medir métricas que pueden usarse para enfoques combinados como análisis de rendimiento, confiabilidad, facilidad de mantenimiento y pronóstico de costos. La implementación de todos los aspectos mencionados anteriormente está respaldada por la herramienta Palladio-Bench, que es extensible y puede servir como base para nuevos enfoques.

Para este trabajo, se consideraron todos los aspectos de Palladio. El DSL utilizado por Palladio es proporcionado por su metamodelo, el modelo de componentes de Palladio (PCM) (Becker, Koziolek, and Reussner, 2007), que se implementa mediante

el marco de modelado de Eclipse (EMF). La arquitectura del software se captura en PCM a través de la estructura estática, el comportamiento, la implementación/asignación, el entorno de recursos/entorno de ejecución y el perfil de uso. Así, los modelos de Palladio se componen de cuatro artefactos diferentes, proporcionados por los papeles involucrados en un proceso de desarrollo de Ingeniería de software basada en componentes (CBSE - *Component Based Software Engineering*) (Becker, Koziolek, and Reussner, 2009a): especificaciones de componentes (por desarrolladores de componentes), modelo *assembly* (por arquitectos de software), modelo de asignación (por desarrolladores de sistemas), y modelo de uso (por expertos en dominios comerciales).

La herramienta Palladio permite el análisis de modelos de sistemas estáticos, o sea, no es posible aplicar cambios al modelo. El enfoque Palladio-Simulizar (Becker, Becker, and Meyer, 2013) ha abierto la posibilidad de realizar análisis predictivos considerando cambios en el modelo inicial.

Los sistemas autoadaptativos modelados con SimuLizar pueden obtener predicciones generales del tiempo de respuesta del sistema. En SimuLizar, las reconfiguraciones son transformaciones del modelo al modelo PCM, el elemento administrado es el sistema autoadaptativo simulado modelado. Se monitorea el sistema simulado, se analizan los resultados del monitoreo, se planifica la reconfiguración y se realiza una reconfiguración en el sistema simulado si es necesario.

La herramienta Palladio es una de las opciones más robustas para el modelado de sistemas. Su extensión para el análisis predictivo de sistemas autoadaptativos, Simulizar, hereda la robustez de Palladio, sin embargo, presenta algunas limitaciones que impiden que la herramienta cumpla con las principales características de un sistema autoadaptativo.

Flexibilidad en el modelado de sistemas autoadaptativos

La herramienta e-Motions (Rivera, Durán, and Vallecillo, 2009) es un framework gráfico desarrollado para Eclipse que admite la especificación, simulación y análisis formal de sistemas. Es una herramienta que proporciona una forma de modelar sistemas específicos de dominio utilizando tanto el metamodelo de sintaxis abstracta como el metamodelo de sintaxis concreta, y le permite agregar reglas al metamodelo por transformación de grafos, posibilitando también el uso de OCL (Object Constraint Language), lo que proporciona una forma de especificar gráficamente los idiomas específicos del dominio (DSL). La sintaxis abstracta de un DSL se especifica como un metamodelo Ecore, que define todos los conceptos relevantes y sus relaciones en el lenguaje. Su sintaxis concreta viene dada por un modelo de Sintaxis Concreta Gráfica (GCS), que adjunta una imagen a cada concepto del lenguaje. Luego, su comportamiento se especifica con transformaciones de modelo.

La propuesta de integrar la robustez del metamodelo PCM,la flexibilidad visual y facilidad de e-Motions se inició en el trabajo de Moreno-Delgado et al., 2014, que proponía una reimplementación parcial basada en un modelo modular de una estructura de la arquitectura Palladio. El trabajo presentó la especificación de los principales DSL de Palladio en el sistema e-Motions, describiendo la semántica básica de simulación como un conjunto de reglas de transformación de gráficos. A partir de este trabajo, se hizo posible que los modelos creados en Palladio-Bench se puedan introducir directamente en el entorno de simulación de e-Motions para su análisis.

La propuesta de la reimplementación parcial de la arquitectura de Palladio abrió el camino para la flexibilización del modelado de sistemas. La propuesta es compatible con las funciones principales de Palladio para definir modelos de uso y modelos de componentes. Además, los modelos creados en Palladio se pueden introducir directamente en el entorno de simulación para su análisis.

Basado en la posibilidad de una configuración flexible para el análisis de sistemas, este trabajo pasa a un enfoque dinámico considerando la reimplementación parcial de Palladio en e-Motions. La idea central es considerar la posibilidad de añadir y retirar recursos y componentes de forma dinámica, abriendo un camino para el modelado y análisis de sistemas autoadaptativos. Para ello, es necesario incorporar recursos adicionales a la definición de Palladio así como comprender el comportamiento de un sistema autoadaptativo para facilitar el modelado y análisis, posibilitando así la experimentación de nuevas funcionalidades y soluciones personalizadas para problemas específicos a un mismo nivel con el bajo costo de desarrollo.

Construcción de Mecanismos de Adaptación

La definición de comportamiento dinámico en este trabajo ocurre a través del avance en el modelado del comportamiento de Palladio; en la modelización de mecanismos de adaptación; y en el modelado de control de requerimientos no funcionales. Todos estos pasos se construyeron utilizando la herramienta e-Motions. En e-Motions unimos todo en un único metamodelo (avanzando en los metamodelos Palladio existentes) y separamos las reglas de comportamiento en: reglas Palladio (avanzando en las reglas Palladio existentes), reglas de QoS, reglas de adaptación y reglas de red.

En presencia de un comportamiento dinámico, tiene que haber alguna forma de especificar cómo se gestiona y controla la adaptación. Este enfoque utilizó SYBL que, aunque no fue diseñado para ser utilizado para el análisis predictivo, lo consideramos muy apropiado para nuestro propósito debido a su capacidad para describir las limitaciones, el seguimiento y las estrategias de adaptación. Otro avance significativo de este trabajo a respecto a la implementación de Palladio en e-Motions fue el modelado de la comunicación entre contenedores.

Los diferentes mecanismos de adaptación disponibles se definen como reglas de transformación sobre el modelo del sistema bajo análisis. Por lo tanto, dada la información de monitoreo, los sistemas pueden adaptarse de diferentes maneras al realizar diferentes operaciones.

Proceso de modelado para sistemas autoadaptativos

Propusimos un framework siguiendo diferentes fases de un proceso: Especificación, Modelado, Definición y Análisis de Comportamiento.

En la fase de Especificación, los requisitos no funcionales y las estrategias de adaptación se especifican utilizando el lenguaje SYBL (Copil et al., 2013). En la fase de Modelado se modela en la herramienta Palladio el modelo inicial del sistema a analizar, en el que se obtienen las diferentes vistas de la aplicación. Este modelo se puede definir de acuerdo con las especificidades de cada aplicación a analizar.

En la fase de Definición de Comportamiento se consideran las reglas definidas por la transformación de grafos en e-Motions. Estas reglas se refieren al comportamiento

de los DSL. Hemos definido reglas que representan el comportamiento de Palladio, reglas de adaptación y reglas que definen el comportamiento de los mecanismos de control de adaptación considerando requisitos no funcionales. Todas estas reglas se pueden modificar fácilmente según sea necesario. Además, en esta fase se genera el código Maude que se obtiene mediante un proceso de transformación de modelos proporcionado por la herramienta e-Motions. La especificación de Maude obtenida puede ser utilizada para realizar la simulación en la fase de Análisis. La transformación de e-Motions toma como entrada: (1) el modelo inicial, definido en Palladio en la fase de Modelado; (2) el metamodelo extendido del modelo de componentes de Palladio (PCM); (3) la especificación e-Motions de la definición de comportamiento de Palladio; (4) la definición conductual de los mecanismos de adaptación; y (5) la definición conductual del control para mecanismos de adaptación y requisitos no funcionales.

En la fase de Análisis se utiliza el sistema Maude para ejecutar las simulaciones de los códigos Maude generados en la fase anterior. Durante y al final de las simulaciones, es posible visualizar sus resultados, que contienen los datos relacionados con el sistema simulado. Estos datos muestran los valores de la variación de las métricas de QoS monitoreadas durante la simulación así como, en su caso, las modificaciones al modelo ocurridas luego de las adaptaciones realizadas. Estos datos permiten el análisis y, posteriormente, los ajustes en el modelo inicial o en los parámetros, según sea necesario.

Desafíos de la investigación

El diseño de sistemas de software, cada vez más grandes y complejos, se ha convertido en un desafío debido a la necesidad de tener en cuenta no solo la estructura general de los propios sistemas, sino también su asignación, funcionalidad y la comunicación de sus componentes. En este sentido, la comunidad de Ingeniería del Software debe concentrar esfuerzos en la construcción de enfoques que apunten a establecer las arquitecturas de los sistemas para su implementación, pero que también sean adecuadas para el análisis de sus atributos de calidad, como el rendimiento.

Los estudios relacionados con el desempeño de los sistemas autoadaptativos se encuentran entre los más recurrentes dentro de la comunidad académica en los últimos años y presentan una serie de desafíos para la evaluación de la calidad de los sistemas autoadaptativos en cuanto a arquitectura, complejidad, valores de referencia y herramientas. Consideramos que los posibles caminos para superar estos problemas se basan en el esfuerzo de construir soluciones en el momento del diseño (de Sousa et al., 2019).

En relación a la arquitectura, los desafíos de no poder verificar todas las posibilidades de adaptación (Criado et al., 2018 Criado et al., 2016) y la dificultad de evaluar diferentes arquitecturas (Chen et al., 2019) podrían tener como alternativa de solución en tiempo de diseño un conjunto de evaluaciones de rendimiento de diseño de diferentes adaptaciones y arquitecturas simuladas en tiempo de diseño, que ayudaría en la toma de decisiones en tiempo de ejecución, ya que el sistema tendría posibles caminos de adaptación predefinidos y podría elegir las adaptaciones más apropiadas para un determinado situación. Este camino facilitaría la construcción de valores de referencia iniciales para ayudar en la interpretación de los resultados, lo que se señala como otro desafío (Edwards and Bencomo, 2018 Bezerra et al., 2018 Serral, Sernani, and Dalpiaz, 2017).

Sin embargo, estos posibles caminos de solución nos llevan a otro desafío: la complejidad de un sistema autoadaptativo debe ser considerada en simulaciones y existe una dificultad para reproducir estas condiciones en escenarios de aplicación del mundo real (Franco et al., 2016) (Sanislav, Mois, and Miclea, 2015). Por lo tanto, las líneas de investigación que buscan avanzar en las áreas de análisis de desempeño de los sistemas autoadaptativos en el momento del diseño son fundamentales en un intento de superar los problemas aún presentes en el área. Los proyectos en esta dirección deben considerar la complejidad típica de estos sistemas y la propuesta de herramientas para automatizar y ayudar en las tareas de pruebas y evaluación de la calidad.

De acuerdo con el análisis presentado en Raibulet et al., 2017 – en el que se realizó una revisión de algunos enfoques para la evaluación de sistemas autoadaptativos presentes en la literatura –, cuando se consideran atributos de calidad (como rendimiento y confiabilidad) utilizados en de los trabajos consultados en su estudio, el 95% de ellos se evalúan en tiempo de ejecución y el 5% en tiempo de diseño. Los autores también señalaron que ninguno de estos enfoques asoció estas evaluaciones a una herramienta, lo que puede dificultar la parte evaluativa.

De hecho, se encuentran pocos estudios y herramientas en la literatura que aborden la cuestión de evaluar el desempeño de los sistemas autoadaptativos en el momento del diseño. En cuanto a las herramientas, la herramienta Palladio (Happe, Koziolek, and Reussner, 2011), por ejemplo, es uno de los marcos de análisis predictivo más exitosos en la actualidad y se usa ampliamente tanto en la industria como en la academia. Aunque el Simulador de Palladio (SimuCom) (Becker, Koziolek, and Reussner, 2009b) se puede utilizar para predecir las propiedades de calidad de servicio (QoS) (rendimiento y confiabilidad) a partir de modelos de arquitectura de software, se limita al análisis de sistemas estáticos. SimuLizar, propuesto por Becker, Becker, and Meyer, 2013, pretendía ampliar Palladio para el análisis de rendimiento de sistemas autoadaptativos en el momento del diseño. El enfoque es efectivo, el comportamiento del sistema y las adaptaciones está encapsulado en la plataforma, lo que dificulta la inserción o modificación de diferentes dominios de aplicación (este es un aspecto importante considerando la constante evolución y cambio en la aplicación de sistemas autoadaptativos).

Publicaciones principales

El desarrollo de esta tesis ha dado lugar a las siguientes publicaciones (clasificadas por categoría y en orden cronológico inverso):

Artículo en Revista

- ARAÚJO-DE-OLIVEIRA, PATRÍCIA; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. A procedural and flexible approach for specification, modeling, definition, and analysis for self-adaptive systems. Softw Pract Exper. 2021;51:1387–1415. https://doi.org/10.1002/spe.2962

Capítulo de Libro

- DE OLIVEIRA, PATRÍCIA ARAÚJO; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. Towards the Performance Analysis of Elastic Systems with e-Motions. In: Cerone

A., Roveri M. (eds) Software Engineering and Formal Methods. SEFM 2017. Lecture Notes in Computer Science, vol 10729, p. 475-490. Springer, Cham.
- DE OLIVEIRA, PATRÍCIA ARAÚJO; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. An Approach to Predictive Analysis of Self-Adaptive Systems in Design Time. In: Braubach L. et al. (eds) Service-Oriented Computing – ICSOC 2017 Workshops. ICSOC 2017. Lecture Notes in Computer Science, vol 10797, p. 363-368. Springer, Cham.

Conferencia Internacional

- DE OLIVEIRA, PATRÍCIA ARAÚJO. Predictive Analysis of Cloud Systems (Extended Abstract). In: International Conference on Software Engineering (Companion Volume), 2017: 483-484. Buenos Aires, Argentina.

Conferencia Iberoamericana

- DE OLIVEIRA, PATRÍCIA ARAÚJO; MORENO-DELGADO, ANTÔNIO; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. Towards the predictive analysis of cloud systems with e-Motions. In: Ibero-American Conference on Software Engineering, 2017: 169-182. Buenos Aires, Argentina.

Conferencia Española

- DE OLIVEIRA, PATRÍCIA ARAÚJO.Towards the model-based predictive performance analysis of Cloud adaptive systems with e-Motions. In: Jornadas sobre PROgramación y LEnguajes (PROLE), 2017, Tenerife. Jornadas de la Asociación de Ingeniería del Software y Tecnologías de Desarrollo de Software (SISTEDES), 2017.

Conclusiones y contribuciones

Al considerar la escalabilidad, la elasticidad y la adaptabilidad de los sistemas, el dinamismo y la autonomía son requisitos desafiantes, y considerarlos en el momento del diseño sigue siendo una tarea difícil. Los comportamientos emergentes y las interacciones oportunistas aún no se pueden predecir en el momento del diseño. Sin embargo, estos esfuerzos deben centrarse principalmente en soluciones relacionadas con el establecimiento de modelos que comprendan la naturaleza cambiante de estos sistemas para que los proyectos modelen los servicios y sus adaptaciones de manera más eficiente. Estos modelos deben ser probados para que sea posible encontrar lineamientos capaces de ayudar en la construcción de sistemas para que mitiguen retrasos y fallas, y que no impacten directamente en la experiencia de los usuarios. Estas directrices se refieren a un conjunto de decisiones en tiempo de diseño, como la especificación de requisitos que consideran restricciones, estrategias de seguimiento y adaptación, así como el modelado de sistemas y su comportamiento, y el análisis del sistema en tiempo de diseño para construir modelos con mejores resultados de calidad de servicio.

Este trabajo propone utilizar Palladio, e-Motions, Maude y SYBL de tal manera que permite expresividad y flexibilidad en la especificación, modelado, definición de comportamiento y análisis de rendimiento de modelos autoadaptativos utilizando

un enfoque procedimental. Tal organización procedimental nos ha ayudado a comprender la formulación de cada uno de los pasos, lo que hace que el enfoque sea flexible y consistentemente reproducible.

Modelamos los mecanismos de adaptación como reglas genéricas de adaptación. Hemos ilustrado nuestro enfoque modelando reglas que se activan en respuesta a infracciones de restricciones. Especificamos los requisitos de elasticidad de los sistemas, permitiendo su ajuste. La verificación de la especificación y la adecuación en diferentes componentes del sistema nos permitió verificar que las estrategias previamente especificadas pueden ser engañosas y nuestro enfoque propone que el reajuste y análisis se realice en tiempo de diseño. Con el desarrollo de las reglas de comportamiento del canal de comunicación, es posible considerar la variación de metrica de QoS y las adaptaciones considerando el enlace de comunicación.

El procedimiento y la experiencia práctica nos han permitido validar nuestra hipótesis. Las especificaciones iniciales, el modelo y las adaptaciones se probaron en tiempo de simulación, y los resultados obtenidos mediante su uso durante una simulación se sometieron a re-simulación y reanálisis, lo que nos llevó a un diagnóstico más preciso del sistema y, eventualmente, a un reajuste hacia mejores resultados. Esto fue posible gracias a la posibilidad de modificar el modelo durante el tiempo de simulación, la facilitación de la escrita de las especificaciones de requisitos utilizando un lenguaje sencillo como SYBL, y el posterior análisis del impacto de las adaptaciones en el sistema.

1 Introduction

Architectural projects are essential in Software Engineering since it is important to recognize the common structures existing in different systems, as well as to understand the relations between them and to reuse this knowledge in new systems. A well-structured project allows us to analyze and describe the properties of a complex system towards a general and complete overview, thus enabling the alternative decision-making in the face of possible problems.

As the size and complexity of software systems have increased, software architecture has become an essential element in building consistent systems. The necessity to take into account not only the overall structure of the system itself but also the allocation, functionality and communication of the software components became a challenge for building software architecture projects. These definitions must establish the system structure consistently for implementations and are directly related to quality attributes, such as reliability and performance (non-functional requirements). Providing services efficiently without delays or failures is crucial for any software system, as it directly impacts the user experience.

In applications for the context of Smart Cities, for example, the proposal of using heterogeneous architectures brought a way to dynamically manage applications and consider their performance. The use of cloud computing and edge computing can compose a heterogeneous architecture that seeks, in a self-adaptive way, to overcome possible performance problems of this type of application (Nardelli et al., 2017).

Koziolek, 2010 and Balsamo et al., 2004 indicate that the predictive analysis of non-functional requirements at design-time might mitigate development costs and failure risks. Indeed, discovering at late stages of the development process that a software system does not meet certain non-functional requirements can be very harmful. The late identification of these issues may require changes that can aggravate the situation, both in terms of costs and risks.

The management of these characteristics in a Self-Adaptive System (SAS) is a challenge that, in addition to complexity, has dynamism as its main feature. It is necessary to considerate possible modifications of its initial configurations in response to changes in the context, which can compromise its functioning due to certain situations such as inconsistent context, poor performance and failures to adapt. This makes the construction of these systems and the decision-making at design time difficult tasks, since the initial model can take different forms and configurations during the execution process.

1.1 Summary of the State of the Art

Studies related to the performance of SAS are amongst the most recurring ones within the academic community in the last years, and they present a series of challenges for quality evaluation of self-adaptive systems regarding architecture, complexity, reference values and tools. We consider that possible paths for overcoming these issues rely on the effort to build solutions at design time (de Sousa et al., 2019).

In relation to architecture, the challenges of not being able to verify all the possibilities for adaptation (Criado et al., 2018 Criado et al., 2016) and the difficulty of evaluating different architectures (Chen et al., 2019) could have as an alternative solution at design-time a set of design performance evaluations of different adaptations and architectures simulated at design-time, which would assist in decision-making at runtime, since the system would have possible predefined adaptation paths and it could choose the most appropriate adaptations for a given situation. This solution would facilitate the construction of initial reference values to assist in the interpretation of results, which is pointed out as another challenge (Edwards and Bencomo, 2018 Bezerra et al., 2018 Serral, Sernani, and Dalpiaz, 2017).

However, these possible solution paths lead us to another challenge: the complexity of a SAS must be considered in simulations and there is a difficulty in reproducing these conditions in real-world application scenarios (Franco et al., 2016 Sanislav, Mois, and Miclea, 2015). Thus, research paths that strive to advance in the areas of performance analysis of self-adaptive systems at design-time are fundamental in an attempt to overcome problems still present in the area. Projects in this direction must consider the typical complexity of these systems and the proposal of tools to automate and assist in the tasks of tests and quality assessment (also pointed out as a challenge by Franco et al., 2016 Bezerra et al., 2016).

According to the analysis presented in Raibulet et al., 2017 – in which it was performed overview of some approaches for the evaluation of self-adaptive systems present in the literature –, when considered quality attributes (such as performance and reliability) used in the consulted works in their study, 95% of them are evaluated at runtime and only 5% at design-time. The authors also pointed out that none of these approaches associated these evaluations to a tool, which may hamper the evaluative part.

Indeed, few studies and tools are found in the literature that address the question of evaluating the performance of self-adaptive systems at design-time. As for tools, The Palladio tool (Happe, Koziolek, and Reussner, 2011) for instance is one of the currently most successful predictive analysis frameworks, and is widely used both in industry and academia. Although the SimuCom (Palladio Simulator) (Becker, Koziolek, and Reussner, 2009b) can be used to predict Quality of Service (QoS) properties (performance and reliability) from software architecture models, it is limited to the analysis of static systems. SimuLizar, proposed by Becker, Becker, and Meyer, 2013, was intended to extend Palladio for the performance analysis of self-adaptive systems at design-time. Even though the approach is effective, the behavior of the system and adaptations is encapsulated in the platform, which hinders the insertion or modification of different application domains (this is an important aspect considering the constant evolution and change in the application of self-adaptive systems).

Saputri and Lee, 2020 conducted a study that found that the application of machine learning (ML) techniques in SAS has increased in recent years and this can be mainly

explained by the trend of using machine learning techniques in software engineering, which typically uses ML to detect software patterns and problems. In addition, the authors point out the use of ML in studies of efficient methods of algorithms to optimize systems. The authors pointed out as a result of their literature review that the implementation of ML in primary studies presents highly reactive results, that is, works that implement techniques that allow the system to change behavior according to the current situation. Furthermore, the authors point to ML as capable of supporting decision-making through the analysis and active learning of the information collected. Thus, the authors claim that ML can deal with SAS uncertainties by dynamically learning new adaptive rules and modifying existing ones. However, most studies ignore its domain characteristics and available knowledge, and they do not have a proper justification for choosing an ML technique. It is necessary, for example, a performance validation of the adaptation mechanism. The authors point out that guidelines are needed for the adoption of the ML technique to improve the adaptation process in a SAS. In that regard,Casimiro et al., 2021 discusses a set of causes for misbehaving ML components in a SAS and proposes necessary changes to the MAPE-K loop for ML-based systems.

Casimiro et al., 2021 also points out that, despite the emergence of numerous ML techniques that could be used as adaptation strategies in SAS Rabanser, Günnemann, and Lipton, 2019, Pinto, Sampaio, and Bizarro, 2019, Huang et al., 2011, Cao and Yang, 2015, Miller et al., 2016, Wu, Dobriban, and Davidson, 2020, determining when and how to take advantage of such strategies to make adaptations is not a trivial task. The authors point out that ML-based components may not perform as expected. Some problems still need to be overcome, such as the dataset shift caused by changes in the operating environment, which can cause problems in the inputs of the ML models Quionero-Candela et al., 2009, or even problems with attacks that try to subvert the intended functionality of the system Gu et al., 2019.

1.2 Motivation and Objectives

This thesis aims to address the problem of building architecture projects for self-adaptive systems that consider the general structure of the system, allocation, functionality, communication, management of dynamism and possible modifications. For the construction of architecture projects of self-adaptive systems we need to consider the changes that may occur throughout their execution process from the requirements defined in the project. In order to reproduce these changes in the architecture at design-time from the analysis of established requirements, we considered the use of graph transformation and a set of other tools to assist the process. Thus, the main research question addressed in this thesis is

RQ *How to obtain an architectural design that considers the main characteristics of a self-adaptive system and that takes into account possible changes from the design-time performance analysis driven by the design requirements?*

Some objectives have been derived from this research question and are exposed in the following.

1.2.1 General Objectives

In order to answer the research question of this thesis, three main general objectives were derived with respect to the techniques for discarding information and the errors that may arise.

- It is intended to facilitate the construction of architecture projects of self-adaptive systems that consider the general structure of the system, allocation, functionality, communication, management of dynamism and possible modifications; for this, one must consider the modeling of the system and its application scenario, its behavior, its analysis and its modification at design-time;
- To consider the main characteristic of a self-adaptive system, it is intended to build adaptation mechanisms that are easy to understand, that facilitate their modification and addition of new mechanisms;
- Finally, we intend to build a guide to indicate the best practices for building self-adaptive architectures at design-time considering requirements.

1.2.2 Specific Objectives

In addition, the following specific objectives were also derived from the research question of this thesis:

- To build self-adaptive architectures, we need to understand the main existing tools, and choose and/or modify the approach that meets our goals;
- Furthermore, we need to define how to model adaptation mechanisms that respect intrinsic characteristics of self-adaptive systems and design requirements;
- Once the tool or method has been defined and the adaptation mechanisms have been modeled, we will perform performance analyzes to understand whether it is possible to contemplate the main characteristics of a self-adaptive system at design-time, considering functional and non-functional requirements and the use of different technologies and possible adaptations.

1.3 Methodology

As a starting point, we sought to understand tools for building architecture projects of self-adaptive systems. A study of the Palladio tool and Simulizar (its version for self-adaptive systems) was carried out. Then, the implementation proposal of Palladio was carried out in depth using graph transformation in e-Motions.

For the construction of models that represent adaptive mechanisms, some self-adaptive strategies were selected and modeled using graph transformation with the objective of allowing a structure capable of being modifiable, scalable and elastic (characteristics of a self-adaptive system) and that consider non-functional system requirements for design-time adaptation.

From the study of the tools and the construction of models of adaptive mechanisms, a proposal was built that incorporated the characteristics of self-adaptive systems from the integration of Palladio, e-Motions and the SYBL elastic resource control language. Thus, it was possible to outline the modeling of self-adaptive systems that consider their main characteristics: general system structure, allocation, functionality, communication, management of dynamism and possible modifications in response to the analysis of requirements and performance evaluation at design-time.

. Finally, an architecture of a self-adaptive system was built considering functional and non-functional requirements and the use of different technologies and possible adaptations in order to illustrate how to model a complex self-adaptive system.

1.4 Contribution of this Thesis

In this thesis, we identified that the use of a flexible and procedural approach can incorporate changes in the model that are necessary for a self-adaptive system, being useful to measure and model the unpredictability of these systems, since the composition and/or integration of different services require a detailed analysis of the adaptation choices. This approach differs from the others because it proposes the creation of models that consider and understand the changing nature of self-adaptive systems, from the possibility of testing them considering the constant change of quality parameters, requirements and context after adaptations occur, understanding that they can affect the system as a whole, even if applied to a specific component. This set of design-time decisions and modifications, such as specifying requirements that consider constraints, monitoring and adapting strategies, as well as modeling systems and their behavior, and system analysis, can be used as a starting point for the construction of systems with better quality of service results, since the behavior of the application is understood even when adaptations occur, still in the design phase.

The main contributions of this work include the use of different concepts and definitions incorporated into existing tools in order to facilitate support for adaptability control and the reformulation of established mechanisms by verifying the impact of adaptations on the system model.

The use of different tools brought us the possibility of using existing, consolidated and tested tools, which, however, do not meet the dynamic nature of self-adaptive systems. Thus, each one of them was complementary so that we could propose the approach of building and modifying adaptive models. In particular, the e-Motions tool was a key piece, since it uses the Maude language to simulate the constructed graph transformation models, as it is a high-performance language that supports logical rewriting, provides executable semantics and can handle status and simultaneous calculations.

Finally, the use of Model-Driven Engineering (MDE), Component-Based Software Engineering (CBSE) and Graph Transformation for specifying, modeling, defining and analyzing self-adaptive systems at design time facilitates their modification and understanding, in addition to making the approach more flexible.

1.5 Outline of this Thesis

This thesis is structured as follows: chapter 2 presents the background with the characterization of self-adaptive systems, concepts and definitions of Model-Driven Engineering (MDE), Component-Based Software Engineering (CBSE) and graph transformation; chapter 3 presents the Palladio Tool for building self-adaptive system projects; chapter 4 shows flexibility in modeling self-adaptive systems; chapter 5 illustrates how to conduct the building of the adaptation mechanisms; chapter 6

presents the modeling process for self-adaptive systems, considering the phases approach: specification, modeling, behavior definition and analysis; chapter 7 illustrates how the modeling process based on functional and non-functional requirements takes place using these tools applied to a very dynamic scenario which involves Cloud and IoT services; chapter 8 discusses the obtained results; and, finally, chapter 9 presents the conclusion of the thesis and discusses future works.

2 Background

Johnsen, Lin, and Yu, 2016 propose a model-based prediction approach to compare the effect, in terms of performance and accumulated cost, of selecting different instance types of virtual servers from Amazon Web Services (AWS). In order to do this comparison they use a highly configurable modeling framework for applications running on Apache YARN, the ABS-YARN, using the executable semantics of Real-Time ABS, defined in Maude, as a simulation tool. However, they do not model the application but instead use values obtained from real measurements.

There are other approaches that propose different solutions for evaluating non-functional properties using models, such as the one by Bernardi et al., 2018 who propose the assessment of non-functional properties using multi-formalism approaches, but they do not address self-adaptive systems. In particular, they are proposing an integrated model to combine performance and reliability models, which allows the construction of models for performance analysis, and automated construction of multi-formalism models with less effort. Our purpose follows a different path, integrating models and languages in the e-Motions framework, which allows the performance analysis of non-functional properties, specifically, on self-adaptive systems, thus introducing an additional degree of advancement in the challenge of assessing non-functional properties of systems models.

2.1 Characterization of a Self-Adaptive System

A self-adaptive system is one that has the ability to adapt to needs that arise during its execution process, making changes autonomously and thus giving a dynamic feature to the application. Such needs may be linked to changes in the execution environment, the availability of resources or even changes related to the needs of users.

The search for this autonomy and dynamism is not a trivial task, as it is necessary to deal with uncertainties in behavior without interruptions in the system. The system must collect data, manage itself, reconfigure itself when necessary and act on changes.

In this work we propose a different look to modeling, allowing it to be done during design-time for self-adaptive systems, which may help designers understand the

system before its implementation, deployment and execution. This was made possible through the advances in model-driven engineering research, which allowed the flexibilization of modeling and the simulation of models at design-time. For this to be possible, it is necessary to understand the behavior and the main features of a self-adaptive system.

Weyns, 2020 establishes two basic principles that determine what a self-adaptive system is, based on two interpretations most commonly found in the literature - this characterization is fundamental for understanding how to deal with this type of system. The first principle concerns the external aspect of a self-adaptive system, which can adjust its behavior in response to changes and uncertainties autonomously, without (or with the minimum) human intervention. The ability to deal with uncertainties, which are traditionally the operators' tasks, makes this interpretation address an essential feature of a self-adaptive system: to deal with external changes related to the availability of resources, the increase or decrease of the workload and the demands, and failures and/or threats - which makes the system also assume the role of an external observer.

The second principle concerns internal aspects that are divided into two parts: the first part deals with questions related to the application domain and the other deals with questions related to adaptation. The second part monitors the first one and acts if a violation of some pre-established criteria is identified, that is, that does not meet the objectives of the users for whom the system was designed. This second interpretation plays the role of the system engineer and aims to deal with self-adaptation by analyzing it from the point of view of how the system was designed.

For more than two decades, researchers have been working to solve existing challenges when it comes to self-adaptation. Weyns, 2020 called "seven waves" the efforts and research related to self-adaptive systems over time.

The first wave consists of the automation of tasks resulting in advances related to the management of complex software systems, allowing modifications in the systems to deal with the variability of resources, changes in user needs and system failures, without the need to paralyze its operation, giving software systems the ability to be easily modifiable over their lifetime. The second wave consists of dealing with self-adaptive systems from a systematic engineering perspective and was called "architecture-based adaptation". The third wave arose with the need to think about the complexity of a concrete project of these types of systems, dealing mainly with runtime models as a design solution for self-adaptive systems (as presented in Vogel and Giese, 2018), as they considered that only at runtime it is possible to know the system uncertainties, and also to predict and plan the changes.

From the understanding of the need for requirements-driven adaptations, the fourth wave seeks to understand the definition of requirements for self-adaptive systems, also considering the feedback loops defined in architecture-based adaptations, in which models and project goals are defined. The fifth wave focused efforts on research related to solutions that sought guarantees in the face of uncertainties.

From the research carried out in the second and fifth waves, the sixth wave arose due to the need for a theoretical framework for self-adaptation, using complexity to provide guarantees for control-based software adaptation. Finally, the seventh wave joins efforts with the previous ones in search of learning from experience, dealing

with an increasing scale and increasingly complex levels of uncertainty. It is important to state that the waves do not replace one another, but are, instead, different developments of the research process.

All waves provided great advances in the perspective of software engineering for self-adaptive systems. However, gaps still persist in some of these waves. For example, in the third wave, there is a separation between traditional models and models of self-adaptive systems, and the understanding that a self-adaptive system should consider design models at runtime, due to the fact that models at design-time are limited regarding uncertainties, prediction and self-adaptation planning.

This limitation is mainly due to the fact that models at design-time are static and do not allow an analysis of the models in order to adapt autonomously to deal with changing conditions. In this work we will discuss how we can advance research related to design-time models that are capable of not only delivering models with initial specifications for the development of software systems, but also being able to deliver models at design-time with a perspective of adaptation actions that can be performed at runtime.

One of the challenges related to the waves presented is to define domain-specific modeling languages that incorporate knowledge of the domain and, consequently, enable the conservation, reuse, and further development of knowledge and know-how. We will discuss throughout this work how we can generate a domain specification for a self-adaptive system from the definition of a specific domain language in a graphic transformation tool that will allow us to carry out simulations and analysis.

2.2 Model-Driven Engineering (MDE)

With the advance in research in the Model-Driven Engineering (MDE) area, software models started to have a greater importance in the development of solutions. They stopped being just documentation artifacts and became software engineering artifacts, allowing the creation or automatic execution of software systems from models and considering concepts and terms that are discussed, abstracted and understood by the scientific community (Rodrigues da Silva, 2015).

In this way, MDE has become a key part for the development of domain-specific software, since developing these systems requires not only an understanding of the programming languages or technologies involved, but also the broad knowledge and understanding of the solution and the application domain. Thus, MDE became an outlet in these contexts since the development of applications based on the definition of models moves away from the complexity of the platform and gives space to be closer to the problem domain than to the implementation domain.

MDE can be understood as a software development methodology which Software Developers (SDs) use to create abstract software descriptions and facilitate the generation of the implementation code (Combemale et al., 2016), focusing on the creation and exploitation of domain models related to a specific problem (Koussaifi et al., 2020). The Table 2.1 presents some resources used in MDE listed in Rodrigues da Silva, 2015.

To allow SDs to work more directly with a domain, MDE makes use of modeling languages defined for a specific domain, called Domain-Specific Modeling Languages

TABLE 2.1: Some resources used in MDE (Rodrigues da Silva, 2015)

Resource	**What it is**	**For what it is**
Modeling language (ex. UML)	Defined as metamodels	Formalizing requests, structures and behaviors of the app in a specific domain
Formal rules	Rules added to the metamodel to verify the model and detect and prevent errors before generating the code	Checking if the instances are in accordance with the model (using for example Object Constraint Language - OCL)
Abstract syntax metamodeling	The abstract syntax specifies the logical relations of the language - defined using metamodeling	Specifying the structure of the modeled system (for example: class diagram that characterizes logical structures of objects)
Concrete syntax metamodeling	It defines how the language is actually written	A substantial benefit of MDE sets in model transformation mechanisms that are used to produce various types of artifacts, such as source code, deployment descriptors, or other models
Possibility to define Domain-Specific Languages (DSLs)	A DSL is a dedicated language (it can be used for programming, modeling or specification) for expressing and solving problems in a specific domain	By defining the actions that can be taken by SDs, it allows them to manipulate and edit specific models
Model editors	They can be graphic or textual	Managing models

(DSMLs), which formalize the structure, behavior and requirements of the application in specific domains, following the abstractions and semantics of the domain. Thus, SDs can focus directly on the concepts and specificities of each domain considering three aspects: the concepts and domain rules (abstract syntax), the representations of these concepts, whether graphic or textual (concrete syntax), and the verification of the model, with formal rules of logical rewriting, for example (semantics).

MDE allows for a higher level of abstraction and, combined with Component-Based Software Engineering (CBSE), it can help to master the complexity and dynamics of modern software systems (Ciccozzi et al., 2017) and it is used to support development, deployment and adaptation at runtime of self-adaptive systems, also supporting the design of feedback loops and their execution at runtime (Vogel and Giese, 2018; Koussaifi et al., 2020).

MDE has already established itself as a methodology to increase the productivity of complex systems and to reduce the time to launch to the market, since the models can be read and explored programmatically to simplify the design, implementation, execution and evolution of software systems (Ciccozzi, Cicchetti, and Wortmann, 2020).

This work we used two MDE frameworks: Palladio and e-Motions. We use Palladio (Happe, Koziolek, and Reussner, 2011) for the system modeling, and e-Motions (Rivera, Durán, and Vallecillo, 2009) to specify Palladio's operational semantics and the adaptation of Palladio systems over time, what allows us to simulate and analyze Palladio-like adaptive systems.

2.3 Component-Based Software Engineering (CBSE)

Component-Based Software Engineering (CBSE) came from the idea proposed by McIlroy et al., 1968 of producing commercial components similar to the ones found in other branches of engineering. However, most research work on CBSE has emerged in recent decades (Vale et al., 2016).

The main focus of CBSE is the reuse of components that can be incorporated into different applications. cite Vale2016 indicated that among the most commonly mentioned objectives for the application of CBSE are: increasing productivity, saving costs, increasing quality, increasing reuse, decreasing complexity, increasing maintenance, increasing flexibility, decreasing risks, increasing efficiency, increasing adaptability and optimizing evolution. Among the objectives, we highlight the issue of decreasing complexity, increasing flexibility and increasing adaptability. The design and development of complex systems, such as self-adaptive ones, require the definition of approaches and the development of methods to add, remove, replace, modify and assemble components quickly and dynamically during operation, which makes it necessary to build more flexible systems.

Both CBSE and MDE (section 2.4) are considered as ways to try to reduce complexity in the development of systems. MDE focuses mainly on the models as a way to abstract the development process and bring it closer to the specific domain of the application, and CBSE allows the division of the desired resources and their complexity in smaller parts, allowing the construction of a system in an incremental and flexible way.

In this work we consider the Palladio tool (Happe, Koziolek, and Reussner, 2011), which, beyond being an MDE framework, is a tool that assumes a CBSE development process, in which component developers specify and implement parametric descriptions of components and their behavior. These descriptions are organized in Palladio in different views: component developers provide component specifications; software architects provide assembly models; system developers provide allocation models; and business domain experts provide usage models. We will present more details on Palladio and its visions in section 3.1.

2.4 Graph Transformation

Hundreds of computational problems are expressed in terms of graphs, and are often used to represent system states, diagrams and networks, such as flowcharts, entity-relationship diagrams, Petri nets, etc. Graph transformation, also known as graph rewriting or graph reduction, combines the potential and advantages of graphs with arithmetic, syntactic and deduction rules, for example, in a single computational paradigm (Andries et al., 1999).

The first graph transformation studies (such as Pfaltz and Rosenfeld, 1969 and Ehrig, Pfender, and Schneider, 1973) mainly dealt with the generation, manipulation, recognition and evaluation of graphs. Subsequently, different approaches related to the application domain, pattern recognition, semantics of programming languages, description of compilers, implementation of functional programming languages, specification of database systems, specification of abstract data types and also specification of distributed systems began to be introduced (Claus, Ehrig, and Rozenberg, 1979; Ehrig, Nagl, and Rozenberg, 1983; Habel, 1992; Plasmeijer, Van Eekelen, and Plasmeijer, 1993; Rozenberg, 1997).

Graph transformation consists of the iteration of rules in a graph, in which each rule application transforms one graph into another, replacing a part of it. For this, each rule (r) contains a left side L and a right side R. The application of a graph G replaces an occurrence of the left side L in G with the right side R. The transformation rules may contain application conditions for that this substitution of L for R is done in a controlled manner (Andries et al., 1999).

There are several graph transformation tools, such as Augur (König and Kozioura, 2005), GraphEd (Himsolt, 1990), PLEXUS (Wanke, 1990), PROGRES (Münch, 2000), AGG (Löwe and Beyer, 1993), GMTE (Hannachi et al., 2013) and PLANED (Franck, 1976), the latter being one of the first visual programming languages based on graph transformation. In this work we will use e-Motions (Rivera, Durán, and Vallecillo, 2009), which uses graph transformation rules to specify the behavior of a system and represent its possible actions. These rules are of the form

$$[\text{NAC}]^* \times \text{LHS} \rightarrow \text{RHS}$$

where NAC (negative application conditions), LHS (left-hand side) and RHS (right-hand side) are model patterns that represent certain (sub-)states of the system. The LHS and NAC patterns express the conditions for the rule to be applied (the definition of NAC patterns is optional), whereas the RHS represents the effect of the corresponding action if its conditions are satisfied. Thus, the action described in RHS can be applied. I.e., a rule can be triggered if a match of the LHS is found in the model and none of its NAC patterns occurs. An LHS may also have positive

conditions, which are expressed, as any expression in the rules, using the Object Constraint Language (OCL).

If several matches are found, one of them is non-deterministically chosen and applied, giving place to a new model where the matching objects are substituted by the corresponding instantiation of its RHS pattern. The transformation of the model proceeds by non-deterministically applying the rules on sub-models, until no further transformation rule is applicable. e-Motions combines different definitions to allow the specification, simulation and analysis of systems, which we will see in more detail in the section 4.5.1.

3 Palladio Tool for Building Self-Adaptive System Projects

Different approaches have been proposed for analysis of self-adaptation systems, including techniques and tools based on queue networks(Arcelli, 2020), Petri nets (Mian and Ahmad, 2017) stochastic models (Weyns and Iftikhar, 2016) Markov chain (Monshizadeh Naeen, Zeinali, and Toroghi Haghighat, 2020) and graph transformation (Bucchiarone et al., 2015).

In order to mitigate the difference in static model development in contrast to dynamism during the operations, Heinrich's iObserver project (Heinrich, 2016) proposes a metamodel that allows the reuse of development models during the operation phase. The approach aims to bridge the differences between the component-based design and its implementation, design and application configuration, and static and dynamic content. Even though the project recognises the importance of closer development models with the characteristics of dynamic systems, such as cloud systems, it does not use a dynamic model approach. Indeed, iObserver integrates design-time models, code generation, monitoring, analysis and run-time model update.

Grassi, Mirandola, and Randazzo, 2009 proposed D-Klaper as a tool for model-driven performance engineering that can be applied to self-adaptive systems. It uses an intermediate language to provide software design models, which can then be analysed. However, the D-Klaper language does not support the modeling of adaptation rules, nor the transformation of input models. The work by Falkner, Szabo, and Chiprianov, 2016 proposes an approach where the performance prediction is made at the beginning of the life cycle. To do this, workloads are modelled by estimating the resource consumption, capturing the CPU, memory and disk requirements. However, although they use models to represent the system, they are used to generate executable code for specific hardware and middleware deployments, and the results of the executions are presented to the expert through specific context views indicating whether the design meets the performance requirements, and cannot be considered a fully design time approach.

Abuseta and Swesi, 2015 propose an assistance tool for software adaptive-system developers in order to reduce development time and ensure robust functionality of the developed system. The framework is a reusable process of elements and system components to simulate and test the activities that exist in the self-adaptive systems, and aims at increasing the developers' confidence on the robustness of their designs prior to their deployment.

Becker, Becker, and Meyer, 2013 propose in SimuLizar a model-oriented approach to model self-adaptive systems and analyse the performance of its transitory phases. It is developed as an extension of Palladio for the performance analysis of self-adaptive systems at design time. The simulation scope is limited to only a set of rules that are triggered between the static environment models, which prevents it from testing multiple possible reachable states of systems at simulation time. SimuLizar was extended by Becker, Brataas, and Lehrig, 2017 to support scale up/down adaptations, specifically the change in the number of replicas and the computation capacity of a node. Krachand and Scheerer proposed a later extension SimuLizar NG (New Generation) (Krach and Scheerer, 2018) with the goal of facilitating domain-specific extensions, and allowing adaptations to the model interpretation semantics through the reactive simulation in demanding scalability scenarios. As part of the Descartes project, which also uses Palladio's PCM for their design time phases, Huber et al., 2012 propose a DSL to describe the behavior of self-adaptive systems based on strategies, tactics and actions. It focuses on runtime performance analysis, not on predictive analysis of applications at design time.

Of the existing tools, the one that presented us with the characteristics we need to build self-adaptive systems was Palladio, which enables modeling, predictive analysis and, like the Simulate approach, allows analysis of self-adaptive systems. More details on the tool will be presented in the next sections.

3.1 The Palladio Approach

Consolidated in both academia and industry, Palladio is an approach for the prediction of Quality of Service (QoS) properties of Component-Based Software Architectures and it can be used to model scenarios from different domains.

There are four important aspects that must be considered in Palladio's approach to academia: (1) its Component-Based Software Development process; (2) its detailed Component-Based Software Architectures metamodel, the PCM (Palladio Componente Model); (3) its software architecture simulator; and (4) its tool holder, the Palladio-Bench. With regard to its Component-Based Software Development process, the approach allows for contributions in modeling component-based software development scenarios. This modeling process uses the PCM metamodel, enabling the creation of Palladio models that can use its software architecture simulator to measure metrics that can be used for combined approaches such as performance analysis, reliability, ease of maintenance and cost forecasting . The implementation of all aspects mentioned above is supported by the Palladio-Bench tool, which is extensible and can serve as a basis for new approaches.

For this work, all aspects of Palladio were considered. We will detail in this section the PCM metamodel and its importance in the Component Based Software Development process, as well as the Palladio-Bench tool and how to create Palladio models that can use its simulator architecture for forecasting QoS properties.

3.1.1 Palladio Component Model (PCM)

The DSL used by Palladio is provided by its metamodel, the Palladio Component Model (PCM) (Becker, Koziolek, and Reussner, 2007), which is implemented using the Eclipse Modeling Framework (EMF). The software architecture is captured in

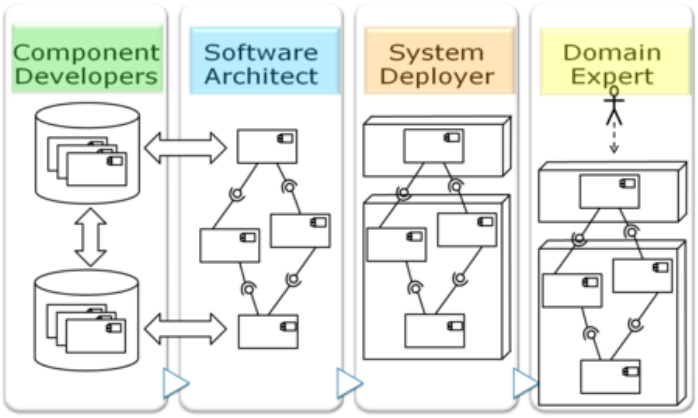


FIGURE 3.1: Artefacts of the Palladio Component Model (extracted from *Palladio Simulator Website*)

PCM through static structure. This means that once the models are defined, the simulation will take place considering this initial definition, and any changes have to be made by directly modifying the initially proposed model, without the possibility of verifying this change in simulation time. These models are related to behavior, deployment/allocation, resource environment/execution environment, and usage profile. Thus, the Palladio models are composed of four different artefacts (Figure 3.1), provided by corresponding developer roles involved in a CBSE development process (Becker, Koziolek, and Reussner, 2009a): component specifications (by component developers), assembly model (by software architects), allocation model (by system developers), and usage model (by business domain experts).

Component specifications. Component developers specify and implement parametric descriptions of components and their behavior. Components' services are described with Service EFFect specifications (SEFF), which abstractly model the externally visible behavior of a service with resource demands and calls to required services.

Assembly model. Software architects assemble components from the repository to build applications.

Allocation model. System deployers model the resource environment and the allocation of components from the assembly model to different resources of the resource environment.

Usage model. Domain experts specify a system's usage in terms of workload (i.e., the number of concurrent users), user behavior (i.e., the control flow of user system calls), and parameters (i.e., abstract characterisations of the parameter instances users utilise).

In addition to the description of the software in terms of components, the PCM uses connectors, interfaces, models of individual service behaviors (which are referred to as Service Effect Specifications - SEFF), containers, network, etc.

The PCM metamodel also allows the capture of elements that affect the non-functional properties (for example, performance and reliability) of a software system. However, these non-functional properties to be analyzed and the semantics of the PCM models are encapsulated in the respective transformations.

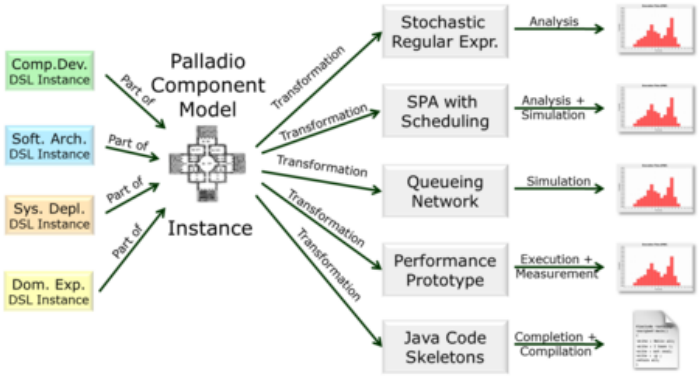


FIGURE 3.2: The Palladio-Bench (extracted from *Palladio Simulator Website*)

In the specification of the Palladio models, the parameters specified are fixed and cannot be changed along executions.

The PCM was developed with a high degree of parameterization, which makes its instances become widely reusable, that is, if, for example, the usage profile of the system changes, it is possible to adapt only the model of the usage profile and the other specifications can be reused. Thus, in a simulation, Palladio considers the impact of the new usage profile for analysis. However, the Palladio tool does not consider that these changes can happen at the time of the simulation due to changes in the context. This paved the way for our approach to propose that, from the use of the PCM, these parameters be changed in simulation time.

3.1.2 The Palladio-Bench

Palladio-Bench is an integrated, open source modeling environment, based on the Eclipse Rich Client platform. It was developed for developers to create instances of the PCM model with graphic editors and conduct analyzes using analytical techniques and simulations, making it possible, for example, to analyze the performance and reliability metrics of the PCM models (Figure 3.2).

Using Palladio-Bench it is possible to model PCM instances, simulate models, view simulation results and obtain software design optimizations. For this approach, Palladio-Bench PCM instance modeling is the most important feature and it is the one that will be used in this work. Figures 3.3, 3.4, 3.5, 3.6 and 3.7 are screenshots of examples of creation of Component Repository (for component specifications by component developers), Service Effect Specification (for modeling individual service behaviors), System Model (for assembly model modeling by software architects), Deployment Model (for allocation model modeling by system developers) and Usage Profile (for usage model modeling by business domain experts) made in the Palladio-Bench and extracted from the screenshots available on the page `https://www.palladio-simulator.com/tools/screencasts/`.

3.2 A running example

Component specifications. Component developers specify and implement parametric descriptions of components and their behavior. Fig. 3.8 shows the Palladio Component repository for this running example, with components LoadBalancer

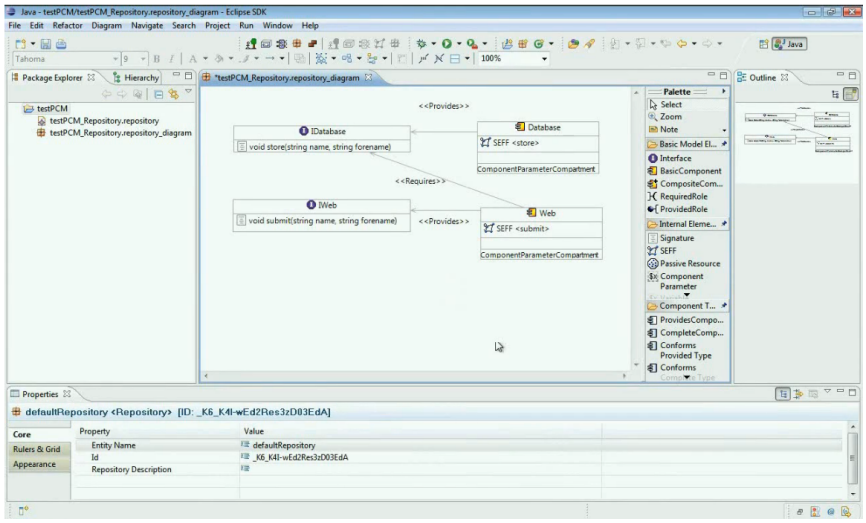


FIGURE 3.3: Creating a Component Repository (extracted from *Palladio Simulator Website*)

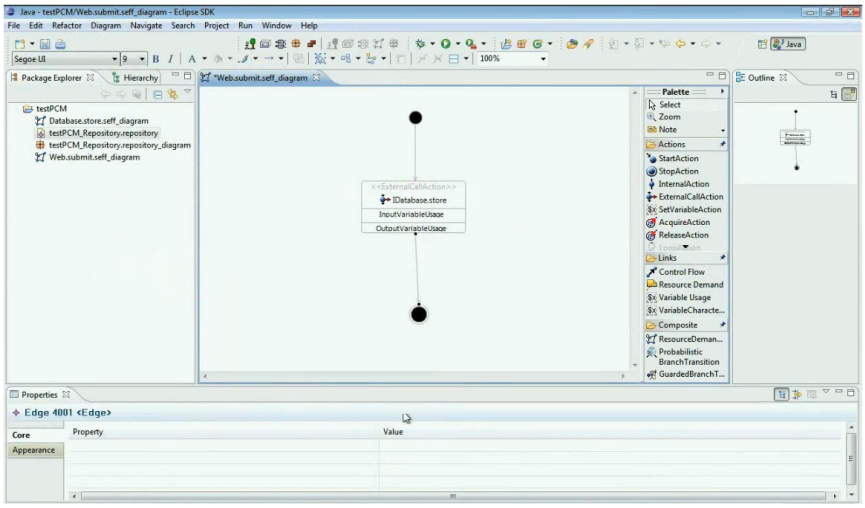


FIGURE 3.4: Creating a Service Effect Specification (extracted from *Palladio Simulator Website*)

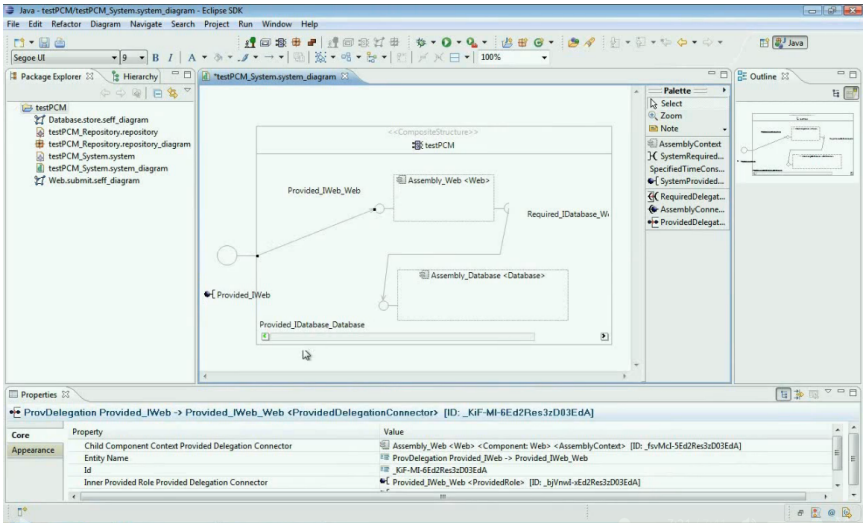


FIGURE 3.5: Creating a System Model (extracted from *Palladio Simulator Website*)

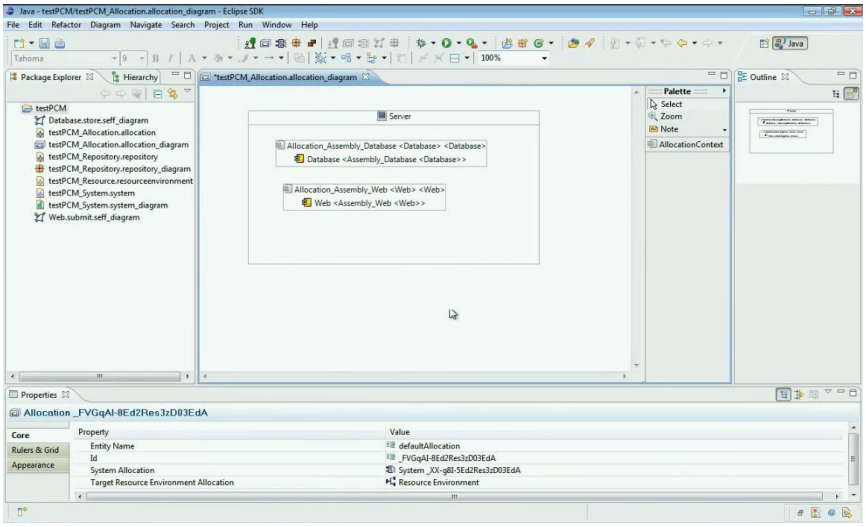


FIGURE 3.6: Creating a Deployment Model (extracted from *Palladio Simulator Website*)

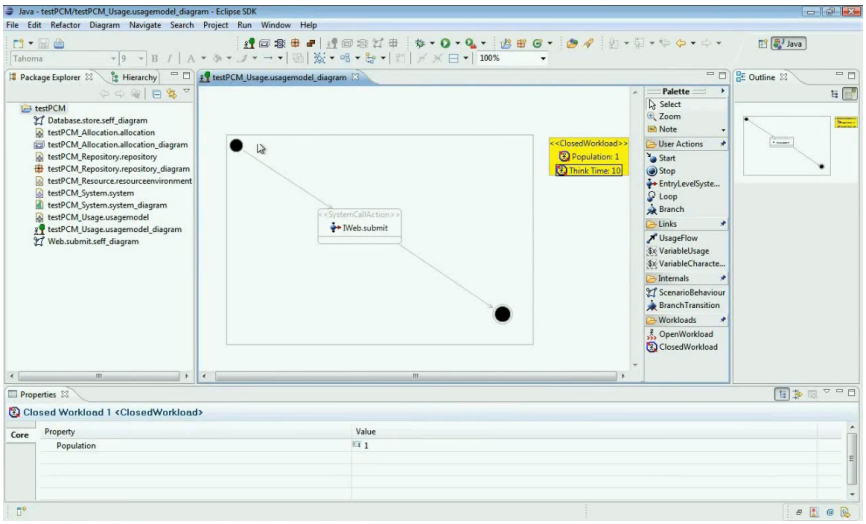


FIGURE 3.7: Creating a Usage Profile (extracted from *Palladio Simulator Website*)

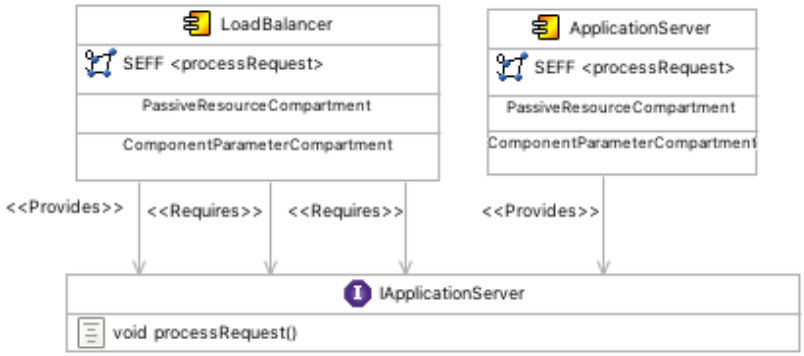
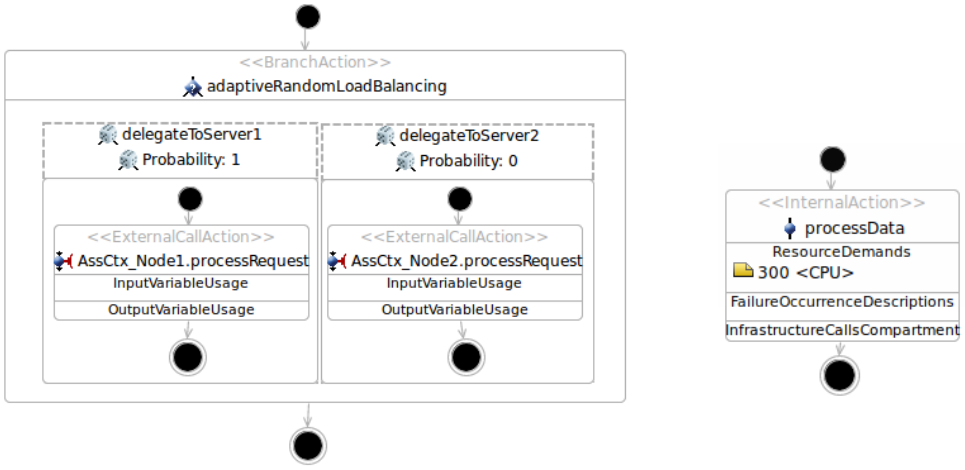


FIGURE 3.8: Components repository



(A) Load balancer (B) Server node

FIGURE 3.9: Components' SEFFs

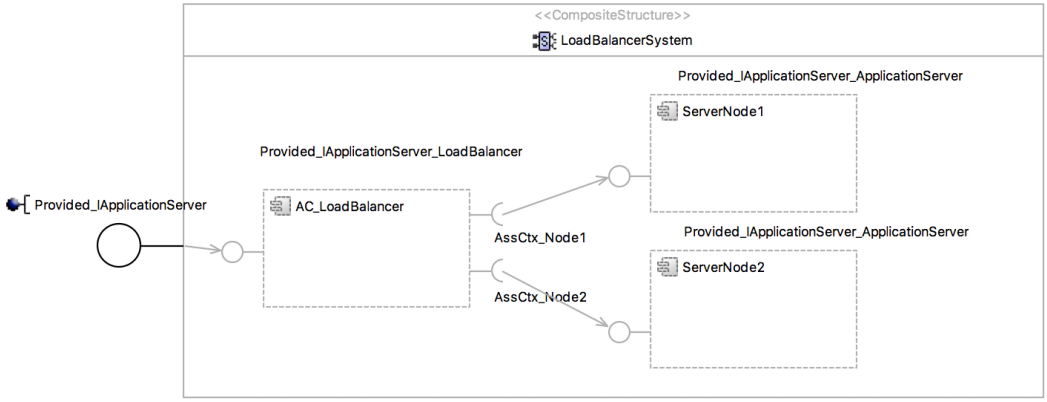


FIGURE 3.10: Assembly model

and ApplicationServer, and their dependencies. In it we can observe that there is an interface implemented by two different components. The first one represents the load balancer and the second one the component itself. Note that there are two Requires relations between LoadBalancer and IApplicationServer, which means that in the system model, the node containing such component (the front-end node) has to be connected to two nodes.

Components' services are described with service effect specifications (SEFF), which abstractly model the externally visible behavior of a service with resource demands and calls to required services. Fig. 3.9 shows the SEFFs of these components. Fig. 3.9a shows that the control flow in the LoadBalancer component may branch into one of two flows, each of them with an external call action to a different node. Each branch can be associated with a particular branch probability to indicate the likelihood of a particular branch being taken. This is the kind of information required to perform execution-time analysis on the component's behavior as is standard in software performance engineering (see, e.g., Smith and Williams, 2002).

Assembly model. Software architects assemble components from the repository to

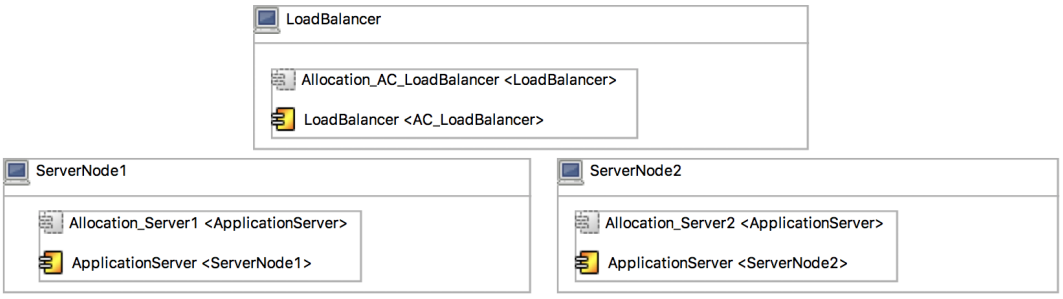


FIGURE 3.11: Allocation Model

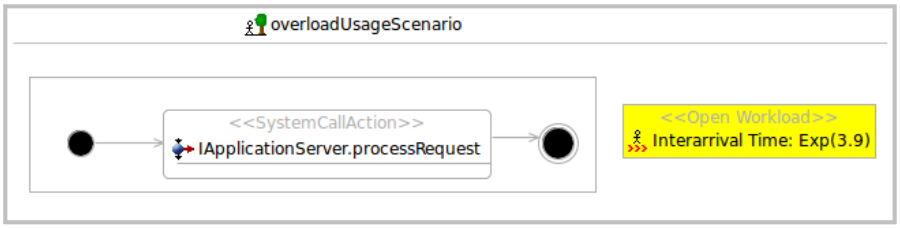


FIGURE 3.12: Usage Model

build applications. Fig. 3.10 shows how the components LoadBalancer and ApplicatonServer are composed. The biggest square surrounding the boxes represents the entire environment. For each provides relation in the repository model, a provided role is created for the container containing such component.

Allocation model. System deployers model the resource environment and the allocation of components from the assembly model to different resources of the resource environment. Fig. 3.11 shows the allocation model for our case study, where we can see how each of the components is allocated in a different node.

Usage model. Domain experts specify a system's usage in terms of workload (i.e., the number of concurrent users), user behavior (i.e., the control flow of user system calls), and parameters (i.e., abstract characterisations of the parameter instances users utilise). Given the usage model definition in Fig. 6.10, tasks will arrive following an exponential probability distribution with rate parameter 3.9 time units (Exp(3.9)), which means that tasks will arrive every ≈ 0.256 time units in average.

3.3 Simulizar

The Palladio tool allows analysis of static system models. In the running example of the previous section, it is only possible to verify the evolution of the systems, as shown in the graph in Figure 3.13, and it is not possible to apply changes to the model.

The analysis of this system with the Palladio Bench produces the graph in Fig. 3.13, producing a mean response time 4.0197 for 100 observations. Notice that with the workload used the system get overloaded, producing increasingly bigger response times as time passes. In the specification of the Palladio models, the parameters specified are fixed, and cannot be changed along executions. For instance, the arrival rate for work arrivals has been established in Exp(3.9), the demand of CPU for the

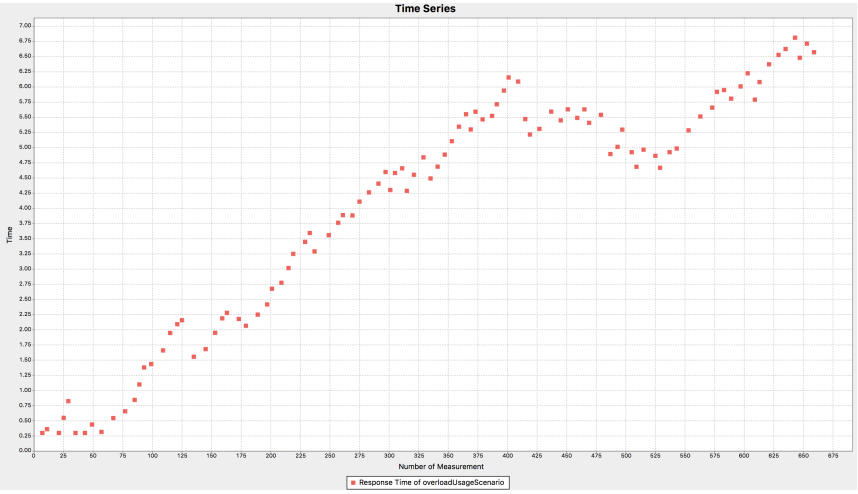


FIGURE 3.13: Response time analysis by Palladio

processing of the internal action in the servers is set to 300, and the number of CPU replicas in each server is 1.

The Simulate approach (Becker, Becker, and Meyer, 2013) opened the possibility of performing predictive analysis considering changes in the initial model (Figure 3.14). A model of a self-adaptive system, in Simulate, consists of two viewpoints with multiple views each. The first view is made up of three views: static view, monitoring specification view, and allocation view. Second view consists of initial state view and state transition view.

With Simulate PCM started to provide modeling views for monitoring view or state transition view. In this way, Palladio Measurement Specification (PMS) is used, which consists of a domain-specific language for viewing the monitoring specification, and self-adaptation rules have been used to specify the state transition view.

With PMS it is possible to specify "sensors" for a self-adaptive system, that is, it is possible to define where the monitoring will be carried out, the type of performance metric to be used, the type of time interval and a statistical characterization. However, you can only choose between the pre-defined Palladio-Simulize options, not being possible to insert new metrics, for example, in an easy and intuitive way.

Palladio-Simulate's self-adaptation rules consist of a self-adapting condition and action. A condition must reference a "sensor" and provide a boolean term, which if it evaluates to true, the self-adapting action is triggered. The self-adapting action part references elements in the PCM model.

Figure 3.15 illustrates the running example with the Simulate conditions and monitoring that gave rise to the results of Figure 3.14. You can see the PCM allocation visualization with measurement specifications, defined through the PMS. In the illustration, the LoadBalancer component is implemented on an lbn node and can vary its probability branch (which is initially set to 0.0, which means that all workload is initially handled by server node sn1), adding 0.1 while condition "MRT" > 0.8 is true.

Self-adaptive systems modeled with SimuLizar can be simulated using Palladio-SimuLizar's MDSPE for self-adaptive systems tool to obtain overall system response time predictions. In SimuLizar, reconfigurations are model transformations to the

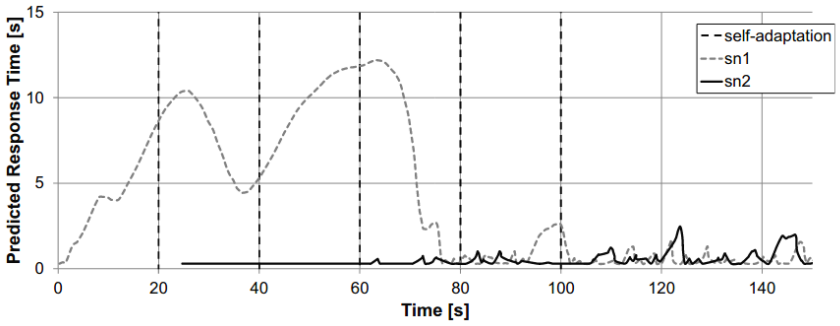


FIGURE 3.14: SimuLizar Predictions (From Becker, Becker, and Meyer, 2013)

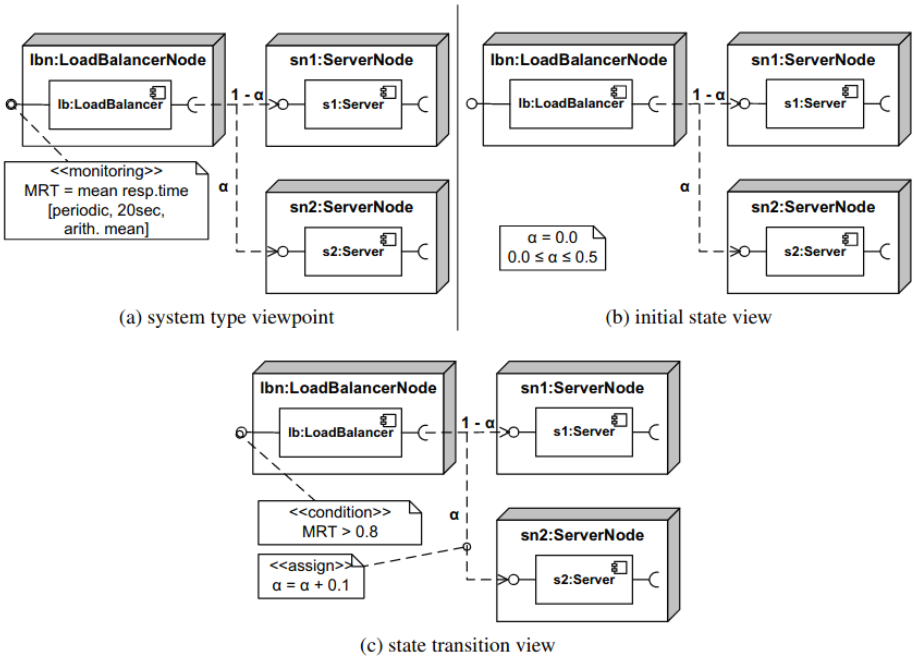


FIGURE 3.15: SimuLizar (a) System Type View,(b) Initial State View, and (c) Transition state view. (From Becker, Becker, and Meyer, 2013

PCM model, the managed element is the modeled simulated self-adaptive system. The simulated system is monitored, monitoring results are analyzed, reconfiguration is planned and a reconfiguration is performed on the simulated system if necessary.

3.4 Results

The Palladio tool is one of the most robust options for modeling systems. Its extension for predictive analysis of self-adaptive systems, Simulate, inherits Palladio's robustness, however, it has some limitations that prevent the tool from meeting the main characteristics of a self-adaptive system. Table 3.1 presents a Palladio-Simulate checklist regarding the characteristics of the system's general structure, allocation, functionality, communication, dynamic management and possible modifications. Although the solution meets or partially meets most of the characteristics, it is still necessary to advance with regard to the functionality and flexibility of modeling self-adaptive systems.

TABLE 3.1: Expected Characteristics for Modeling Self-Adaptive Systems

Features	**Palladio-Simulizar**
General system structure	meets
Allocation	meets
Functionality	does not meet
Communication	partially meets
Dynamism management	partially meets
Modifications	partially meets
Design Time	meets
Flexibility	does not meet

4 Flexibility in Modeling Self-Adaptive Systems

In this chapter we will present how to flexibly model a self-adaptive system, using the e-Motions tool, PCM modeled in e-Motions, also considering the advances in Palladio's behavior modeling obtained by this work.

4.1 The e-Motions Tool

The e-Motions tool (Rivera, Durán, and Vallecillo, 2009) is a graphical framework developed for Eclipse that supports the specification, simulation, and formal analysis of systems. It is an MDE tool since it provides a way to model domain-specific systems using both the abstract syntax metamodel and the concrete syntax metamodel, and it allows you to add rules to the metamodel by graph transformation, also making it possible to use OCL (Object Constraint Language).

It provides a way to graphically specify the Domain-Specific Languages (DSLs). The abstract syntax of a DSL is specified as an Ecore metamodel (Figure 4.1), which defines all relevant concepts and their relations in the language. Its concrete syntax is given by a Graphical Concrete Syntax (GCS) model (Figure 4.2), which attaches an image to each language concept. Then, its behavior is specified with (graphical) in-place model transformations (Figure 4.3).

The in-place model transformations used to specify the behavior of systems are defined by graph transformation rules, each of which represents a possible *action* of the system (Figure 4.4). A model of time can be defined for these rules, supporting features like duration, periodicity, etc., and mechanisms to state action properties (in the Figure 4.4 the rule spends one time unit on its execution and is triggered every ten time units, i.e , every eleven time units, a new peer is created).

From the definition of the abstract syntax, of the concrete syntax, and the specification of behavior by means of graph transformation rules, e-Motions generates an executable Maude (Clavel et al., 2007) specification which can be used for simulation (Figure 4.5). In this work we consider all of these aspects of e-Motions: the definition of the abstract syntax and of the concrete syntax, the specification of behaviors and the simulation capability of systems.

In simulation time several matches of rules can be found, and one of them is non-deterministically chosen and applied, giving place to a new model where the matching objects are substituted by the appropriate instantiation of its RHS pattern. The

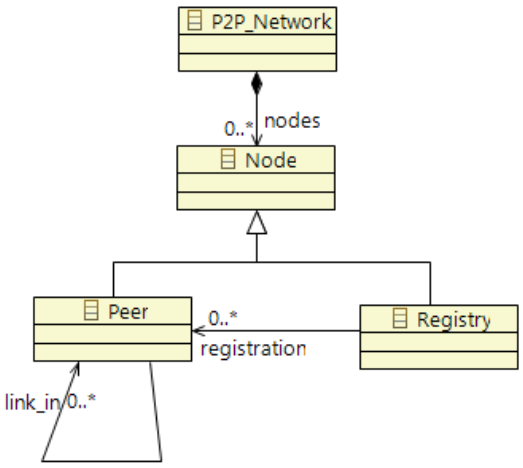


FIGURE 4.1: e-Motions Abstract Syntax Example: a Reconfigurable P2P Network (extracted from *e-Motions Examples*)

FIGURE 4.2: e-Motions Graphical Concrete Syntax Example: Peer and Registry Classes Picture of the P2P Network (extracted from *e-Motions Examples*)

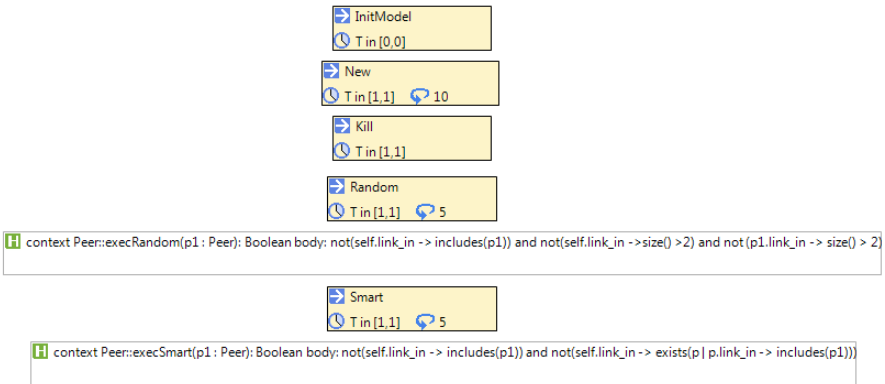


FIGURE 4.3: e-Motions behavior Rules Example: Five Rules and Two Helpers to Model the Behavior of the P2P Network System (extracted from *e-Motions Examples*)

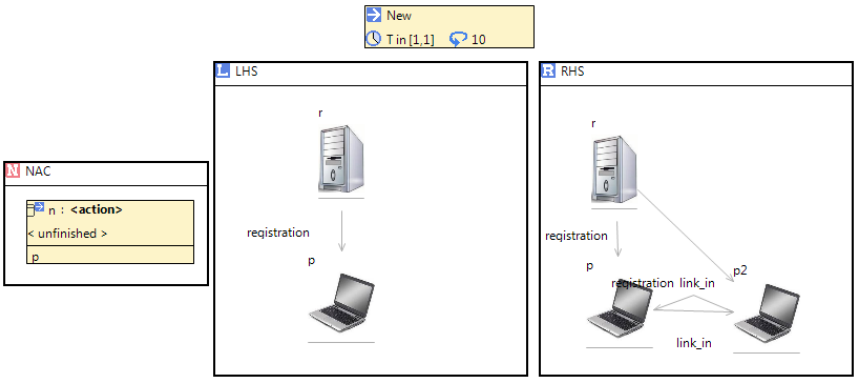


FIGURE 4.4: e-Motions Graph-Transformation Rules Example: The Rule Models the Creation of a new Peer in a P2P Network, Registers it, and Links it with an Existing Peer (extracted from *e-Motions Examples*)

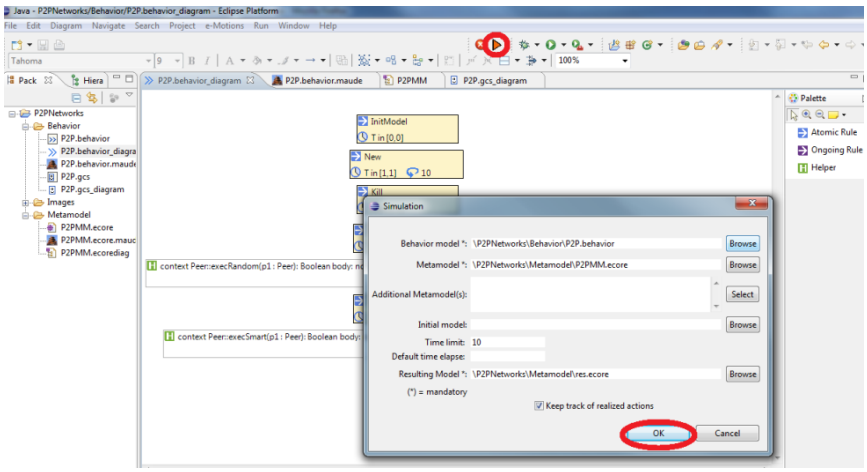


FIGURE 4.5: e-Motions Launcher Screenshot to Run a Simulation (extracted from *e-Motions Examples*)

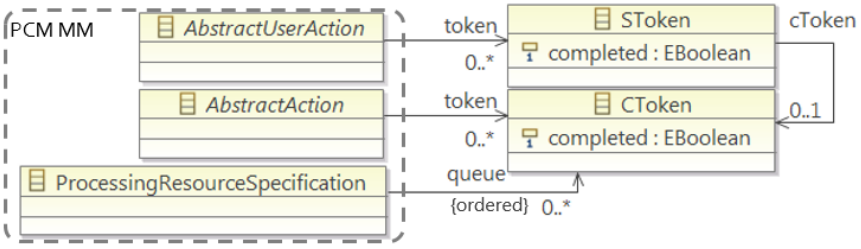


FIGURE 4.6: Palladio Abstract Syntax Defined in e-Motions Example: Token Metamodel (extracted from *e-Motions Examples*)

FIGURE 4.7: Concrete Syntax of the Palladio in e-Motions Example: Start Action and Token SEFF icons

transformation of the model proceeds by applying the rules on sub-models of it in a non-deterministic order, until no further transformation rule is applicable. This characteristic has opened the way for dynamically modifying models during simulation time, enabling a simulation of self-adaptive systems, which is the focus of this work.

4.2 Flexibilization as a New Path for Systems Modeling

The proposal to integrate the robustness of the PCM metamodel and the visual flexibility and ease of e-Motions was initiated in the work of Moreno-Delgado et al., 2014, which proposed a partial reimplementation based on a modular model of a Palladio Architecture structure. The work presented the specification of Palladio's main DSLs in the e-Motions system, describing the basic simulation semantics as a set of graph transformation rules. From this work it became possible that the models created in the Palladio-Bench can be fed directly into the e-Motions simulation environment for analysis. The Palladio DSL is provided by Palladio Component Model (PCM) and, as for any DSL, the e-Motions definition of Palladio includes its abstract syntax (the PCM) (Figure 4.6), its concrete syntax, and its behavior. Its concrete syntax uses the same images found in Palladio-Bench to represent Palladio models' concepts (there are some examples in the Figures 4.7 4.8, 4.9, 4.10, 4.11 and 4.12). Its behavior is defined by graph transformation rules, thus becoming explicit at a very high level of abstraction (the Figure 4.13 shows an example of the behavior of starting a SEFF of the Palladio in the behavior rule in e-Motions).

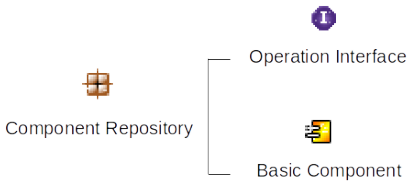


FIGURE 4.8: Concrete Syntax of the Palladio in e-Motions Example: Component Repository with Operation Interface and Basic Component icons

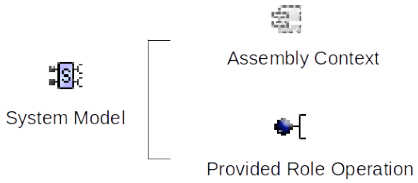


FIGURE 4.9: Concrete Syntax of the Palladio in e-Motions Example: System Model with Assembly Context and Provided Role Operation icons

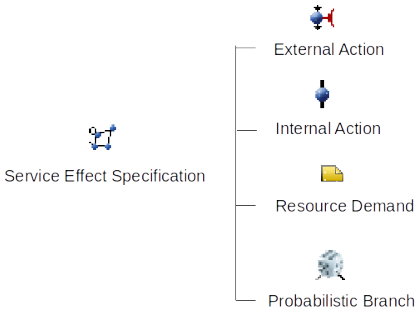


FIGURE 4.10: Concrete Syntax of the Palladio in e-Motions Example: Service Effect Specification with External Action, Internal Action, Resource Demand and Probabilistic Branch icons

FIGURE 4.11: Concrete Syntax of the Palladio in e-Motions Example: Deployment Model with Resource Container icon

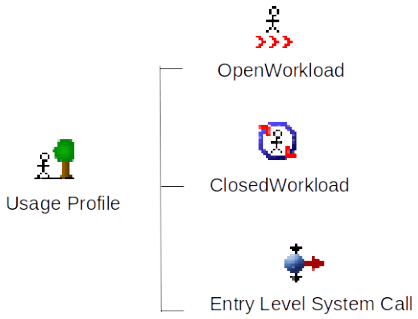


FIGURE 4.12: Concrete Syntax of the Palladio in e-Motions Example: Usage Profile with OpenWorkload, ClosedWorkload, and Entry Level System Call icons

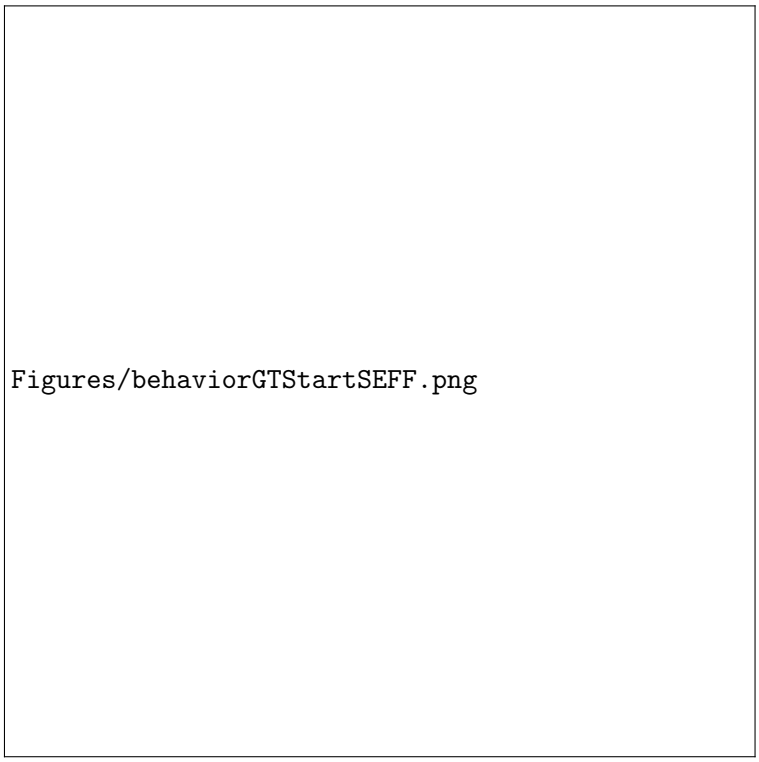


FIGURE 4.13: Start SEFF behavior is defined by graph transformation rules

The operational semantics of Palladio, i.e., its behavior, is given as a token-based execution model, where each work that enters the system is modelled as a token that moves around the different services of the system, and inside each service description, around the different tasks (start, stop, branch, loop, etc.) in its SEFF descriptions. Each of the actions that may occur in the system are then specified by e-Motions transformation rules.

The proposal of the modular model of the partial reimplementation of Palladio's architecture paved the way for the flexibilization of systems modeling. The proposal supports Palladio's main features for defining usage models (start, stop, delay and entry-level system call) and component models (start, stop, branch with any number of probabilistic branches, internal action and CPU specifications). In addition, the models created in Palladio can be fed directly into the simulation environment for analysis.

Based on the possibility of a flexible configuration for systems analysis, this work moves to a dynamic approach considering the partial reimplementation of Palladio in e-Motions. The central idea is to consider the possibility of adding and removing resources and components dynamically, opening a pathway for the modeling and analysis of self-adaptive systems. For this, it will be necessary to incorporate additional resources to the definition of Palladio as well as to understand the behavior of a self-adaptive system to facilitate modeling and analysis, thus enabling the experimentation of new features and customized solutions for specific problems at a very low cost of development.

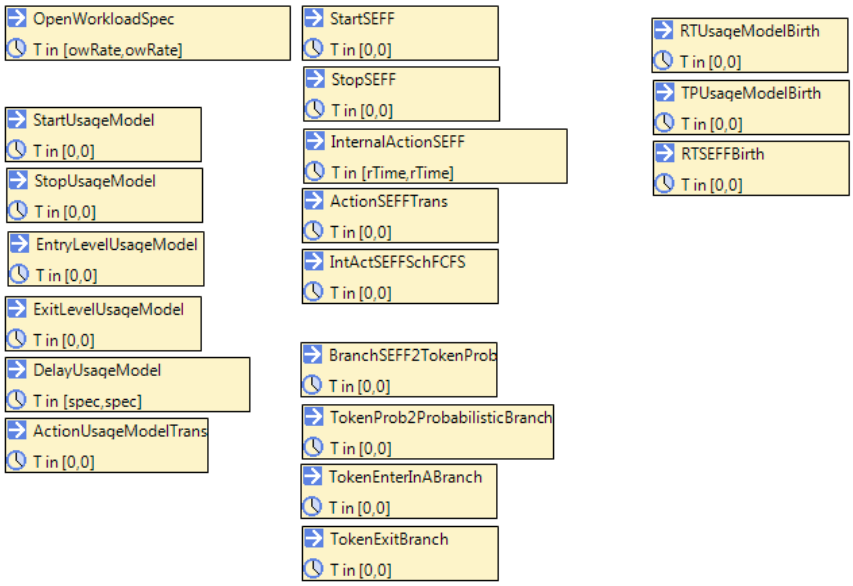


FIGURE 4.14: Palladio Rules by Moreno-Delgado et al., 2014 (extracted from *e-Motions Examples*)

4.3 PCM modeled in e-Motions

One of the main features of this approach is the use of explicit specifications of the behavior of Palladio. Since these specifications are modifiable, users have absolute control on the models and the operations available.

Figure 4.14 shows all behavior rules of Palladio provided by the work of Moreno-Delgado et al., 2014. The understanding of the Palladio metamodel, the rules of behavior already modeled and their flow were one of the first steps of this work. Figure 4.15 illustrates the activity diagram created in this work to facilitate the understanding and the visualization of the flow between the behavior rules initially modeled. Based on this understanding, it was possible to advance in this proposal, concerning the modeling of new rules or the redesign of the existing behavior rules of Palladio.

The modeled rules represent the behavior of the Usage Model and the Service Effect Specification (SEFF) of the Component Specification, which are central behaviors for the execution of a complete workflow, since only Palladio's Usage Model and Component Specification views represent system actions. Therefore, only these views were considered for modeling Palladio's behavior in e-Motions. The other views of Palladio are important for the modeling of the system and are represented in the abstract and concrete syntaxes in e-Motions and its elements are used in the definition of the Palladio's rules of behavior.

The rules in Figure 4.14 were reused in our work, modified when necessary (without changing the core of their operation) and other rules were inserted according to the needs of our proposal. The RTUsageModelBirth, TPUsageModelBirth and RTSEFFBith rules were separated from the behavior rules of Palladio and are now part of the set of QoS rules. In addition to the Palladio rules and the QoS rules, we also have the set of Adaptation rules and the set of Network rules, which interact with each other to compose the performance analysis.

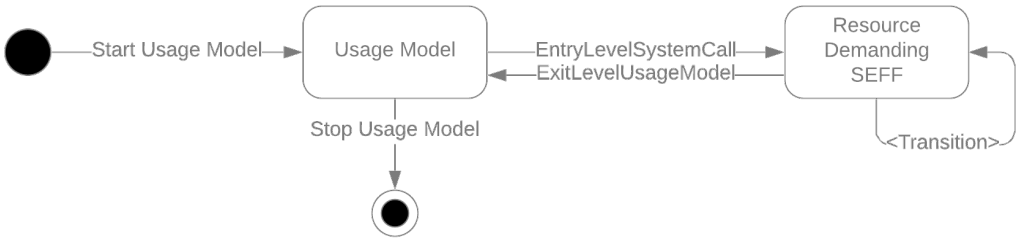


FIGURE 4.15: Initial Activity Diagram of the Palladio behavior

4.4 Advances in the Modeling Palladio Behavior

Palladio's abstract and concrete syntax was already available in e-Motions from the work of Moreno-Delgado et al., 2014, as well as some rules of behavior. This work has advanced in defining some rules already available and created others as needed.

The available rules were basically divided into Usage Model rules and Component Specification rules. It is worth mentioning that the rules in e-Motions that correspond to the features present in Palladio represent the behavior of these features if any of them are being used in a Palladio model used in e-Motions.

Figures 4.16 and 4.17 we present the progress made at in this work in relation to the creation of new rules (which is available at `https://www.scenic.uma.es/GTPAAS/`) and we illustrate an organization considering the aspects present in Palladio Bench (version 3.5).

Regarding the Usage Model rules, we separated according to the groups of features present in Palladio Bench 3.5: Actions, Actions Details and Workloads.

In Palladio's Actions we have some features that can be used in creating a usage model: Create New Start (StartUsageModel rule in e-Motions), Create New Stop (StopUsageModel rule in e-Motions), Create New Delay (DelayUsageModel rule in e-Motions), Create New Entry Level System Call (EntryLevelSystemCall and ExitLevelUsageModel rules in e-Motions), Create New Loop (LoopToLoopTokenUM, LoopStartTokenUM, LoopStop2StartUM and LoopStartTokenExitUM rules in e-Motions) and Create New Branch (BranchUM2TokenProb rule in e-Motions)). The ActionUsageModelTrans e-Motions rule (in *ActionTransitions) has no representation in Palladio but is inserted because a rule is needed to control the Tokens of transition between actions in the Usage Model.

Palladio's Actions Details are all the details of the functionalities inserted in the model. Create Branch Transition in Palladio is represented in e-Motions by the rules TokenEnterInABranchUM and TokenExitBranchUM. The creation of the rule(s) for the use of Variable in the Usage Model is still pending.

Workloads in Palladio are represented so far in e-Motions by the rules for OpenWorkload (OpenWorkloadSpec, DecreaseCountDown and RemoveTokenOb). The creation of the rules that represent ClosedWorkload is still pending.

Regarding the Component Specification rules, we separated according to the Service Effect Specification feature groups present in the Palladio Bench 3.5: Actions and Actions Details.

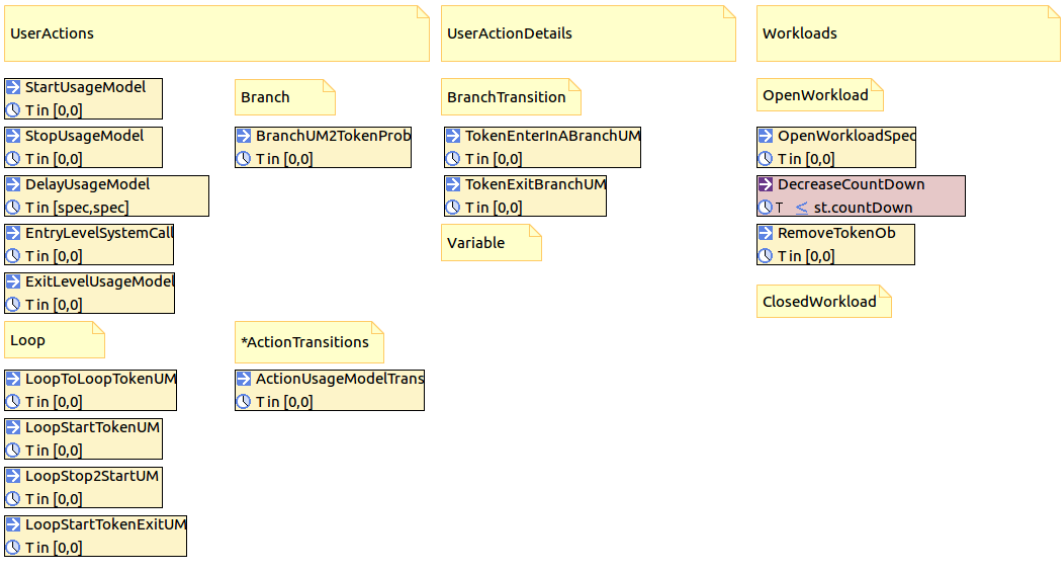


FIGURE 4.16: Palladio Usage Model Rules to Support Dynamism

In Palladio's Actions we have some features that can be used in creating a SEFF: Create New Start (StartSEFF rule in e-Motions), Create New Stop (StopSEFF rule in e-Motions), Create New Internal Action (InternalActionSEFF and IntActSEFFSchFCFS rules in e-Motions), Create New External Call Action (EnterInExternalCallAction and ExitOfAndExternalCallAction rules in e-Motions), Create a Set Variable Action (SetVariableActionSEFF and ReturnVarExitCallAction rules), Create New Loop Action (LoopToLoopTokenSEFF, LoopStartTokenSEFF, LoopStop2StartSEFF e LoopStartTokenExistSEFF in e-Motions), Create New Branch Action (TokenEnterInABranchSEFF and TokenExitFromBranchSEFF rules in e-Motions).

As with the Usage Model rules, the ActionSEFFTrans e-Motions rule (in *ActionTransitions) has no representation in Palladio but is inserted because a rule is needed to control the token transition between actions in the Component Specification SEFF. In addition, it was necessary to insert a rule for behavior regarding the change of containers after an External Action (ContainerChange in * Communcation) and some rules to control the transition of Tokens in the External and Internal Actions, and in the Branch (TransitionToEA, TransitionToIA and TransitionBranch).

The creation of the rules that represent the SEFF functionalities is still pending: Emit Event Action, Acquired Action, Release Action, Collection Iterator Action, Fork Action and Recover Action. So far these features have not been necessary for our work.

The SEFF Actions Details inserted were: Create New Probalistic Branch Transition (BranchSEFF2TokenProb and TokenProb2ProbalisticBranch rules in e-Motions) and Create New Guarded Branch Transition (EnterInBranchGuarded, GuardedBranch2TokenGuarded, TokenGuarded2Guardedbranch, TokenEnterInABranchGuarded and TokenExitFromBranchGuarded).

4.4.1 Changes to the existing Definition of Palladio Rules in e-Motions

Let us illustrate the e-Motions rules by showing two of the rules that define Palladio's behavior. The first one, in Figure 4.18, is the OpenWorkloadSpec rule, which models the action of initiating a new execution. In it, we see how a usage scenario

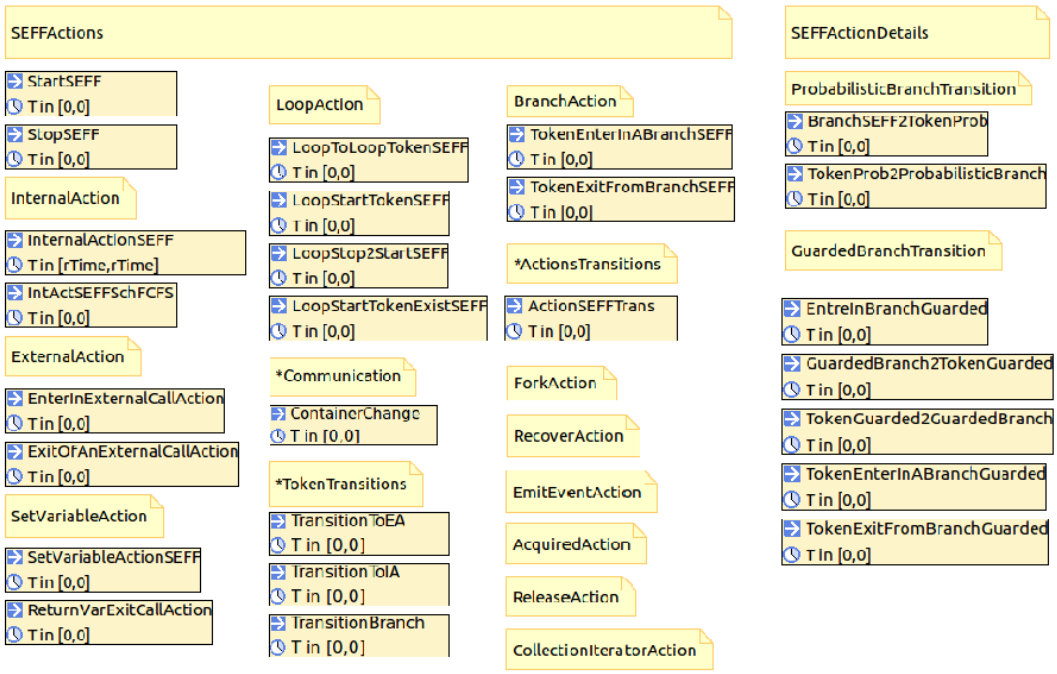


FIGURE 4.17: Palladio Component Specification Rules to Support Dynamism

(usSc) is linked to a scenario behavior (scBeh) that defines the sequence of actions to perform and a specification of the workload description. In this case, the usage scenario is described by an open workload (ow), which has a timer associated. The rule models the arrival of a new work into the system. Specifically, when the timer comes to zero, a new token is associated to the start action of the scenario behavior. The timer is set to the amount specified by the corresponding stochastic expression — e.g., Exp(5.0) in our running example. Note the use of the owRate variable, which temporarily holds the result of the evaluation of the given expression.

This rule is an example of one of the modifications made to the existing Palladio rules (Figure 4.19). The first modification was the insertion of a Sand Timer in detriment to the definition of the maximum and minimum duration (T in [owRate, owRate]) in the atomic rule, since the arrival time defined in the model concerns the time interval between the arrival of works and not the time it takes to get the work done. At intervals of time this had no impact on the execution of the simulations, however it was modified to avoid problems in other scenarios. Instead of the time definition in the atomic rule, an Ongoing rule "DecreaseCountDown" was created to account for the passage of time, it has the sole function of counting down the time arrived value (defined in the model) that was stored in the Sand Timer object.

Another modification made to the rule in Figure 4.18 in relation to the rule in Figure 4.19 was the removal of the observer's creation of the Time Stamp Token object. This is because observers are treated more generally and are now linked to SYBL monitors.

Figure 4.20 shows the e-Motions rule that specifies the execution of an InternalAction, like the ones used in Figures 6.7b and 6.7d. This rule represents a generic execution of an internal activity by a component service, possibly using a resource, like a HDD

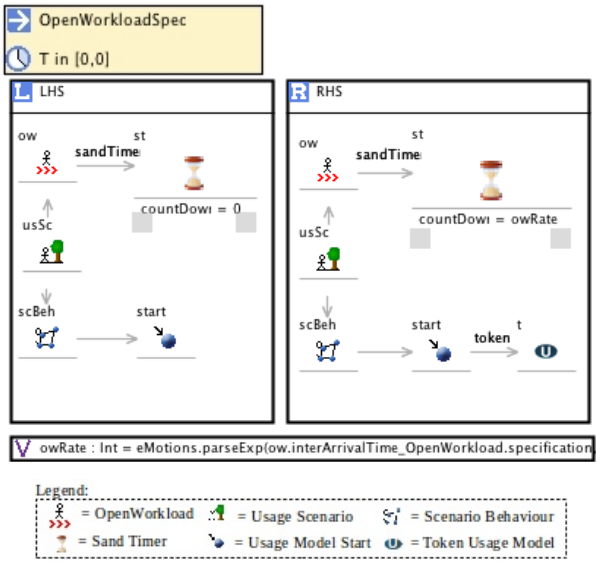


FIGURE 4.18: OpenWorkloadSpec rule

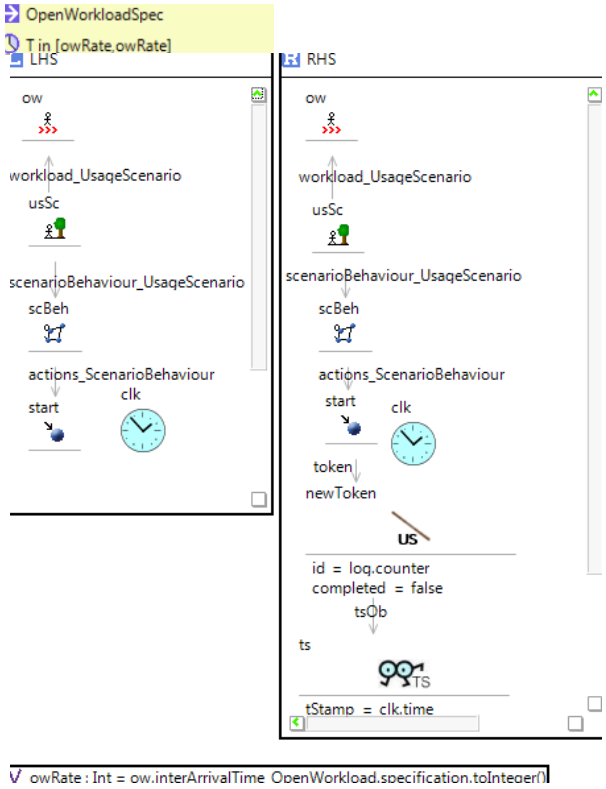


FIGURE 4.19: OpenWorkloadSpec rule Old

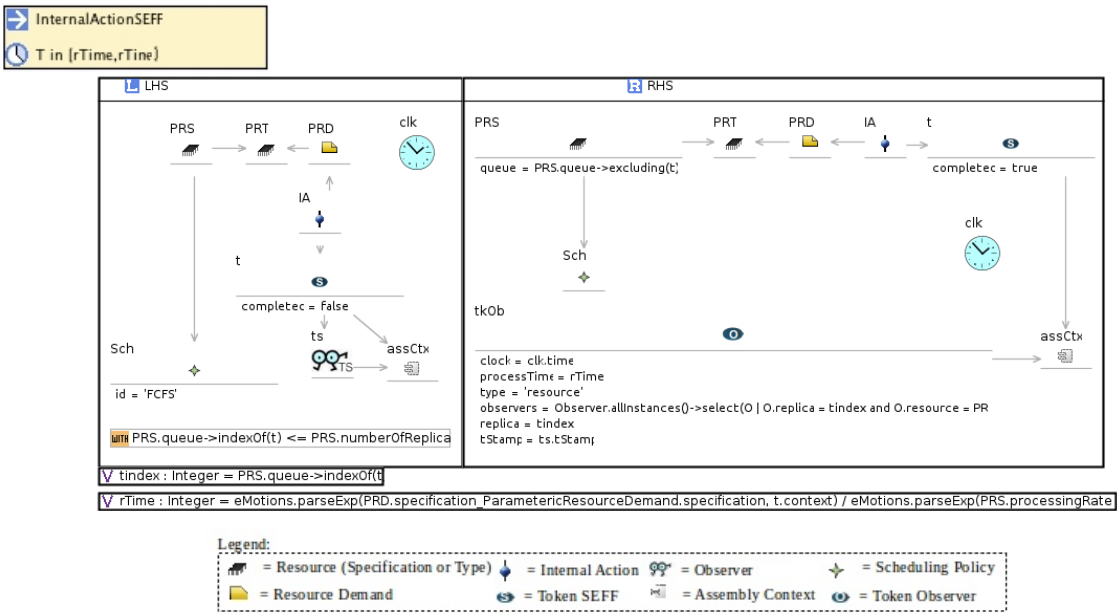


FIGURE 4.20: InternalActionSEFF rule

or CPU, following an FCFS (First Come First Served) strategy. In Palladio, these executions present a high level of abstraction, and the resource demands are described as stochastic expressions. In the e-Motions rule, the LHS indicates that if there is an internal action (IA) in the system linked to a token (t) with the completed attribute with value false, the RHS will execute in time rTime, calculated using the expression of the header of the rule (PRD / PRS), i.e., the duration of this action depends on the corresponding Palladio elements, specifically on the PRD (Parametric Resource Demand) and on the PRS (Processing Resource Specification). For instance, in our example the IA was specified with PRD of 300 units of CPU (Figure 6.7d) — the resource type is defined in the object PRT (Processing Resource Type). The container in which the component that specifies this IA was allocated presents the processing rate (PRS) of 1000 units of CPU per time unit. Thus, each work will take 0.3 time units to be executed. Tokens are served following an FCFS strategy by using a queue associated to each resource type. Only the first PRS.numberOfReplicas tokens in the queue PRS.queue get to be executed. Once an internal action is executed, its token is removed from the queue (PRS.queue→excluding(t)), and marked as completed, being then 'moved' to the following task in the service description. Please, note the indication of the assembly context, and the recording of the execution information by the Observer Token object (tkOb), which will be used by the monitors.

Previous versions of this rule explicitly included the checks and actions associated to required changes in the system. In our new approach, however, monitors gather all the information on the metrics of interest, which will then be handled independently by corresponding observers. In this way, new observers may be added at any time without needing to modify these rules. With this modification, generic tokens were created to store the values obtained in the simulation and subsequently update the observers. This creates greater flexibility for the inclusion of new metrics for measuring non-functional properties, as it eliminates the need to manipulate the rules specifying the behavior of Palladio to change the requirements or adaptation mechanisms of specific systems. In summary, observer objects are in charge of collecting information on non-functional properties with which the performance analysis will

be carried out by monitors.

4.4.2 Creation of new Palladio rules in e-Motions

The new Usage Model rules present in e-Motions are: LoopToLoopTokenUM, LoopStartTokenUM, LoopStop2StartUM, LoopStartTokenExitUM and BranchUM2TokenProb (in Actions); TokenEnterinABranchUM and TokenExitBranchUM (in Actions Details); and, DecreaseCountDown and RemoveTokenOb (in Workloads).

LoopStartTokenUM, LoopStartTokenUM, LoopStop2StartUM and LoopStartTokenExitUM relate to the set of behaviors required for the Loop action of a workflow. BranchUM2TokenProb concerns identification of a Branch in a usage model, and TokenEnterinABranchUM and TokenExitBranchUM concern the actions of entering and leaving a Branch, respectively. DecreaseCountDown is an Ongoing rule created to help control the time flow of users established in the model (previously this control was in charge of the OpenWorkloadSpec rule). Finally, the RemoveTokenOb rule was created so that Tokens created from created observers were deleted after the completion of each work.

The new Component Specification rules present in e-Motions, are: EnterInExternalCallAction, ExitOfAnExternalCallAction, SetVariableActionSEFF, ReturnVarExitCallAction, LoopToLoopTokenSEFF, LoopStartTokenSEFF, LoopStop2StartSEFF, LoopStartTokenExistSEFF, ContainerChange, TransitionToEA, TransitionToIA and TransitionBranch (in Actions). EnterInBranchGuarded, GuardedBranch2TokenGuarded, TokenGuarded2Guardedbranch, TokenEnterInABranchGuarded and TokenExitFromBranchGuarded (in Actions Details).

EnterInExternalCallAction and ExitOfAnExternalCallAction are rules that establish the behavior of the External Call Action, which is an important action of Palladio, as it is responsible for the calls of other components. In defining this behavior, we established that the components to be called can be located in the same node or in a different node, and in this last option, the network that connects the nodes can be considered or not. To make this possible, the ContainerChange rule was created, which considers the information established in Communication Link Resource Specification, such as latency and throughput, in Palladio's Resource Environment.

In addition, the TransitionToEA, TransitionToIA and TransitionBranch rules were modeled, which have the role of controlling the transition between External and Internal Actions and within a Branch. SetVariableActionSEFF and ReturnVarExitCallAction are the rules responsible for the action details variable usage that can be defined in a SEFF. The LoopStartTokenSEFF, LoopStop2StartSEFF and LoopStartTokenExistSEFF rules are related to the set of behaviors necessary for the Loop action of a SEFF. Finally, EnterInBranchGuarded, GuardedBranch2TokenGuarded, TokenGuarded2Guardedbranch, TokenEnterInABranchGuarded and TokenExitFromBranchGuarded correspond to the set of actions necessary to establish the behavior of a Guaded Branch action in SEFF.

4.4.3 Artefacts and Processes Proposal

With the objective of extending Palladio's predictive capabilities to support dynamic systems, we propose to use the e-Motions implementation of the Palladio Component Model (PCM) described in Moreno-Delgado et al., 2014, and use the graph rewriting approach used there to also model the elastic behavior of systems. Given

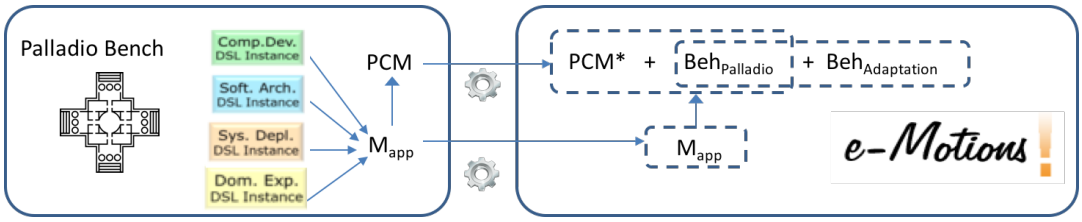


FIGURE 4.21: Artefacts and Processes Proposal

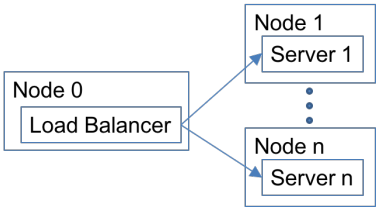


FIGURE 4.22: Structure of the example

the metamodel of Palladio and its operational semantics expressed in terms of graph-transformation rules, this implementation allows to analyze (static) systems modeled in Palladio Bench. However, the relevance of this e-Motions specification is the capability of integrating new adaptation rules, which extend the behavior of Palladio, making feasible the analysis of dynamic systems, and increasing its expressiveness. The facilities for the analysis of different metrics, including response time, resource usage and throuhtput, and the facilities for the extension of the language, present an optimal setting for the definition of the adaptation mechanisms and the analysis of the performance of elastic systems thus defined.

The diagram in Figure 4.21 depicts the main elements in our proposal. In e-Motions, a Domain Specific Language (DSL) is specified by its syntax (a metamodel) and a behavior (an operational semantics described as a set of graph transformation rules). The systems to be analyzed are specified using Palladio, and specifically the Palladio Component Model (PCM). Models conforming to the PCM are composed of four different submodels, which correspond to respective views of the system. The Palladio language is specified in e-Motions by taking an extended PCM, denoted PCM^*, which includes definitions for tokens and dynamics of systems, and its behavior. As presented in Moreno-Delgado et al., 2014, static systems defined in the Palladio Bench (models M_{app} conforming to the PCM) can be loaded and analyzed in e-Motions using the DSL $PCM^* + Beh_{Palladio}$. In this work, to deal with adaptive systems, the behavior to be used is extended with adaptation mechanisms, specified as additional e-Motions transformation rules, $Beh_{Adaptation}$. Then, a specific model M_{app} can be used to analyze the performance of the described elastic system using the DSL $PCM^* + (Beh_{Palladio} + Beh_{Adaptation})$.

4.5 A running example

To illustrate the Palladio views, we present a very simple scenario with a single server and a load balancer structure (depicted in Figure 4.22). Then, we will use our adaptation rules to add and remove nodes when necessary from this initial scenario.

Figure 4.23a shows the component repository for our example. It depicts two components and their corresponding interfaces: ApplicationServer implements IApplicationServer and LoadBalancer implements ILoadBalancer. There is one Requires relations from the LoadBalancer component to the IApplicationServer interface, that offers the processRequest() operation. As we can see in the assembly model in Figure 4.24a, the front-end node, containing the LoadBalancer component, will invoke the processRequest() operation provided by the IApplicationServer interface. Components' services are described by service effect specifications (SEFF), which abstractly model the externally visible behavior of a service with resource demands and calls to required services. Figure 4.23c shows the SEFF of the processRequest service, which models the behavior of the server component. Such processing is very simple in our example, it just consists in an internal action that consumes 300 units of CPU (CPU cycles). Figure 4.23b shows the SEFF of the balancer() operation, which models the control flow in the LoadBalancer component as a probabilistic branching. In our example, the system starts with one server, we will see in the coming sections how the dynamic extensions is in charge of adding and removing nodes as needed. Since there is only one branch in this initial Palladio definition, the probabilistic branching initially has one single branch, with probability 1, which has an external call action to the single node of the model. As new nodes are added to the architecture, new branches will be added to this action, thus modeling the distribution of works between the existing servers handled by the load balancer.

Software architects assemble components from the repository to build applications, represented by assembly models in Palladio. Figure 4.24a shows how the services of the LoadBalancer and ApplicationServer components are composed. The biggest square surrounding the boxes represents the entire environment. For each 'provides' relation in the repository model (Figure 4.23a), a provided role is created for the container containing such component.

Allocation models are provided by system deployers, who model the resource environment and the allocation of components from the assembly model to different resources of the resource environment. Figure 4.24b shows the allocation model for our initial model, where we can see how each of the components is allocated in a different node.

Finally, usage models are provided by domain experts, who specify a system's usage in terms of workload, user behavior, and parameters. Given the usage model definition in Figure 6.10, in our case study, tasks will arrive following an exponential probability distribution with rate parameter 4.9 time units (Exp(4.9)).

4.5.1 Palladio Specification in the e-Motions System

The e-Motions system Rivera, Durán, and Vallecillo, 2009 is a graphical framework that supports the specification, simulation, and formal analysis of real-time systems. It provides a way to graphically specify the dynamic behavior of DSLs using their concrete syntax, making this task very intuitive. The abstract syntax of a DSL is specified as an Ecore metamodel, which defines all relevant concepts and their relations in the language. Its concrete syntax is given by a GCS (Graphical Concrete Syntax) model, which attaches an image to each language concept. Then, its behavior is specified with (graphical) in-place model transformations.

The e-Motions language provides a model of time, supporting features like duration, periodicity, etc., and mechanisms to state action properties. From a DSL definition, the e-Motions tool generates an executable Maude Clavel et al., 2007 specification which can be used for simulation and analysis. For instance, we can perform reachability analysis, model checking, and statistical model checking of the DSLs defined using e-Motions (see Rivera, Durán, and Vallecillo, 2009 and Durán, Moreno-Delgado, and Álvarez-Palomo, 2016).

The in-place model transformations used to specify the behavior of systems are defined by rules, each of which represents a possible *action* of the system. These rules are of the form $[\text{NAC}]^* \times \text{LHS} \rightarrow \text{RHS}$, where LHS (left-hand side), NAC (negative application conditions) and RHS (right-hand side) are model patterns that represent certain (sub-)states of the system. The LHS and NAC patterns express the conditions for the rule to be applied, whereas the RHS represents the effect of the corresponding action if its conditions are satisfied. Thus, the action described in RHS can be applied, i.e., a rule can be triggered, if a match of the LHS is found in the model and none of its NAC patterns occurs. An LHS may also have positive conditions, which are expressed, as any expression in the RHS, using OCL (Object Constraint Language). If several matches are found, one of them is non-deterministically chosen and applied, giving place to a new model where the matching objects are substituted by the appropriate instantiation of its RHS pattern. The transformation of the model proceeds by applying the rules on sub-models of it in a non-deterministic order, until no further transformation rule is applicable.

Palladio is a DSL, and has been specified in Moreno-Delgado et al., 2014 using the visual facilities of the e-Motions system Rivera, Durán, and Vallecillo, 2009. As for any DSL, the e-Motions definition of Palladio includes its abstract syntax (the PCM), its concrete syntax, and its behavior. Its concrete syntax is provided in e-Motions by a GCS model in which each concept in the abstract syntax being defined is linked to an image. These images are used to graphically represent Palladio models in e-Motions, which uses the same images that the PCM Bench to represent these concepts. Its behavior is defined by graph transformation rules, thus becoming explicit at a very high level of abstraction.

The operational semantics of Palladio, i.e., its behavior, is given as a token-based execution model, where each work that enters the system is modeled as a token that moves around the different services of the system, and inside each service description, around the different tasks (start, stop, branch, loop, etc.) in its SEFF descriptions. Each of the actions that may occur in the system are then specified by e-Motions transformation rules. For example, Figure 4.25 shows the e-Motions rule that specifies the execution of an InternalAction, like the one shown in Figure 4.23c. This rule represents a generic execution of an internal activity by a component service, possibly using some resources, like HDD or CPU. In Palladio, these executions present a high-level of abstraction, and the resource demands are expressed as stochastic expressions. In the e-Motions rule, the LHS indicates that if there is an internal action not completed in the system, the RHS will execute in time rTime. The duration of this action depends on the corresponding Palladio elements, specifically on the Parameter Resources Demanded (PRD) and on the Processing Resource Specified (PRS). For example, a PRS may have an initial specification of 300 work units per second (PRS.processingRate) and 1 CPU replica (PRS.numberOfReplicas). Tokens are served following an FCFS (First Come First Served) strategy by using a queue associated to each resource type. Only the first PRS.numberOfReplicas tokens in the

queue PRT.queue get to be executed. Once an internal action is executed, its token is removed from the queue (PRT.queue→excluding(t)), and marked as completed, being then 'moved' to the following task in the service description.

The behavior of Palladio's core features has been specified by time-aware in-place transformation rules, corresponding to the possible model changes. Once the whole DSL has been defined, and given a model as initial state, it may be simulated by applying the rules describing its behavior. However, this model does not collect information on non-functional properties (NFPs), and therefore is not ready for performance analysis. For this, an observer mechanism Troya et al., 2013 is used to measure the non-functional properties of each of the components in the system. The PCM metamodel is extended with a family of observer classes, and their semantics is defined by appropriate definitions in the rules defining the behavior of Palladio. In the rule of the Figure 4.23c, the rus object collects information about PRS resource at run-time, which can then be used for quality informations both during the simulation or for post-simulation analysis. Since the PCM is used as metamodel in the e-Motions definition of Palladio, models developed using the Palladio Bench can be directly loaded into the e-Motions tool. The complete e-Motions definition of the Palladio DSL is available at `http://atenea.lcc.uma.es/e-Motions`.

4.5.2 Adaptation Rules in e-Motions

In our approach, the different adaptation mechanisms are defined as transformation rules on the model of the system under analysis. Thus, given monitoring information on the different metrics under observation, systems may adapt in different ways by performing different operations, like scale up/down (increase/decrease of the amount of resources like computation capacity, memory, etc.), scale in/out (adding/removing computation or storage nodes), etc. As usual, the monitoring of metrics is stored in the observer objects, which allows us to consult current values, windows of values of certain length, and complete histories of data. To illustrate how our approach works, we focus on one of the most challenging of these operations: the scale out associated to a load balancer as the one in our example, scaling out when the average usage of CPU in the last time window goes over 65%.

Adaptation rules operate on the structure defined in the Palladio models, which are indeed used to provide the initial models for the simulations. The addition of a node in any state of the system (see Figure 6.3) will imply the modification of the models of the different views in the Palladio description:

- In the component repository, when the addition of a node occurs, a Requires relation between the LoadBalancer component and the IApplicationServer interface should be created.
- In the balancer SEFF, which models the control flow in the LoadBalancer component, a branch (with appropriate likelihood and corresponding external call action) should be created.
- An assembly context should be created along with its communication with load balance for each server added.
- The allocation model also is modified at runtime: for each node added, an allocation of the ApplicationServer component should be added to this node.

All these actions, including balancing actions, are carried out by e-Motions rules in Figures 4.26–4.30. Specifically, they model the addition of a new node, with the same characteristics of the existing component in the system.

Since the Palladio models the e-Motions tool operates on, represent entire system states, we can specify system adaptations in exactly the same way we model their evolution.

The condition that triggers the application of a rule is provided by its LHS and conditions, and its effect by its RHS (the pattern in the LHS is replaced with the pattern in the RHS with the corresponding substitution).

The Add Node Rule (Figure 4.26) is the main one firing the addition of the node. It is in charge of creating action tokens that will fire and guide the execution of the subsequent rules. The Add Node Rule is triggered when the attribute usPerc of the observer ob indicates that the current value of the average resource usage in the time window is over 65%. The RHS of the rules specifies that the action to perform to react to such an event consist in the creation of a new node nNode in the environment recEnv; in addition, a link is established with the computing lan center linkRes, a token indicating the creation of a new node tNnode and a token of the resource specification spec. After this rule, and fired by the reception of these tokens, four additional rules are in charge of configuring and inserting the node in the existing context.

Figure 4.27 shows the rule that creates the resource specification of the new node, which is triggered when there is a token resource specification spec linked to a node object nNode, the resource type rType and the scheduling policy sPolicy (both values in the token specification spec) is in the model. In such a situation, this rule creates the indicated specification for the new rule, removing the token specification spec when the action is performed.

Figure 4.28 shows the establishment of the context of the new node. The action is triggered when there is a new node token tNnode linked with a node object nNode. The rule uses the context of the component to allocate the new node, associated to the signature sign, Resource Demanding SEFF seffRD, Application Server reBasComp (Basic Component), Operation Provided Role prov and Operation Interface reOpInterface. When applied, the rule allocates the component to the new node, creating a new assembly context for it.

Figures 4.29 and 4.30 are executed to conclude the process, creating a connection with the Load Balancer component lbResBasComp and a new probabilistic branch transition to the new node seffProbBranchTrans, respectively.

4.6 Results

As for static systems, we can now carry on performance analysis of system designs. Specifically, we discuss here the analysis of the resource usage, response time and throughput of the example specified using Palladio. This system is taken as input initial model, on which adaptation transformation rules operate, together with the rest of rules defining the operational semantics of the system. As a result we can observe how the system behaves and evolves, what will allow us to study different parameters, and take informed decisions on the best configurations and parameters for it before its deployment.

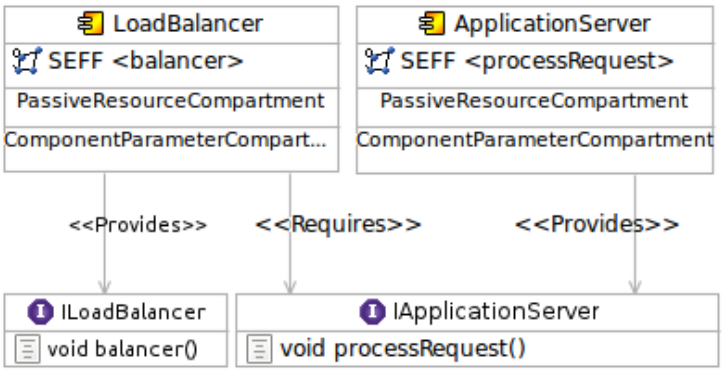
To improve the presentation of the example, we have limited the possible adaptations that operate on the example to scale up and down depending on the usage of CPU. To allow a more stable evolution of the system, we have considered a time

window of 10 units and a minimum time to repair (TBA) of 5 time units. Figure 4.31 shows evolution of the average CPU usage and the number of servers. The CPU usage of each of the servers is also shown. In the graph we can observe how the system starts with one servers (initial model provided in the Palladio specification), and although the average CPU usage is 100% (the rate of incoming works clearly overrates the capacity of the only server), a second server is not created until time 5 (the application of the first scale-out adaptation occurs at time 5.07). We can observe an immediate drop in the average resource usage after the creation of this second server. However, it is not quick enough, after 5 time units a new adaptation occurs, and after 5 more a third adaptation leaves the system with 4 servers (at times 10.19 and 15.27). This configuration with 4 servers goes on until time 68.32, where the average CPU usage goes below 35%, the threshold for a scale in. The works in the queues produce a new scale out some time later, and one more scale in after some more time. After this last adaptation the average CPU usage of the servers stays within the established thresholds and the system stays with 3 servers.

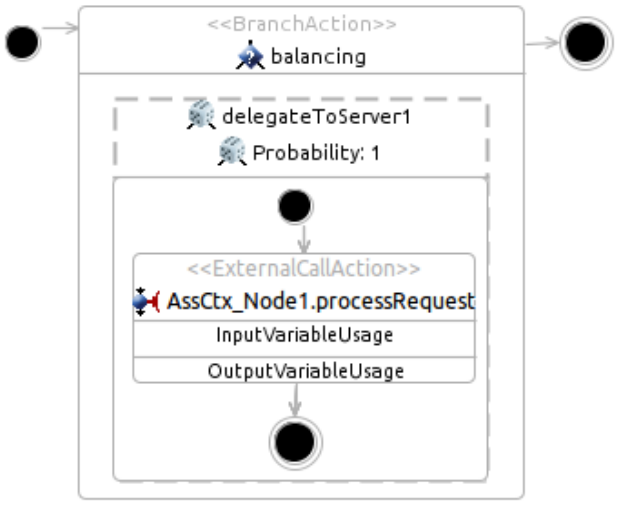
The scale in operation is quite similar to the scale out one described in Section 4.5.2. In this case, the node is removed, the references to it are removed, and the probabilities of the remaining nodes are redistributed. However, although no further works are submitted to a node being removed, it is kept in operation until all the works in its execution queue are processed. This explains the non-abrupt ending lines for nodes being destroyed.

The chart in the Figure 4.32 shows the throughput (defined as the number of works processes per time unit) and response time values compared with the adaptations that have occurred over time. We can see the increase in the throughput as the number of servers increases. The impact on the response time can be noticed from time 10. Notice however that after the first server is added, in the time 5.07, there are some works with response times above 4 time units and others with response time below 1 time unit. This of course happens because the works are distributed between different servers, and the server that has just been added can handle new works inmediately until it gets saturated. The response time stabilizes after some time, staying below 2 time units after time 15.

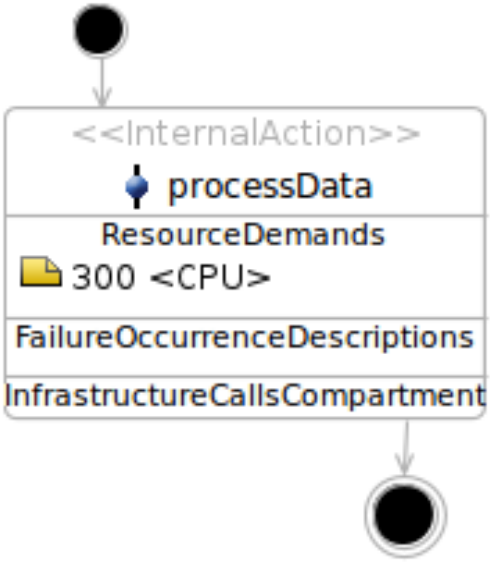
Many things can be learnt from these graphs. For instance, for such a workload we may very well start with three servers from the begining, since it seems to be its stabilization point for the given parameters. Using a different size for the time window may avoid the initial fluctuations, although it is not necessary the case. Figures 4.33 and 4.34 show the charts for time window 10 and minimum time between adaptations 5.



(A) Components repository

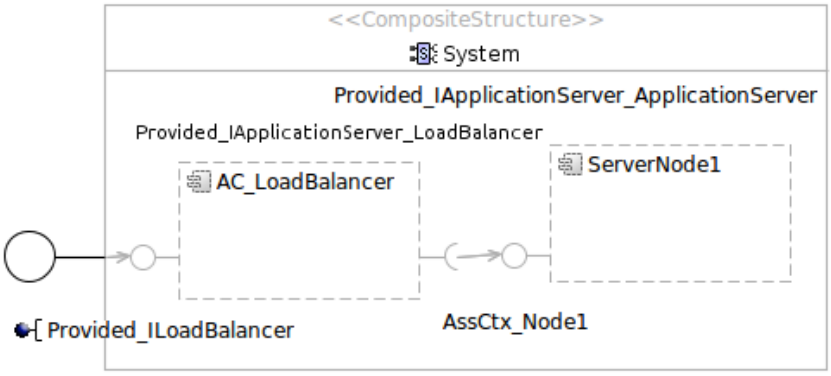


(B) Balancer SEFF

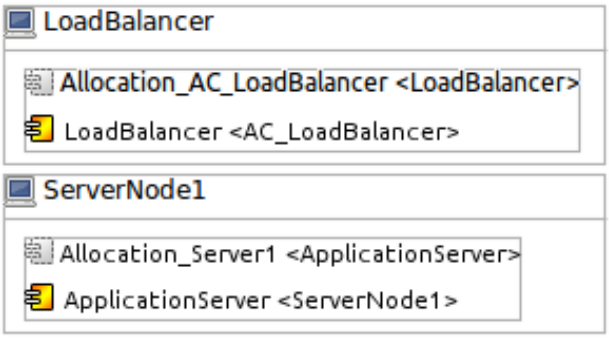


(C) Request SEFF

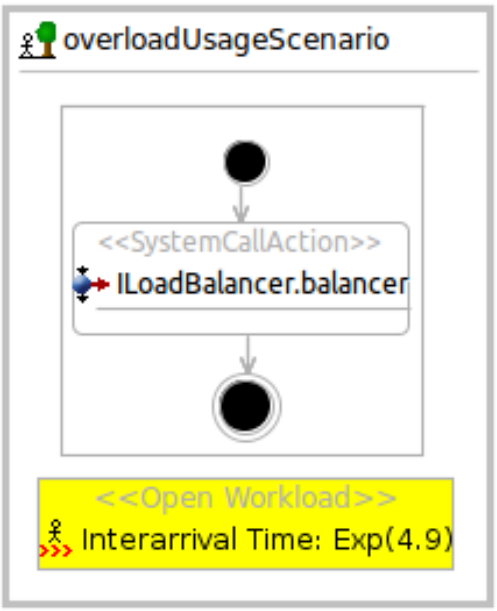
FIGURE 4.23: Component Model



(A) Assembly Model



(B) Allocation Model



(C) Usage Model

FIGURE 4.24: Allocation, Usage and Assembly Models

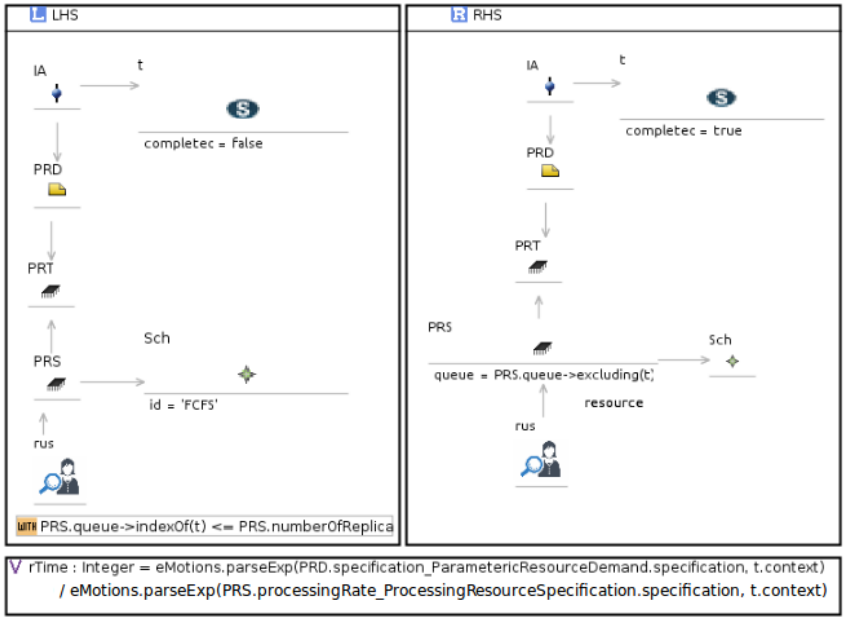


FIGURE 4.25: Internal Action SEFF rule

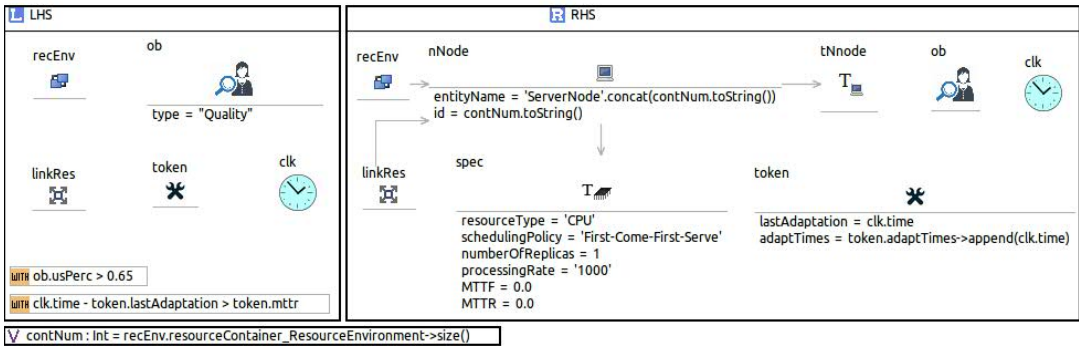


FIGURE 4.26: Add Node Rule

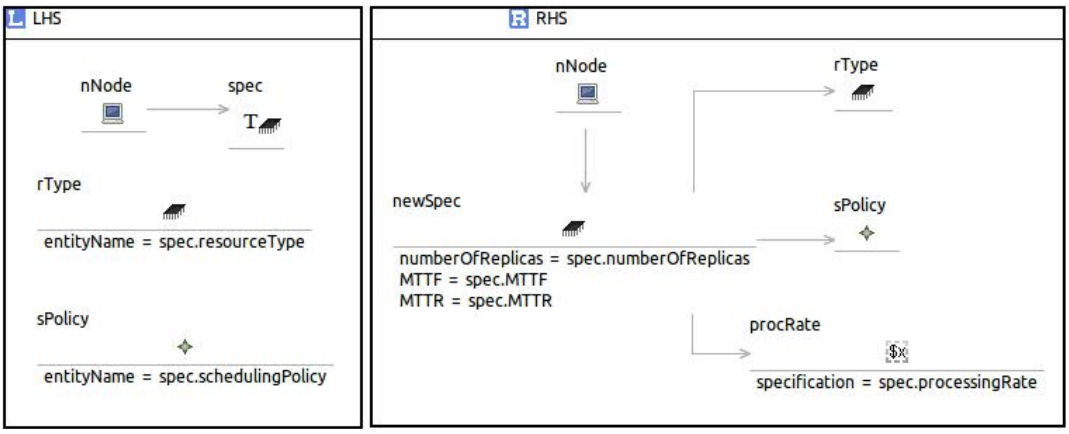


FIGURE 4.27: Resource Specification New Node Rule

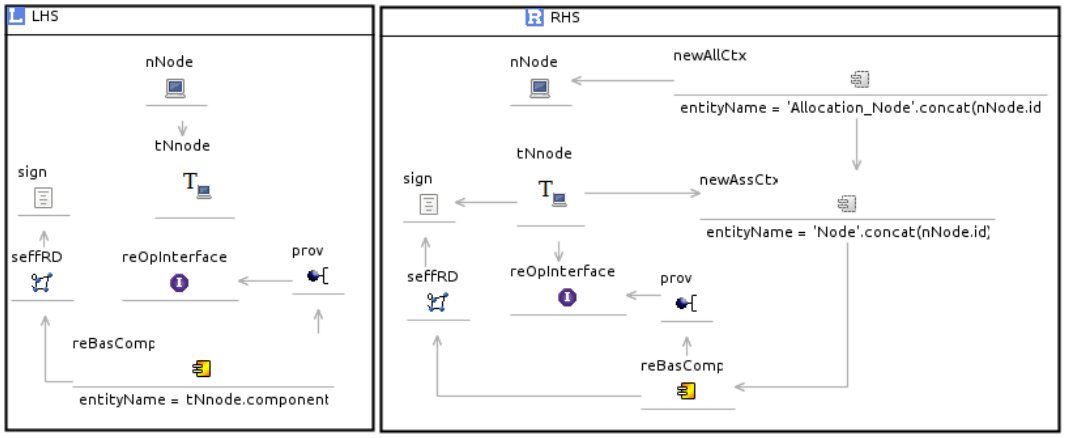


FIGURE 4.28: Context New Node Rule

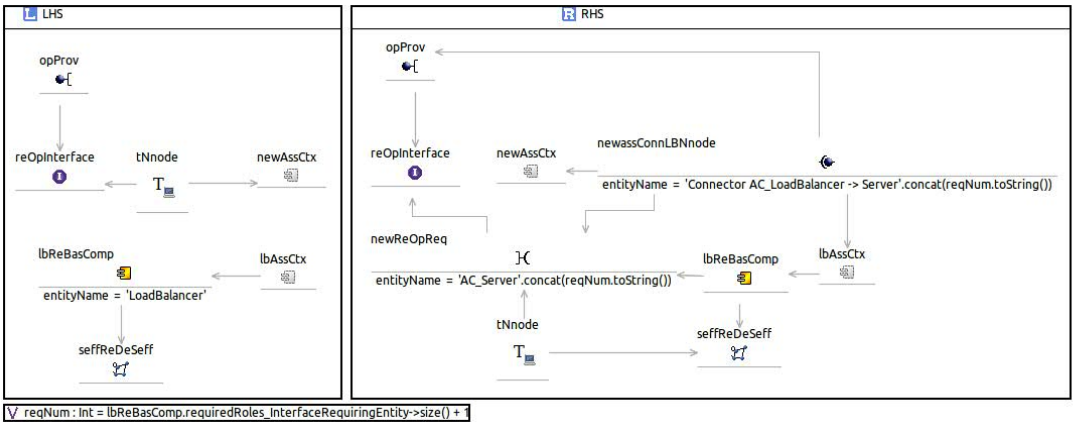


FIGURE 4.29: Load Balancer and New Node Connection Rule

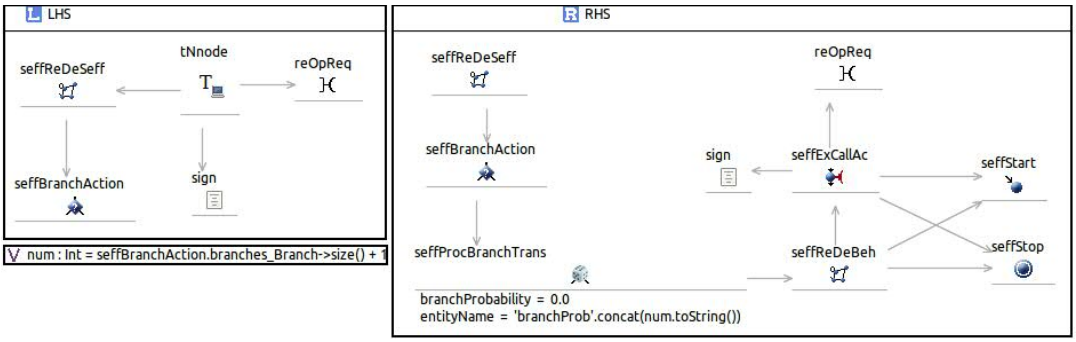


FIGURE 4.30: Add New Branch Rule

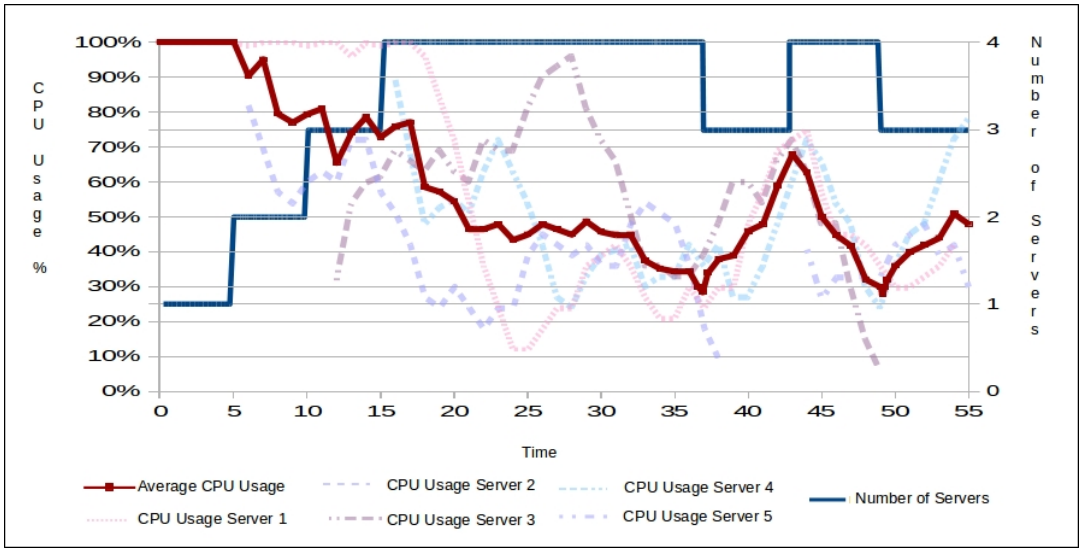


FIGURE 4.31: CPU usage correlation with the number of servers scaled: TW 5, TBA 5

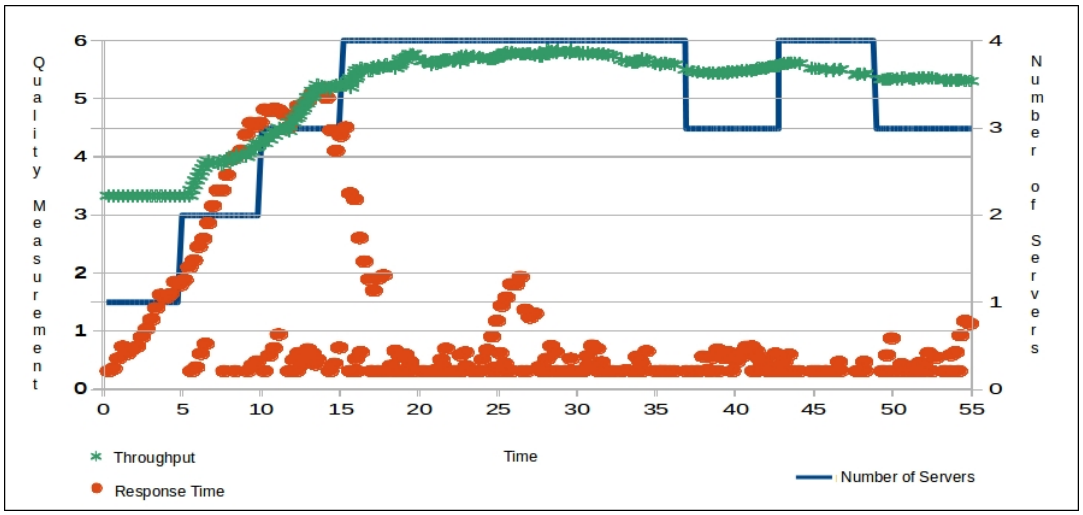


FIGURE 4.32: Throughput and response time: TW 5, TBA 5

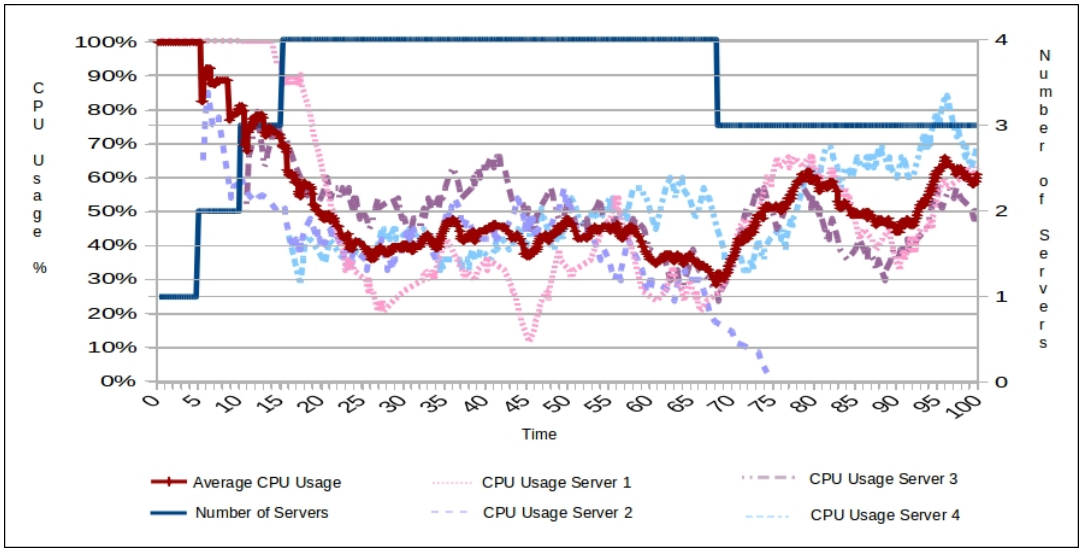


FIGURE 4.33: CPU usage correlation with the number of servers scaled: TW 10, TBA 5

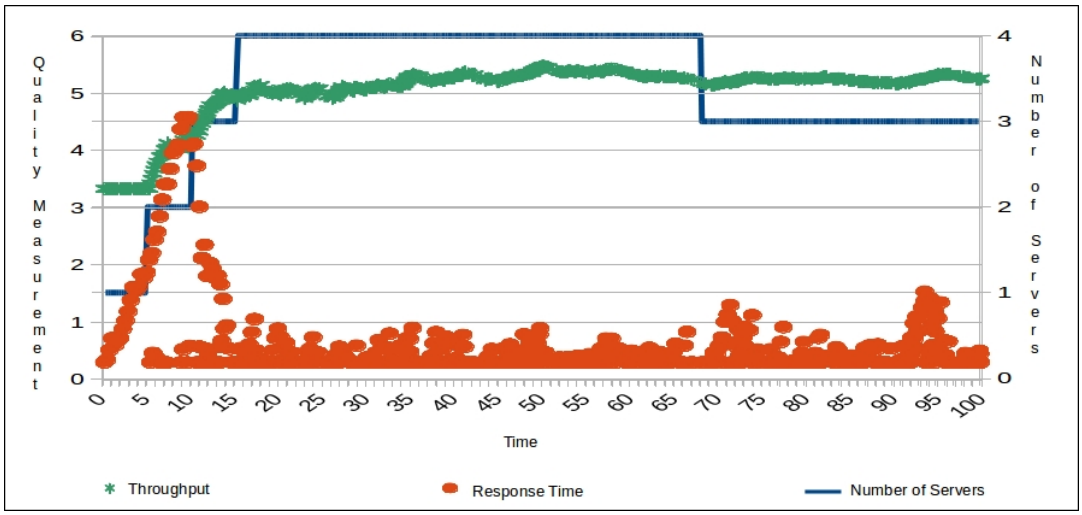


FIGURE 4.34: Quality metrics impact: TW 10, TBA 5

5 Building Adaptation Mechanisms

The definition of dynamic behavior in this work occurs through the advancement in modeling Palladio's behavior; in the modeling of adaptation mechanisms; and in the modeling of non-functional requirements control. All of these steps were built using the e-Motions tool. In this section we will present the details of the modeling of each one of them. In e-Motions we unite everything in a single metamodel (advancing in the existing Palladio metamodels) and separate the rules of behavior in: Palladio rules (advancing in the existing Palladio rules), QoS rules, adaptation rules and network rules.

5.1 Modeling of Adaptation Mechanisms

In this section, we describe how the adaptation rules can be modelled[1]. Although other adaptation operations could be similarly specified, we focus here on what in Cloud Computing is known as vertical and horizontal scaling. Vertical scaling is the ability of resizing a server to increase or decrease its processing/storage capacity by adding or removing the amount of resources provided. Vertical scaling is limited by the amount of resources available. Horizontal scaling is the ability of a system to change resource capability by adding or removing nodes (i.e., instances or virtual machines), enabling the use of new resources, perhaps with better and larger processing/storage capacity, or releasing them if they are not further necessary.

The linkage between the rules specifying the adaptation operations and the QoS and SYBL rules of behavior happens thanks to the SYBL and the SYBL Annotation classes created. In adaptation rules, the adaptation Strategy is represented by objects of these classes.

Figures 5.1 and 5.2 show, respectively, sketches of the scale-up and scale-out adaptations. Figure 5.1 illustrates a scale up operation on the number of replicas of the CPU resource available in the Server Node. Given n replicas, its number can be increased in any amount m. The increment is specified as one of the adaptation parameters. A scale out consists on adding nodes, thus creating new containers and their resource specification and allocating a component on them. Figure 5.2 sketches the effect of a scale out operation on our running example. Given some number of DataBase nodes in some state of the system, each with a corresponding number of CPU resource

[1] The rules is available at https://www.scenic.uma.es/GTPAAS/.

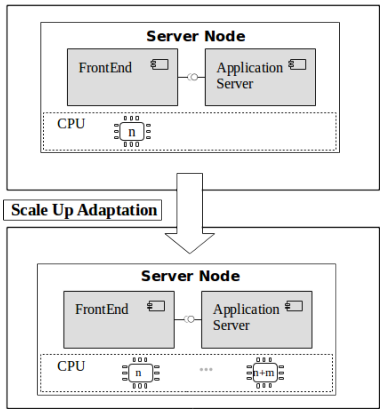


FIGURE 5.1: Scale up adaptation scheme

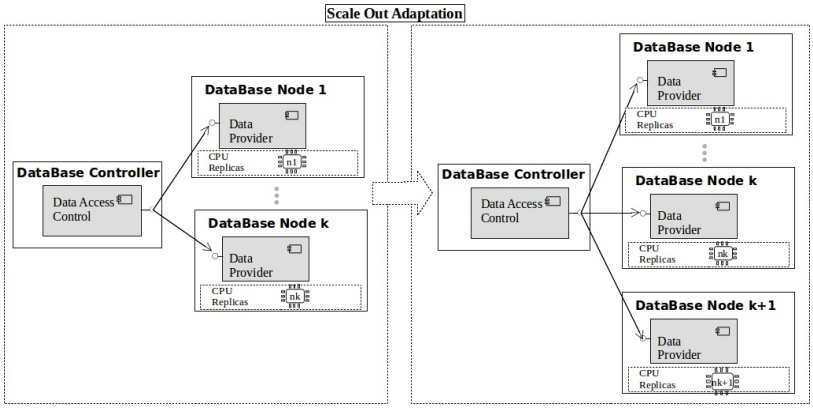


FIGURE 5.2: Scale out adaptation scheme

replicas, the rule scheme represents the addition of a new application node, with the specified initial number of replicas. Inverse operations — scale down to decrease the number of resource replicas and scale in to remove nodes — are also available.

Let us consider one of these operations in some further detail. Specifically, let us focus on the most challenging of them, the scale out associated to a DataBase Controller as the one in our example: when the average usage of CPU in the last time window goes over 65% a new node with the Database component has to be deployed. Although presented for our running example, the operation can be performed on any cluster of nodes upon the occurrence of the correspondent signal. Upon the validation of a SYBL annotation's constraint, a token command is released to trigger the action specified in the corresponding strategy. Adaptation rules then operate on the structure defined in the Palladio models. The addition of a node in any state of the system will imply the modification of the models of the different views in the Palladio description:

- In the component model:
  - a Requires relation between the Controller component and the IDataBase interface should be created;
  - in the SEFF <control>, which models the control flow in the Controller component, a branch (with appropriate likelihood and corresponding external call action) should be created;
- In the assembly model:

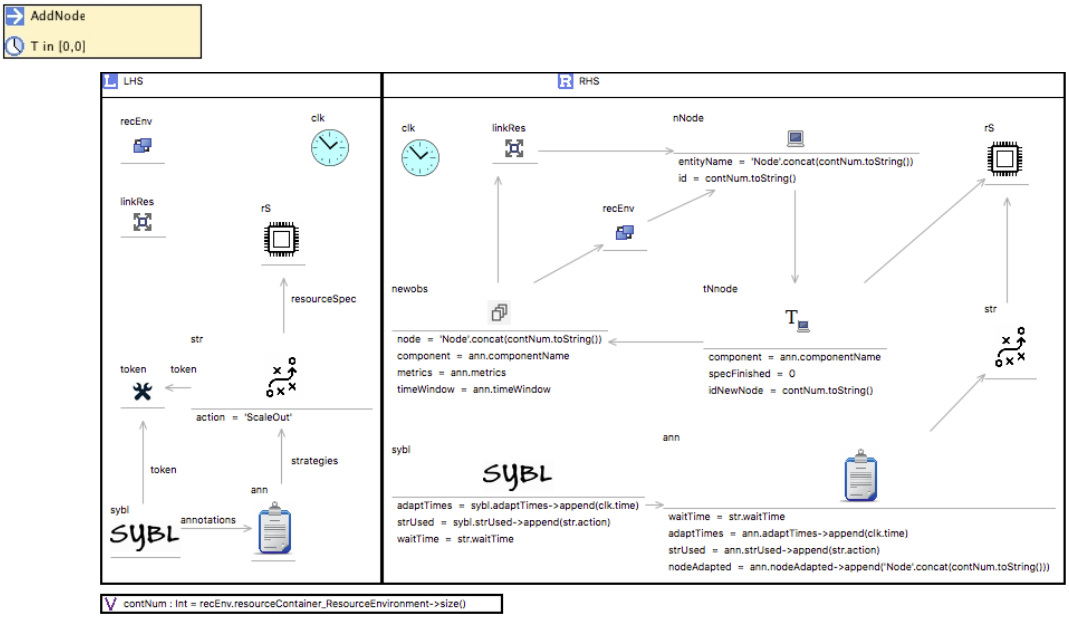


FIGURE 5.3: AddNode rule

  - an assembly context for the new node should be created along with its communication with the Controller component;
- In the allocation model:
  - an allocation of the DataBase component should be added to the new node.

Instead of having one single complex rule modeling what is sketched in the scale out adaptation depicted in Figure 5.2, this operation is implemented by several e-Motions rules taking care of the different actions. The AddNode rule, depicted in Figure 5.3, is the main one, triggering the addition of the node. Given an adaptation token linked to a SYBL annotation strategy with a ScaleOut action, a new node nNode is created in the environment recEnv, linked to the computing lan centre linkRes. In addition, a token indicating the creation of a new node tNnode will guide the triggering of subsequent rules, performing the rest of the necessary actions.

The scale out operation proceeds as follows:

1. As above explained, and depicted in Figure 5.3, a scale out begins with the creation of the node in the AddNode rule.
2. Then, the NewNodeContext rule creates assembly and allocation contexts for the new node. The component for which the new context is created is indicated in the annotation of the specification, as well as the operation signature and the operation interface linked with it. When applying the rule, the new node token is linked with the new assembly context, the operation signature, and the operation interface, and receives the monitor list from the SYBL annotation.
3. The SpecResourceNewNode rule is responsible for configuring the specified resources, as defined in the initial model, for the new node. This rule is applied for each of the resources modelled in Palladio.
4. The ReqProvConnNewNode rule uses the assembly context, the operation signature and the operation interface to create the new assembly connector and the new operation required in the component that has the external action call. This

component can be either a load balancer or a database controller. The external call is within a probability branch in a branch action.
5. The AddBranchNewNode rule adds a new probability branch, with the corresponding external call action, and the signature as indicated by the new-node token.
6. The IncBranchProbability rule balances the probabilities of all branches.
7. The CreatedRTOb and ResourceUsageOM rules create and link observers in accordance with what is specified in the monitors in the annotation.
8. The RemoveTokenNewNode rule is in charge of removing remaining operation tokens once all monitors have been appropriately handled.

The scale in operation proceeds as follows:

1. The DecBranchProbability rule disconnects a probability branch from the branch action, whose probabilities are rebalanced.
2. The RemoveBranch rule removes the probability branch object and the external action linked to it.
3. The RemoveNode rule eliminates the node along with all its connections and assembly context. This rule is triggered when all works present in the resource usage queue of the node to be eliminated have already been completed.
4. The RemoveObservers and RemoveTokenRemove rules remove all observers when it is certain that none still has any value that can be considered by a time window.

5.2 Modeling of Non-Functional Requirements Control

modeling the systems and the environments where they are going to be deployed is not enough in the presence of dynamic behavior, once some kind of adaptation has to be modelled as well. Indeed, we need some way of specifying how the adaptation is managed and controlled.

The e-Motions observers proposed by Troya et al., 2013 use of metrics and adaptation specifications defined directly on the behavior rules and linked to the object to be analysed. In order to make the definition of quality metrics even more flexible, in this work the management of the non-functional properties is performed by a series of independent rules of behavior.

As the proposal of this work, SYBL Monitors work in an integrated way together with e-Motions observers to collect monitoring information on the different metrics being observed. Systems may then, depending on this gathered information, adapt in different ways by performing different operations, like scale up/down, scale in/out, etc. Since the collected monitoring information is stored in the Observer objects, direct access to current values, windows of values of certain length, and complete histories of data are provided through them. The monitoring infrastructure is integrated with the e-Motions observers and runs on the different levels: for each observer, a monitor is created, and a monitor can be associated to one or more observers (in the case of two or more replicas of resource, for example, one observer is created for each replica). Moreover, a monitor can monitor another one. The monitor which directly monitors an observer belongs to level one and the monitor of other monitors belongs to level two. The metamodel proposed in Figure 5.9 along with the definition of behavioral rules allow any quality metrics to be inserted.

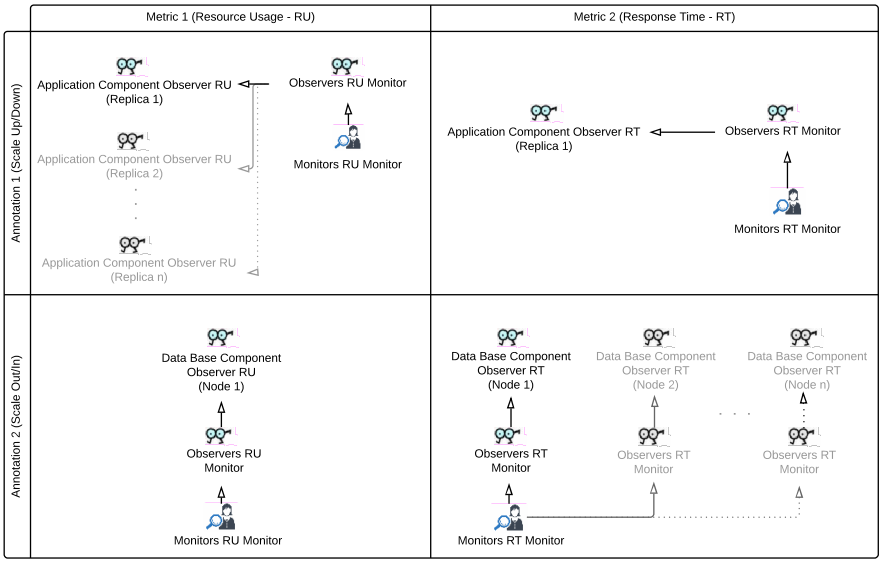


FIGURE 5.4: Relations between observers and monitors

The specific instantiation of observers and monitors for our running example is depicted in Figure 5.4. For inserting a new metric, we just have to specify it in SYBL and create, in the metamodel, a subclass Observer corresponding to it. The information about each metric is saved and calculated during simulation time using the monitors.

The SYBL specifications are used as a reference for controlling non-functional requirements, which are evaluated using a set of rules integrated with the rules defined for the adaptation control.

The CheckConditions rule in Figure 5.5, specifies the evaluation of the conditions associated to a specific SYBL annotation. The checkStatus and checkConstraint OCL functions[2] check the values of the restrictions established in the specification and compare with the values obtained during the simulation by the observers managed by the monitors. This rule checks whether a restriction is violated. If so, it creates a trigger for other rules to decide which adaptation strategy should be applied, according to specification. This is done by releasing tokens that control the flow of rules. The behavior rules checks whether an adaptation action is necessary, and if so, it decides which strategy should be applied, according to specification, and releases a token that indicates the action to be executed.

The management of non-functional properties and the adaptation operations includes rules for the management of controllers, monitors, observers, and time windows. Figure 5.6 depicts the procedure. The beginning of the scheme indicates the first rule for the adaptation control and non-functional requirements. The `CheckConditions` rule continuously verifies the specified constraints. This verification consists in the comparison of the defined values in the SYBL specification to the values obtained during the monitoring in simulation time. The scheme indicates that the monitoring

[2]OCL (Object Constraint Language) auxiliary functions can be defined in e-Motions using e-Motions Helpers.

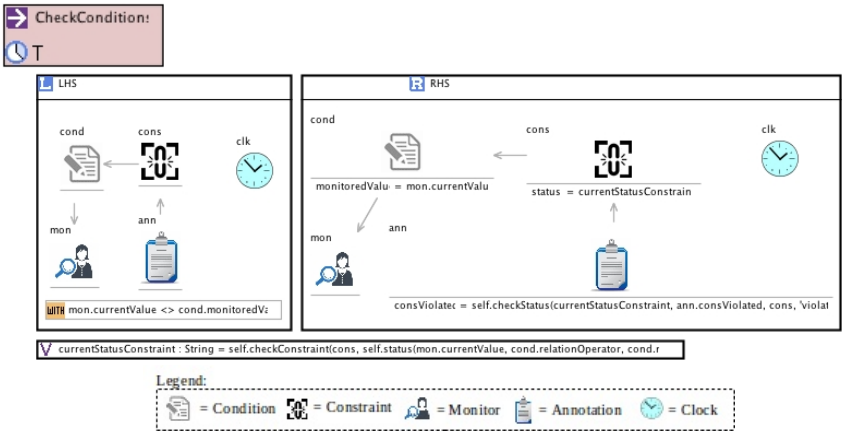


FIGURE 5.5: CheckConditions rule

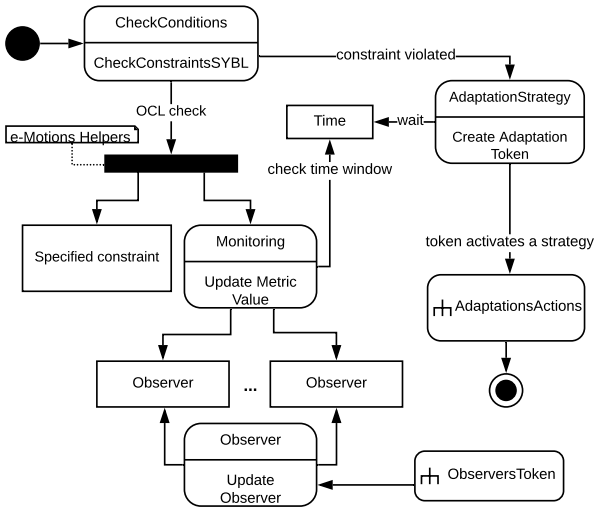


FIGURE 5.6: Controller scheme

values are obtained from the observers, which are updated by tokens that were inserted in the Palladio rules. The update of the monitoring values occur following a pre-established time window.

If the `CheckConditions` rule identifies that a constraint was violated, the `AdaptationStrategy` rule is triggered and it will indicate which adaptation should be applied, according to the SYBL specification. The waiting time between adaptations is also pre-established. Thus, if the time between adaptations overcomes the waiting time a token is created and it will trigger the corresponding adaptation rule to the specified adaptation. The impact of the adaptation actions will be discussed in the next section.

The control of the evolution of the systems can be seen as a typical MAPE-K (Monitor-Analyse-Plan-Execute over a shared Knowledge) feedback loop.Sinreich, 2005 As illustrated in Figure 5.7, in our case, the knowledge is represented by the set of specifications, models and behavior definitions — the `Palladio Rules`, the `Palladio Model`, the `Adaptations` rules, the `Non-Functional Requirements` specification, and the `Control` rules. In summary, the monitoring is carried out with the information obtained by the tokens in the Palladio rules, and updated in the observers. The values obtained from the monitoring are often analysed by comparing them with the

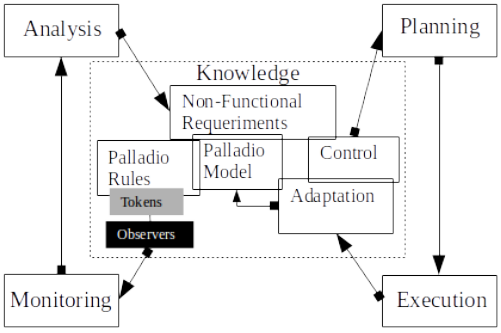


FIGURE 5.7: A MAPE-K Loop interpretation of the approach

specified values of non-functional requirements. The control rules trigger the adaptation planning and, if necessary, they are executed when possible.

5.3 Specifying a SYBL Annotation

In the presence of dynamic behavior there has to be some way of specifying how the adaptation is managed and controlled. This approach used SYBL which, although it was not designed to be used for predictive analysis (but to connect different monitoring tools and cloud APIs, and to indicate elasticity mechanisms for different services), we consider it very appropriate for our purpose due to its ability to describe constraints, monitoring and adaptation strategies.

SYBL is a language for controlling elastic capabilities of cloud applications at runtime, where elasticity is not only referred to resource scalability, but also to cost and QoS properties. In this way, SYBL is capable of expressing conditions combining these three dimensions. Another feature of SYBL is its ability to enable different kinds of users (application developers, software providers, IaaS/PaaS end-users or cloud providers) to specify the elastic features of an application. The language was designed to be API independent, and it provides a control service that can use different monitoring tools and cloud APIs, regardless of the technology used. Furthermore, SYBL was designed to be extensible, that is, it facilitates the involvement of new concepts and allows to link metrics or syntactic entities of the language to external elements that can correspond to measurable variables or units in the system being modelled.

The SYBL expressiveness concerns not only to the multi-dimensional elasticity (resources, cost and quality) but also to the multi-level description possibilities. Thus, it is possible to specify different elasticity constraints and strategies at the application, component and programming levels, by means of the so-called *directives*. An annotation is a set of different directives, and represents the requirement specifications for an application and each of their components. For each annotation, we need to indicate: the specification level (application, component or programming), and a label for defining the monitoring metrics, for describing the constraints to be taken care of, and for establishing the strategies to apply when necessary. In addition, other information could also be provided (identifier, component name, priorities of constraint and strategies, etc.).

⟨*Constraint*⟩ := constraintName : CONSTRAINT **ComplexCondition**

⟨*Monitoring*⟩ := monitoringName : MONITORING varName=**MetricFormula**

⟨*Strategy*⟩ := strategyName : STRATEGY WHEN **ComplexCondition** : action(parameterList)
| strategyName : STRATEGY WAIT **ComplexCondition**
| strategyName : STRATEGY STOP
| strategyName : STRATEGY RESUME

⟨*MetricFormula*⟩ := metric | number | metricFormula **MathOperator** metric
| metricFormula **MathOperator** number

⟨*ComplexCondition*⟩ := **Condition** | **ComplexCondition BitwiseOperator Condition**
| (**ComplexCondition BitwiseOperator Condition**)

⟨*Condition*⟩ := metric **RelationOperator** number | number **RelationOperator** metric | **Violated**(name)
| **Fulfilled**(name)

⟨*MathOperator*⟩ := + | − | ∗ | /

⟨*BitwiseOperator*⟩ := OR | AND | XOR | NOT

⟨*RelationOperator*⟩ := <| >| >= | <= | == | !=

FIGURE 5.8: BNF grammar of SYBL

5.4 SYBL Metamodel

The grammar of the SYBL language, in Backus Naur Form (BNF), is shown in Figure 5.8. In our approach, to be able to use SYBL annotations textually specified using this grammar, we have created in e-Motions the Control metamodel, which is shown in Figure 5.9, in this metamodel Classes represent the expressions and terms that the grammar defines.

The `SYBLrequirements` class relates to one or more `REQUIREMENTSAnnotation` classes, has two attributes and indicates when SYBL annotation has been violated or fulfilled. The `REQUIREMENTSAnnotation` class represents the built specification. Each annotation must contain the specification of the constraints (`REQUIREMENTSComplexCondition`), the adaptation strategies (`REQUIREMENTSComplexStrategy`), the specification of the monitors and their monitoring metrics (`REQUIREMENTSMonitoring` and `REQUIREMENTSMetric`). SYBL Monitors work in an integrated way together with e-Motions observers (the e-Motions observers was proposed by Troya et al., 2013) to collect monitoring information on the different metrics being observed.

The proposed metamodel, along with the definition of behavioral rules, allow any quality metric to be inserted. For inserting a new metric, we just have to specify it in SYBL, and create, in the metamodel, a subclass Observer associated to it. The information about each metric is collected and calculated at simulation time using these monitors.

The SYBL specifications are written according to the metamodel of Figure 5.9 and are used as a reference for controlling non-functional requirements, which are evaluated using a set of rules integrated with the rules defined for the adaptation control.

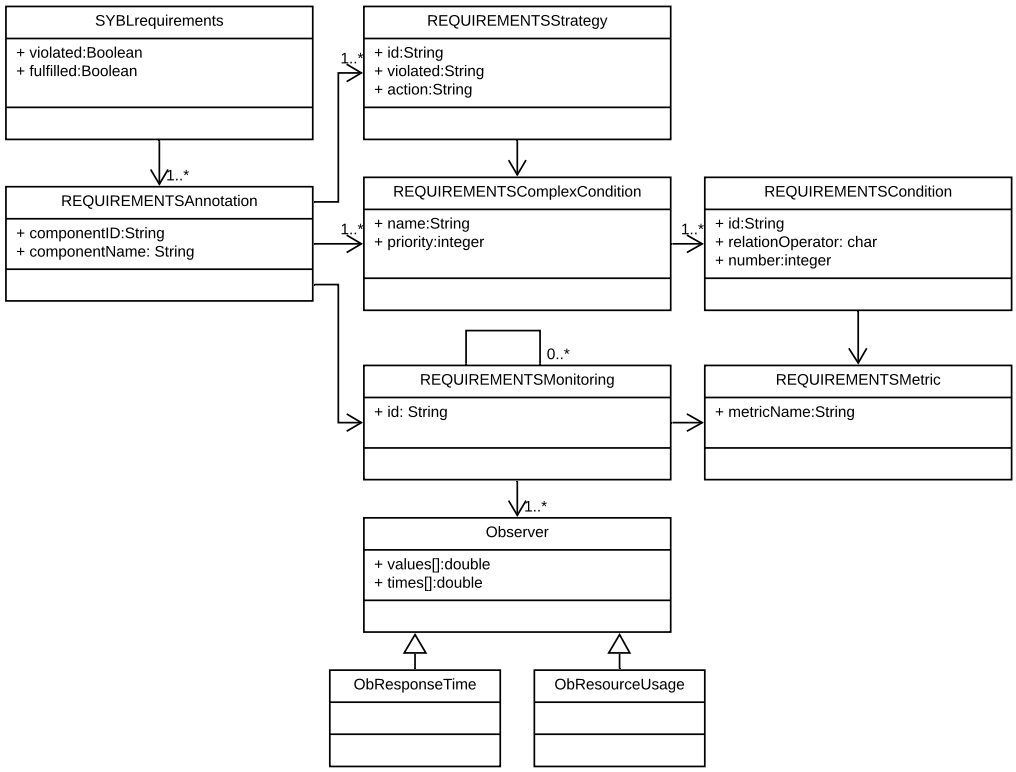


FIGURE 5.9: Control Metamodel

5.5 Modeling of Communication Channel

Another significant advance of this work regarding the implementation of Palladio in e-Motions was the modeling of the communication between containers. To clarify the modeling, first we present the activity diagram that represents the operation of Palladio in e-Motions (Figure 5.10), which already includes the contribution made regarding the modeling of the "Link Resource".

The flow starts in the Usage Model whose role is to call each work to the system. The first step of the system flow is to verify the Assembly model that will follow the execution according to the proposed system modeling. The Assembly model contains one or more Assembly contexts. The identification of Assembly contexts is important because this is where the component models, which describe the SEFF services, are encapsulated.

An Allocation context may contain one or more Assembly contexts. Each Allocation context, in turn, belongs to an Allocation model which is linked to the resource environment that contains containers and the modeled link that allows the communication between each container. A new Assembly context is always created when a "Scale Out" occurs, because a new node will be created that will perform a service. SEEF services allow external calls that may be directed to services on the same node or on another one. Initially, Palladio's implementation in e-Motions did nott consider the resource link. This implementation was made from this work.

Figure 5.11 represents the transition rule for an Internal Action and Figure 5.12 represents the transition rule for an External Action. This distinction was important

since it is only in External Actions that calls are made to execute works on different nodes.

The transition rule for External Action is more complex because it is necessary to identify several elements, such as the Assembly context that makes the call and the Assembly context that receives it, as well as the Allocation context that makes the call and the one that receives it. In the Transition to External Action rule (Figure 5.12) is also necessary to store possible variables inserted in the External Action modeling and the information about the resource link that was defined in the modeling of these contexts. The External Action rule was modified (Figure 5.11) to take into account the latency of the link between containers.

Figure 5.14 shows the container change rule that is activated when a service is called and is in a different container than the one that requests it. This call will then go through the communication channel, and the network link settings must be considered. The rule checks the specified throughput and latency values.

Another important modeled rule is present in Figure 5.15, which represents the Transition Branch rule. It is responsible for identifying the branch type modeled on the corresponding SEFF (it could be a probability branch or a Guarded branch, for example). Modeling this rule was important to understand what kind of conditions might be present in branches. Initially it was possible to carry out only probabilistic transitions, but now the possibility of transitions with more complex conditions was introduced using the Guarded branch, such as the verification of the lowest latency between the nodes that allocate a certain service to choose which node to execute.

5.6 Results

5.6.1 Adaptations Mechanisms Rules

Different adaptation mechanisms available are defined as transformation rules on the model of the system under analysis. Thus, given monitoring information on the different metrics being observed, systems may adapt in different ways by performing different operations, so far, we have available the adaptations scale up/down and scale in/out. The collected monitoring information is stored in the observer objects, giving direct access to current values, moreover, windows of values of certain length, and complete histories of data are being used.

Scale Out Mechanism

Consists on adding computation or storage nodes, typically organised in a cluster. The scale out operation proceeds as follows:

Add Node rule. The Add Node rule is the main one firing the addition of the node. Given an adaptation token token linked to a SYBL annotation strategy with a ScaleOut action, a new node nNode is created in the environment recEnv, linked to the computing lan center linkRes. An observer object is created together with the node to collect information on this new node. In addition, a token indicating the creation of a new node tNnode will guide the firing of subsequent rules, performing the rest of the necessary actions.

New Node Context rule. The New Node Context rule creates assembly and allocation contexts for the new node. The component for which the new context is created

is indicated in the annotation of the specification, as well as the operation signature and the operation interface linked with it. When applying the rule, the new node token is linked with the new assembly context, the operation signature, and the operation interface, and receives the monitor list from the SYBL annotation.

Resource Specification of the New Node rule. The Resource Specification of the New Node rule is responsible for configuring the specified resources, as defined in the initial model, for the new node. This rule is applied for each of the specified resources, and a resource observer is created for each of them.

Required and Provided Connection of the NewNode rule. The Required and Provided Connection of the NewNode rule uses the assembly context, the operation signature and the operation interface to create the new assembly connector and the new operation required role in the component that has the external action call. This component can be either a load balancer or a database controller. The external call is within a probability branch transition in a branch action.

Add Branch New Node rule. The Add Branch New Node rule adds a new probability branch transition, the branch action, with the corresponding external call action, and the signature as indicated by the new-node token.

Increase Branch Probability rule. The Increase Branch Probability rule balances the probabilities of all branches.

Created Response Time Observer rule. The Created Response Time Observer rule create and link observers in accordance with the specified annotation monitors.

Created Resource Usage Observer rule. The CreatedResource Usage Observer rule create and link observers in accordance with the specified annotation monitors.

Scale In Mechanism

Consists on removing computation or storage nodes, typically organised in a cluster. The scale in operation proceeds as follows:

Decrease Branch Probability rule. The Decrease Branch Probability rule disconnects the probability branch transition and the branch action connected to the constraint violated that triggered the adaptation.

Load Balancer rule. The probabilities of the branch action are re-balanced.

Remove Branch rule. The Remove Branch rule removes the probability branch transition object and the external action linked to it.

Remove Node rule. Rule Remove Node rule eliminates the node along with all its connections, assembly context, and observers. This rule is triggered when all works present in the resource usage queue of the node to be eliminated have already been completed.

Remove Observers rule. The Remove Observers rule remove all observers, making sure that no one still has any value that can be considered by a time window.

Remove TokenRemove rule. The Remove TokenRemove rule remove the token remove.

Scale Up and Scale Down Mechanisms

The scale up/down operations change the number of replicas available in an node.

Scale Up. The scale up rule increase the number of replicas of the resource specified.

Scale Down. The scale down rule decrease the number of replicas of the resource specified.

5.6.2 QoS Metrics Measurement Rules

In this approach, the different adaptation mechanisms available are defined as transformation rules on the model of the system under analysis. Thus, given monitoring information on the different metrics being observed, systems may adapt in different ways by performing different operations, like scale up/down, scale in/out, etc. The collected monitoring information is stored in the observer objects, giving direct access to current values, windows of values of certain length, and complete histories of data.

5.6.3 Adaptation Control Rules

Modeling a system and the environment where it is going to be deployed is not enough in the presence of dynamic behavior, when some kind of adaptation has to be modelled as well. We use SYBL to describe elasticity requirements and strategies to react upon the occurrences of given conditions in the environment.

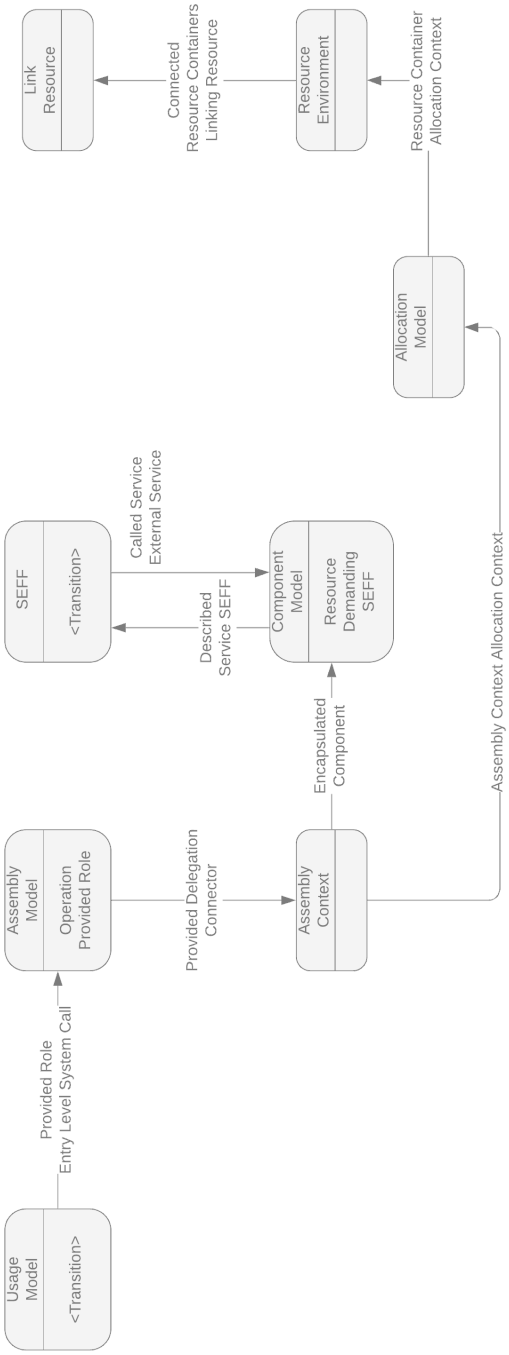


FIGURE 5.10: Palladio in e-Motions Activity Diagram

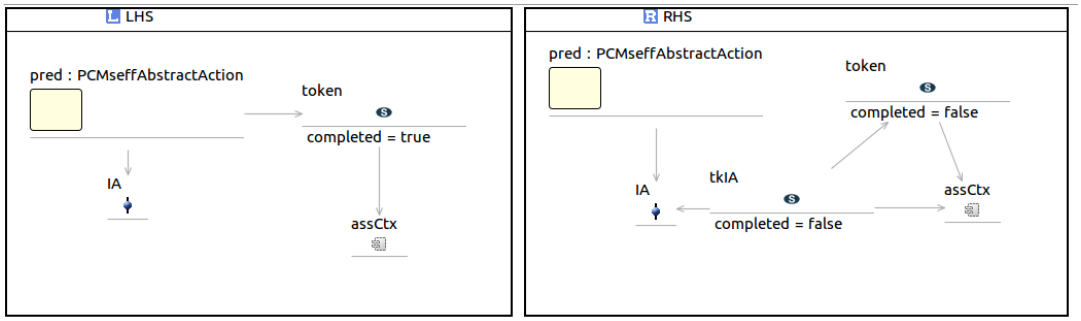


FIGURE 5.11: Transition to Internal Action Rule

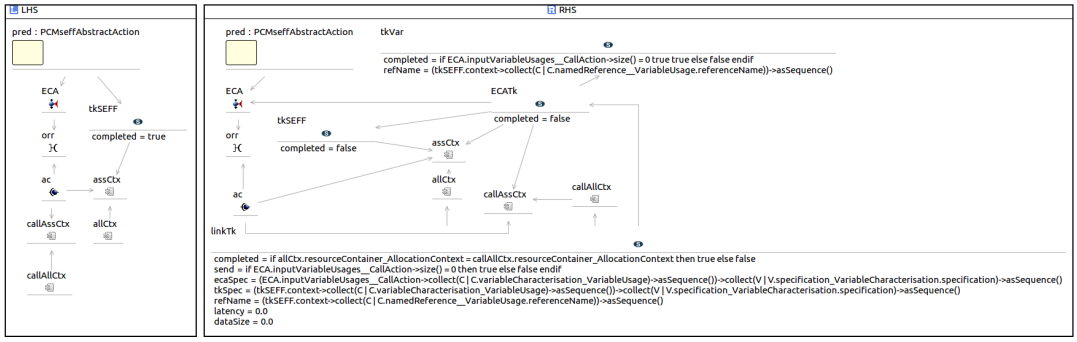


FIGURE 5.12: Transition to External Action Rule

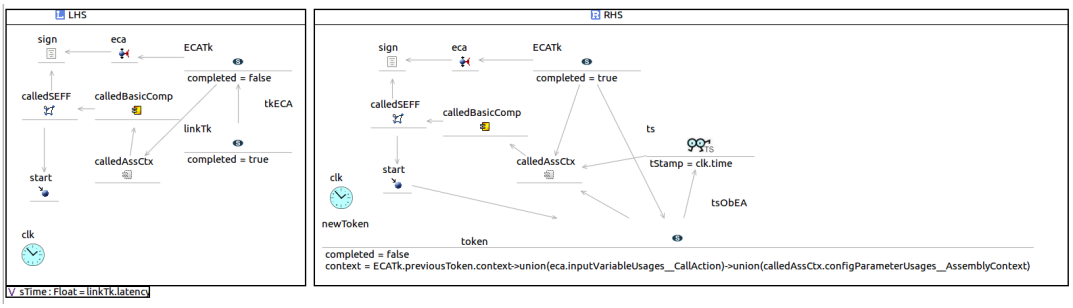


FIGURE 5.13: Modified External Action Rule

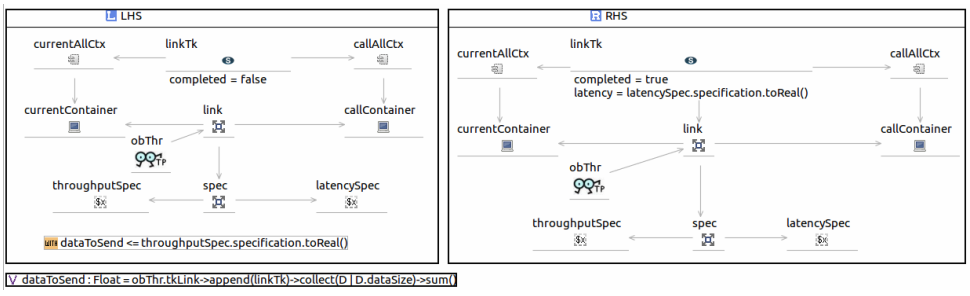


FIGURE 5.14: Container Changer Rule

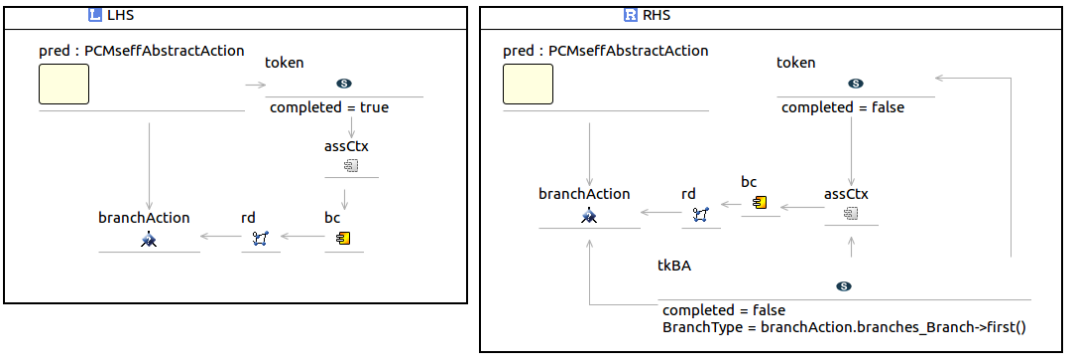


FIGURE 5.15: Transition Branch Rule

6 Modeling Process for Self-Adaptive Systems

In this chapter we will present the modeling process for self-adaptive systems, considering the phases approach: specification, modeling, behavior definition and analysis. We will present the proposed framework following the illustration of each phase of the process.

6.1 Process Phases

Figure 6.1 shows a process model for our approach using the *Business Process Model and Notation (BPMN) – Version 2.0*, in which we can see the role of each tool and the different activities of the process: Specification, Modeling, behavior Definition and Analysis.

In the Specification phase, the non-functional requirements and adaptation strategies are specified using the SYBL language (Copil et al., 2013). In the modeling phase, the initial model of the system to be analysed is modelled using Palladio tool. As a result of the task, we obtain models of the different views of the application. This model can be defined according to the specifics of each application to be analysed.

In the behavior Definition phase we consider the rules defined by graph transformation in e-Motions. These rules concern the behavior of DSLs. We have defined rules that represent Palladio's behavior, adaptation rules and rules that define the behavior of adaptation control mechanisms considering non-functional requirements. All of these rules can be easily modified as needed. Furthermore, at this phase the Maude code is generated (`Model transformation [e-motions]`). It is obtained by a model-transformation provided by the e-Motions tool. The Maude specification obtained can then be used to carry out the simulation in the Analysis phase. The e-Motions transformation takes as input: (1) the initial model, defined using Palladio in the modeling phase; (2) the metamodel extended of the Palladio Component Model (PCM); (3) the e-Motions specification of definition of the behavior of Palladio; (4) definition of the behavior of the adaptation mechanisms; and (5) definition of the behavior of the control for adaptation mechanisms and non-functional requirements.

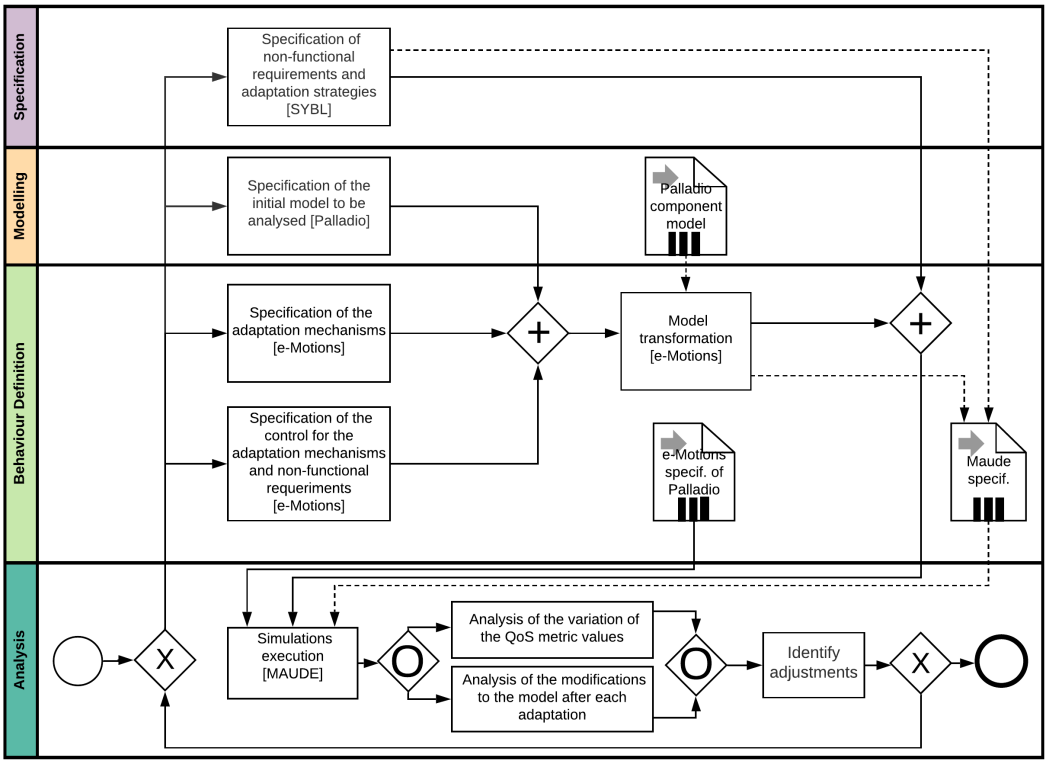


FIGURE 6.1: Procedural view of the approach using the BPMN standard

In the Analysis phase, the Maude system is used to execute the simulations of the Maude codes generated in the previous phase. During and at the end of the simulations, it is possible to view the its results, which contain the data related to the simulated system. This data shows the values of the variation of the QoS metrics monitored during the simulation as well as, if applicable, the modifications to the model that occurred after the performed adaptations. This data allows the analysis and, later, the adjustments in the initial model or in the parameters, as needed.

6.2 Approach Framework

The diagram in Figure 6.2 depicts the main elements in our proposal and how they are integrated. The upper left square represents Palladio's dual role. The first role

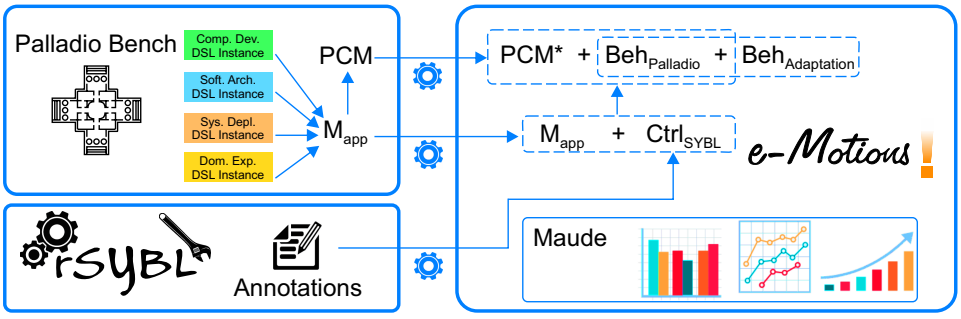


FIGURE 6.2: Artefacts and processes of the approach

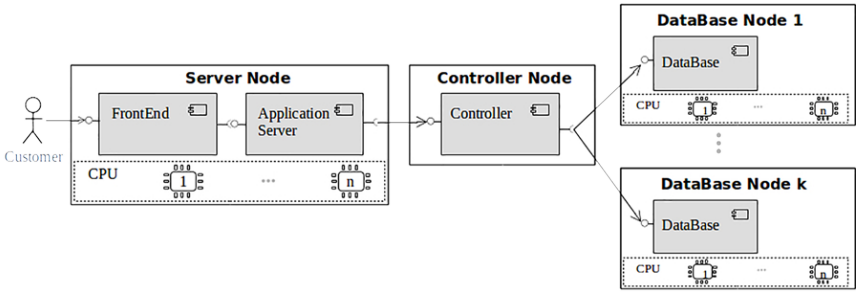


FIGURE 6.3: Structure of the example

is in building the system's models to be analysed (M_{app}). Models conforming to the PCM are composed of four different sub-models, which correspond to the respective views of systems. The second role is the representation of its abstract syntax (the PCM).

The lower left corner represents the description of the control of the execution, performed by SYBL Annotations, which are written in Maude code to be interpreted in the metamodel and in the adaptation rules modeled in e-Motions.

The right side represents the role of the e-Motions tool. Palladio is specified in e-Motions both through its syntax — an extended PCM denoted PCM^* — and through its behavior — denoted $Beh_{Palladio}$. The static systems defined in Palladio (models M_{app} conforming to the PCM) can be loaded and analysed in e-Motions using the DSL $PCM^* + Beh_{Palladio}$. To deal with adaptive systems, the Palladio behavior was extended, including adaptation mechanisms specified as additional e-Motions rules ($Beh_{Adaptation}$). This allows us to use available Palladio models, or use the Palladio tools to design them ourselves. SYBL is specified in e-Motions through of the metamodel and by behavior rules $Ctrl_{SYBL}$. The combination $M_{app} + Ctrl_{SYBL}$ conforms to the DSL metamodel $PCM^* + (Beh_{Palladio} + Beh_{Adaptation})$, and they are all transformed into Maude using the e-Motions tool. The generated Maude code can then be used for simulation. Finally, the results obtained from the simulations may be used for performance analysis.

6.3 Results

6.3.1 Specification of the application

In the specification phase, some steps are necessary to be followed. First, we need to understand the application domain, defining the requirements. With this step completed, it is necessary to specify the SYBL annotations, which will be used in the following phases. In this section we will introduce these steps in detail.

Application Domain

To the first phase of the self-adaptive system modeling process, we first need to understand the application domain. This requires the defining of the domain properties as well as specifying of the requirements.

Domain properties

Figure 6.3 shows the structure that we are going to use as example with the following components: Front End represents the Web Server with the HTTP service; Application Server represents the process request service; the Controller component is responsible for the load balancer to the database; and the DataBase component, with the data process service. The structure in Figure 6.3 not only showed these components, but it also showed their allocation. Each of the components is allocated in its own node, with the exception of the FrontEnd and Application Server, which will be allocated together in the Server node. The Figure 6.3 also includes information on resources. The Server node only allows vertical scaling, that is, we can only increase or decrease the processing capacity of its CPU resource. However, the Controller and the DataBase nodes are allocated to cloud providers that allow to scale not only vertically, but also horizontally (i.e., to add nodes and allocate components on them).

To simplify the abstraction, we consider that all user requests will need the same demand for resources. However, they will be different for the Application Server and DataBase components. Of course the workload will change along the simulations. As an initial model, we will have three nodes: Server, Controller and DataBase.

Requirement Specifications Example

For the example, let us consider the following requirements (R1-R4) which will be used as a basis for the SYBL annotation that will be presented next:

- R1: The response time for a user request should not be, on average, greater than 0.5 time units (t.u.).
- R2: The percentage of resource usage for nodes should not be greater than 65%.
- R3: If R1 is not met, that is, if the average response time is greater than 0.5 t.u., the system must restore an average response time of 0.5 t.u. or less as soon as possible.
- R4: If R2 is not met, that is, if the percentage of resource usage of the nodes is greater than 65%, the system should re-adjust to decrease the workload on the nodes as soon as possible.

SYBL Annotation

Listings 6.1 and 6.2 show samples of SYBL annotations with resource usage and response time quality metrics, respectively. Both are component-level annotations, which specify a ComponentID, its ComponentName, and the description of constraints, monitoring and strategies. The constraints section of the annotations includes the specification of conditions and their priorities. Each condition indicates the expected threshold for certain monitoring metrics — conditions Co1 and Co2 on the CPU usage (cpuUsage) in Listing 6.1, lines 4-5, and Co3 and Co4 on the response time (rt) in Listing 6.2, lines 4-5. Metrics referring to resources usage are given as percentages, whereas those on time are expressed in time units. The strategies section provides the actions to be performed when constraints are violated. The St1 strategy in Listing 6.1, line 9, will produce a scaling-up operation when the CPU usage is over 65% (condition Co1 is violated), while the St2 strategy (line 10) will produce a scaling-down operation when the CPU usage is under 30% (condition Co2 is violated). Similarly, the strategy St3 in Listing 6.2 will produce a scaling-out operation when the response time of the service offered by the DataBase component is greater than 0.5

LISTING 6.1: Component-level SYBL annotation for Application-Server Component

```
@SYBL_ComponentContext(
  ComponentID = Component3;
  ComponentName = ApplicationServer;
  constraints = "Co1: CONSTRAINT cpuUsageAPP < 65;
                 Co2: CONSTRAINT cpuUsageAPP > 30;
                 Priority(Co1) = 2,
                 Priority(Co2) = 1,
  monitoring = "Mo1: MONITORING cpuUsageAPP = ObResourceUsage",
  strategies = "St1: STRATEGY WHEN Violated(Co1): ScaleUp;
                St2: STRATEGY WHEN Violated(Co2): ScaleDown)
```

LISTING 6.2: Component-level SYBL annotation for DataBase Component

```
@SYBL_ComponentContext(
  ComponentID = Component4;
  ComponentName = DataBase;
  constraints = "Co3: CONSTRAINT rt < 0.51;
                 Co4: CONSTRAINT rt > 0.21;
                 Priority(Co3) = 2,
                 Priority(Co4) = 1,
  monitoring = "Mo3: MONITORING rt = ObResponseTime",
               St3: STRATEGY WHEN Violated(Co3): ScaleOut;
               St4: STRATEGY WHEN Violated(Co4): ScaleIn")
```

(condition Co3 is violated) and the St4 strategy will produce a scaling-in operation when the response time is under 0.2 (condition Co4 is violated).

6.3.2 Modeling an Application on the Palladio Bench to use in e-Motions

The static systems defined in Palladio (models M_{app} conforming to the PCM) can be loaded and analysed in e-Motions using the DSL $PCM^* + Beh_{Palladio}$, which was initially proposed in (Moreno-Delgado et al., 2014).

To illustrate the Palladio views, we present the modeling of the scenario with previously specified (Figure 6.3). Figures 6.4-6.10 are snapshots of the specification of the system defined using the Palladio Bench.

Figure 6.4 shows a resource environment example modelled in Palladio in which there are three containers with processing resources specifications of the CPU type. In this model, the communication link between the containers is not considered. On the other hand, Figure 6.5 shows the same resource environment example modelled in Palladio, however, the model considers different communication links between containers, where latency and throughput are specified. Latency is defined as the round trip time for a workload to go through that communication channel, and throughput is defined as the maximum number of workload that can go through that channel per time unit. The adaptations can act directly on theses containers, whether by adding a new container or by changing the number of replicas of the resource.

In Palladio, component models describe the components and interfaces in the system and the relations between them. Figure 6.6 shows the component repository for our example. It depicts four components and their corresponding interfaces: FrontEnd implements IFrontEnd, ApplicationServer implements IApplicationServer, Controller

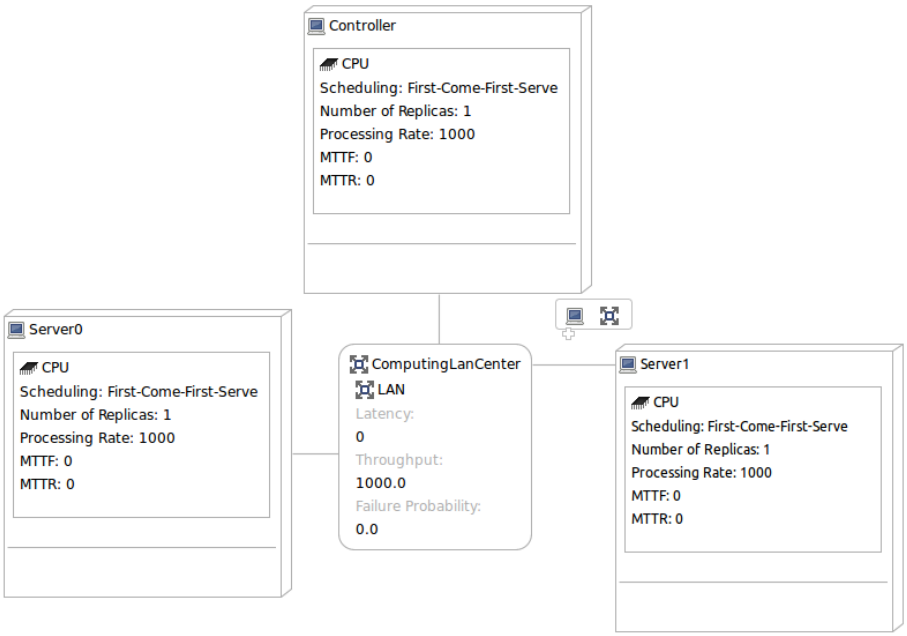


FIGURE 6.4: Resource Environment without considering communication between containers

FIGURE 6.5: Resource Environment with communication between containers

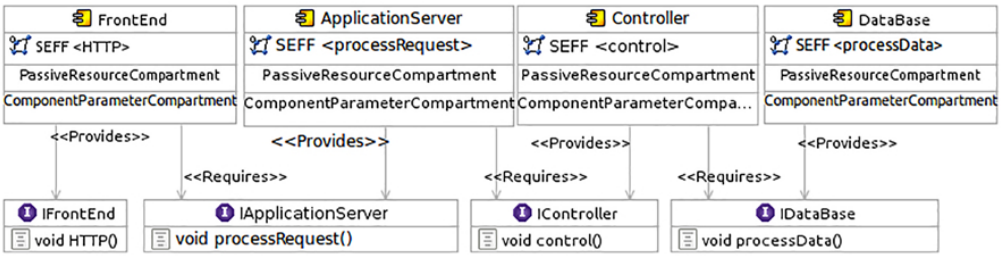


FIGURE 6.6: Component Model

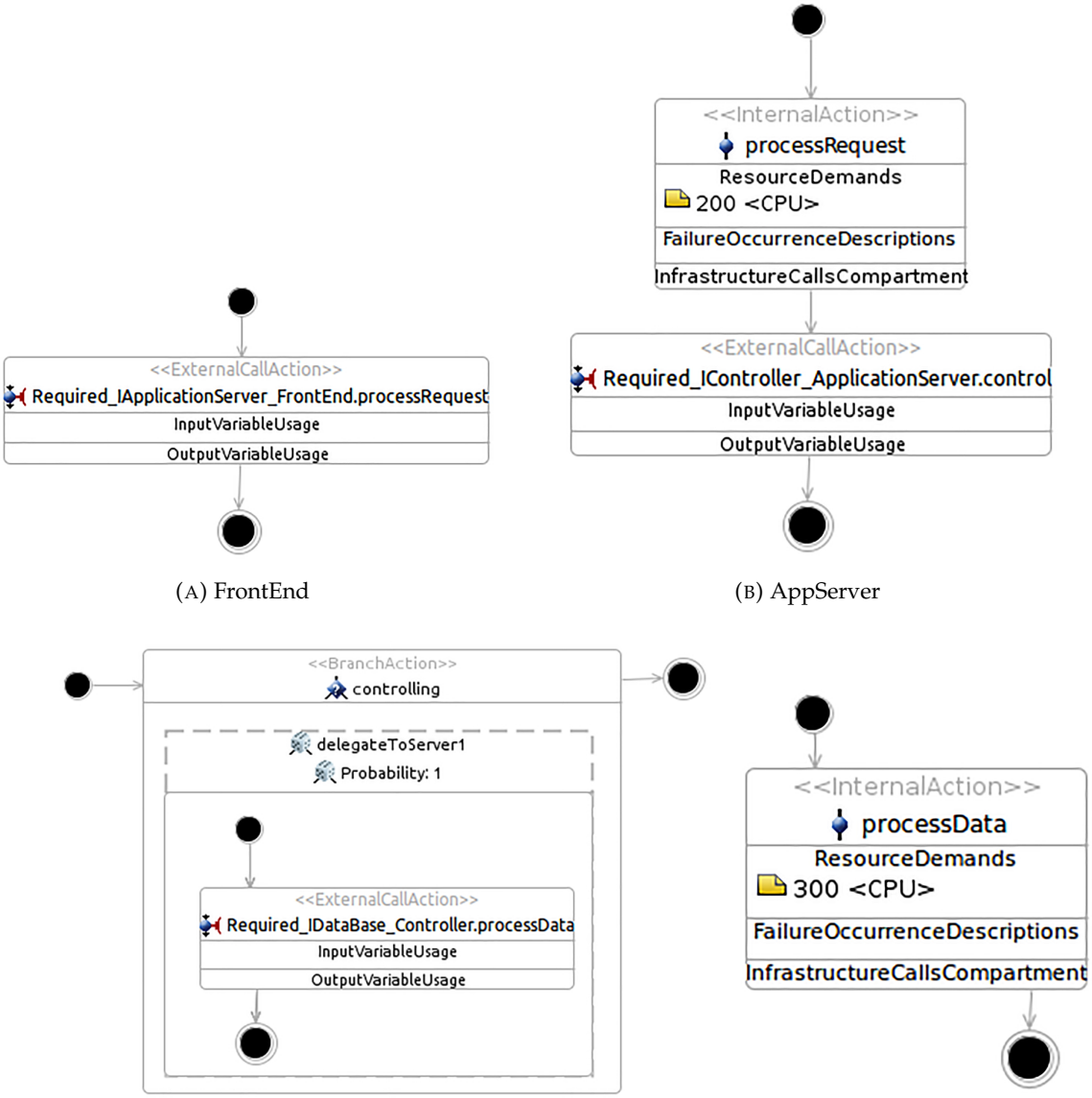
implements IController and DataBase implements IDataBase. Relations between components are represented using Provides and Requires links. There are three Requires relations: (1) from the FrontEnd component to the IApplicationServer interface, that offers the processRequest() operation; (2) from the ApplicationServer component to the IController interface, that offers the control() operation; and (3) from the Controller component to the IDataBase interface, that offers the processData() operation.

Components' services are described by service effect specifications (SEFF), which abstractly model the externally visible behavior of a service with resource demands and calls to required services. Figure 6.7a shows the SEFF of the HTTP() service, which models the behavior of the FrontEnd component that contains an external call action to the ApplicationServer component. Figure 6.7b shows the SEFF of the processRequest() service, which models the behavior of the ApplicationServer component, processes an internal action that consumes 200 units of CPU (CPU cycles), then and carries out an external call action to the Controller component. Figure 6.7c shows the SEFF of the control() operation, which models the control flow in the Controller component as a probabilistic branching. Figure 6.7d shows the SEFF of the processData() service, which models the behavior of the DataBase component — such processing consists in an internal action that consumes 300 units of CPU (CPU cycles).

The proposed in this work allows changes such as increasing/decreasing the resources available, or adding/removing nodes as needed. Since there is only one branch in this initial Palladio definition, the probabilistic branching of the Controller component starts at 100%. This branch has an external call action to the DataBase node of the model. As new nodes are added to the architecture, new branches will be added to this action, thus modeling the distribution of works between the existing servers handled by the controller. Note that this type of operations is not possible neither in Palladio nor in its SimuLizar extension for adaptive systems.

Software architects assemble components from the repository to build applications, represented by assembly models in Palladio. Figure 6.8 shows how the services of components are organised. The biggest square surrounding the boxes represents the entire environment. For each provides relation in the repository model (Figure 6.6), an assigned role is created for the container with such component.

Allocation models are provided by system deployers, who model the resource environment and the allocation of components from the assembly model to different resources of the environment. Figure 6.9 shows the initial allocation model for our example, where we can see how each of the components is allocated in the nodes. The FrontEnd and the ApplicationServer components are allocated in the same node, whereas the Controller and DataBase components are allocated in different nodes.



(A) FrontEnd (B) AppServer

(C) DataBase Controller (D) DataBase

FIGURE 6.7: Components SEFFs

Finally, usage models are provided by domain experts, who specify a system's usage in terms of workload, user behavior, and parameters. Given the usage model definition in Figure 6.10, in our example, tasks will arrive following an exponential probability distribution with rate parameter 5.0 (Exp(5.0)).

After modeling all the views in Palladio, we can then make the transformation to Maude code in e-Motions, which can be used in simulations. For this, it is necessary to indicate: the e-Motions files that correspond to the behavior rules and the metamodel that will be used in the simulation; the files of each view of Palladio; a simulation time; and where the generated files will be saved, as shown in Figure 6.11.

The use of these Palladio models in our approach was facilitated by the work by

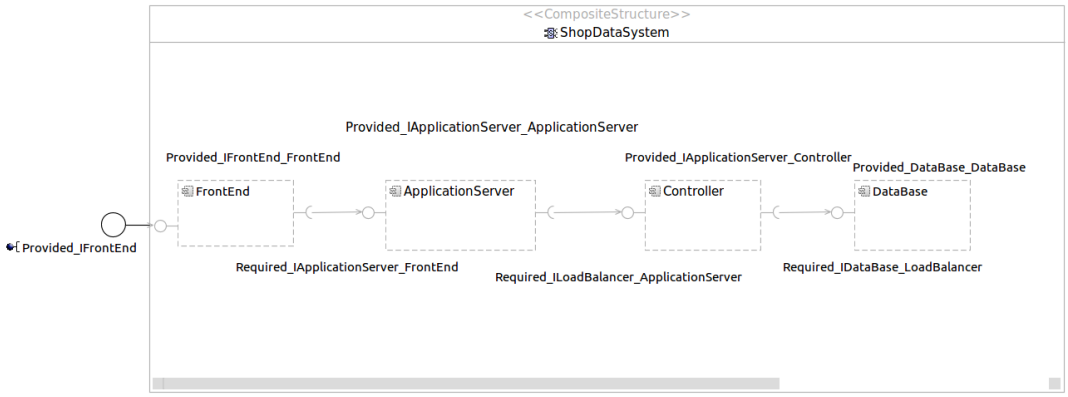


FIGURE 6.8: Assembly Model

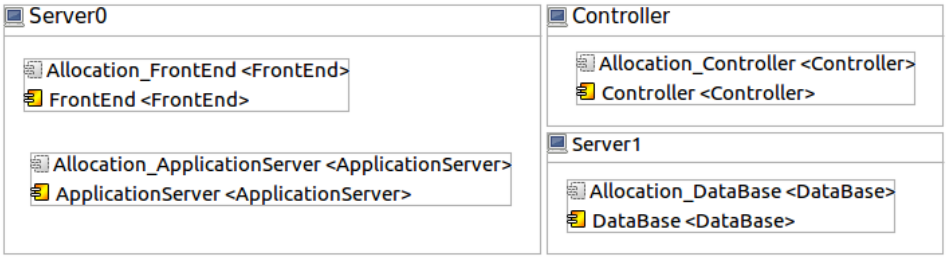


FIGURE 6.9: Allocation Model

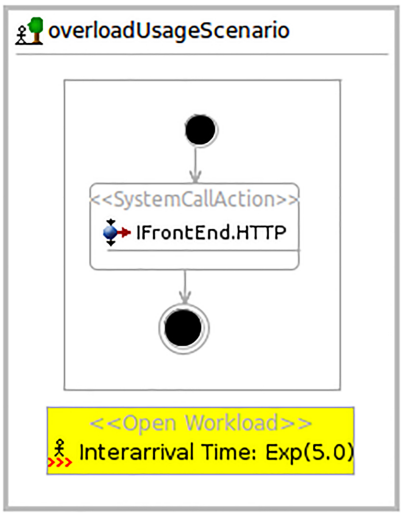


FIGURE 6.10: Usage Model

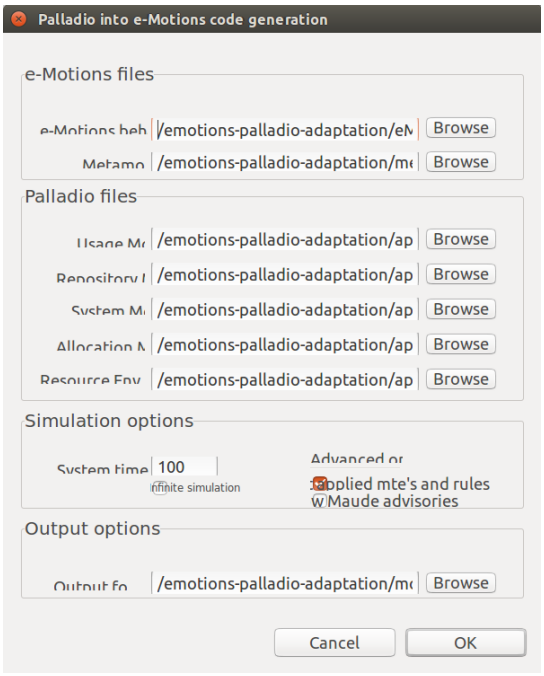


FIGURE 6.11: Screenshot Palladio into e-Motions Code Generation

Moreno-Delgado et al., 2014, which allowed the transformation of the Palladio system models into Maude code, using the e-Motions tool. In this work, we did not make changes regarding the model transformations.

6.3.3 Analysis in Design Time

System with no Adaptation

As a first step for analysis, we need to understand its behavior without any kind of adaptation, to view the problems in the model configuration, as for example the possibility of bottlenecks causing delays or if there is a problem related to the use of resources in the node where one of the components is allocated.

First, we need to add a few additional parameters and assumptions. In our example, three nodes were specified, and to simplify both the presentation of the model and of the results, we consider that the nodes on which the components are allocated have the same configuration. Each of these nodes is specified with CPU processing rate of 1000, First-Come-First-Serve scheduling, and initially there is one CPU resource replica for each of them.

The workload arrives first in the components allocated in Server0; then, after processing in the two components allocated in Server0, the workflow goes to a request in the component allocated in node Controller. To send this request the system goes through a communication link of 0.002 t.u. of latency and 50 work per time unit of throughput. The component allocated in node Controller, in turn, sends a request to Server1, and this request goes through a communication link of 0.005 t.u. of latency and 100 work per time unit of throughput.

In the example, the ApplicationServer and DataBase components demand, respectively, 200 and 300 CPU units. The demand of CPU of a specific service, the processing rate of the node on which it operates give us its runtime. Thus, the service

of the ApplicationServer component — CPU resource demand 200 — is executed on a CPU with processing rate 1000, which means that its execution takes 0.2 t.u. and the service of the DataBase component — CPU resource demand 300 — is executed on a CPU with processing rate 1000, which means that its execution takes 0.3 t.u.[1]

We use the Resource Usage (RU) and the Response Time (RT) metrics to obtain information on the ApplicationServer and DataBase components. The RU metrics will be obtained from the monitoring of the use of the nodes where each of these components is allocated. The RT of component ApplicationServer will indicate the total running time of the system (0.5 t.u.) plus the latency of the communication channels (0.007 t.u.), and the RT of component DataBase will indicate the time it takes for a work from its arrival to the completion of the data processing (0.3 t.u.) plus communication channel latency from Controller node to Server1 node (0.005 t.u.). We use a closed workflow with time between arrivals given by the stochastic expression $\mathsf{Exp}(5.0)$, that is, tasks arrive after an exponential probability distribution with rate parameter 5.0 t.u.

The observation of the metrics on the simulation of the system with these parameters produces the charts in Figure 6.12. Regarding resource usage (Figure 6.12b), we observe that the DataBase component's node goes to using 100% of its resources 5 t.u. after the beginning of the simulation. The usage varies in the node of the ApplicationServer component in the range 60-100% for most of the time. In both cases, the response time remained increasing throughout the simulation (chart in Figure 6.12c). In addition, the resource usage queue of the node of the ApplicationServer component remains most of the time below 5. In contrast, the resource usage queue shows a bottleneck of the node that allocates the DataBase. Remembering that the throughput defined for the first link is 50 and for the second one is 100. With the data presented from the resource usage queue, it is not possible to notice a considerable number of users to consider an adaptation in the communication channels.

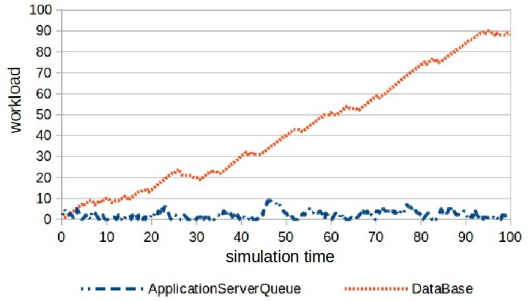
These charts tell us that the response time degrades along time, and possibly due to an under allocation of resources, perhaps in the node of the ApplicationServer component, but for sure in the DataBase component's node.

Analysis with Ann1: modifying the number of replicas in the ApplicationServer's node

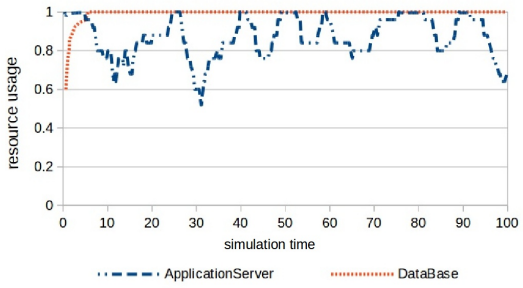
Annotation Ann1 (Listing 6.3) specifies an adaptation on the ApplicationServer component considering the resource usage of the node where the component is allocated. Specifically, we define that adaptations must occur when the average in the resource usage monitor is above 65% (scale up) and below 30% (scale down).

Figure 6.14 depicts the resource usage in components ApplicationServer and DataBase in a simulation with the Ann1 elasticity specification. The vertical lines in the charts show the times at which the adaptation operations get triggered. Thus, we observe a scale up adaptation at time 0.4. Right after that time, the resource usage decreases considerably for the ApplicationServer component, varying between 30% and 60% for most of the simulation time. However, the change occurred only in the ApplicationServer component's node, which explains the change in its resource usage, while the node where the DataBase component was allocated stayed with a resource usage of

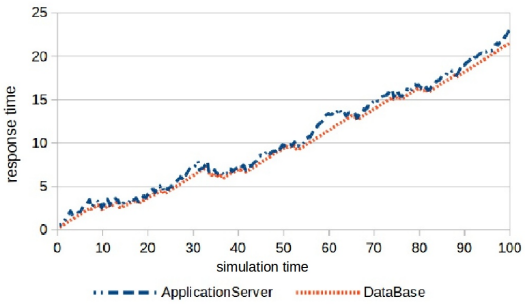
[1] Notice that due to their specificities, the FrontEnd and Controller components were modelled with no CPU demand. Hence, we carry out the analysis of the quality metrics regarding the impact of the workflow evolution on the ApplicationServer and DataBase components.



(A) Resource usage queue without adaptation actions



(B) Resource usage without adaptation actions



(C) Response time without adaptation actions

FIGURE 6.12: Resource usage queue, resource usage and response time and without adaptation actions

LISTING 6.3: Component-level SYBL annotation 1 for ApplicationServer Component

```
1  @SYBL_ComponentContext(
2    ComponentID = Component3;
3    ComponentName = ApplicationServer;
4    constraints = "Co1: CONSTRAINT cpuUsageAPP < 65;
5                   Co2: CONSTRAINT cpuUsageAPP > 30;
6                   Priority(Co1) = 2,
7                   Priority(Co2) = 1,
8    monitoring = "Mo1: MONITORING cpuUsageAPP = ObResourceUsage",
9    strategies = "St1: STRATEGY WHEN Violated(Co1): ScaleUp;
10                  St2: STRATEGY WHEN Violated(Co2): ScaleDown)
```

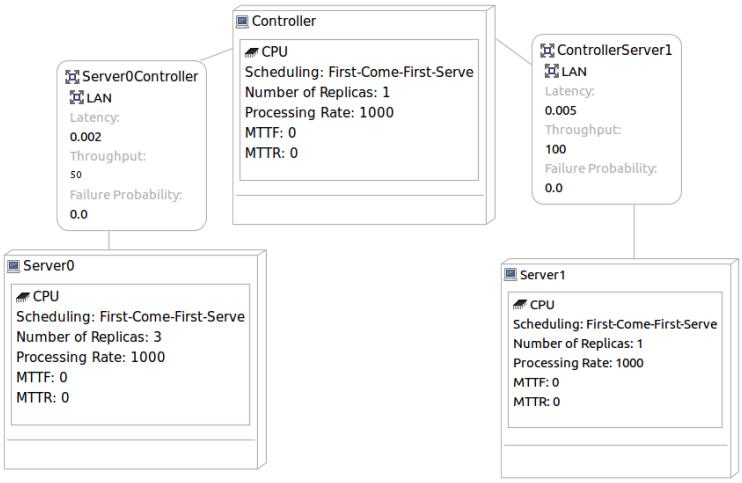


FIGURE 6.13: Resource Environment after Ann1

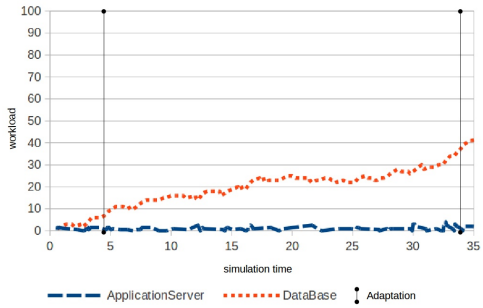
100% during the most part of the simulation. Therefore, despite the improvement, the response time values of both components stayed as the ones presented in the simulation without adaptations, which can be explained by the fact that the ApplicationServer component depends on the DataBase component to finish its execution. I.e., not improving the response time of the DataBase component means that the response time of the ApplicationServer component will not be altered. In the end of the simulation using Ann1, the only modified model was the resource environment. Figure 6.13 shows the final configuration of the model where the Server0 container ends 3 CPU resource replicas.

As the value of resource usage metrics is obtained from monitoring the use of the nodes on which each of these components is allocated, and as we use FCFS scheduling for the resource usage, this value considers the time in the resource usage queue plus the processing time (resource demand / node processing rate), the monitoring of the resource usage queues may bring some insight. Figure 6.14a shows that the resource usage queue of the node of the ApplicationServer component remains most of the time below 5 in the adapted scenario operated by Ann1. The resource usage queue of the node that allocates the DataBase component remains growing in both scenarios, which shows a bottleneck yet. We considered that, to improve the response time of our application, adaptations that focus on the DataBase component should be a priority.

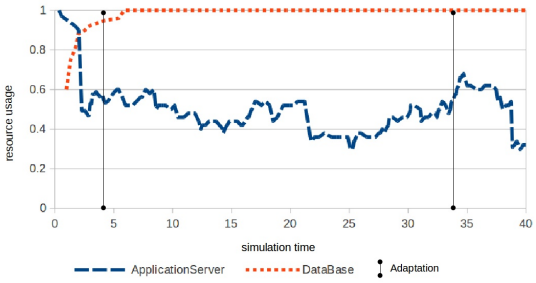
Analysis with Ann2: increasing the amount of nodes of the DataBase

Annotation Ann2 (Listing 7.2) specifies an adaptation on the DataBase component considering its response time, which refers to the delay time of a work to be concluded by a component, from its request until its response. In accordance to the Ann2 specification, the adaptations occur when the average in the response time monitor is below 0.2 t.u. (scale in) or above 0.5 t.u. (scale out).

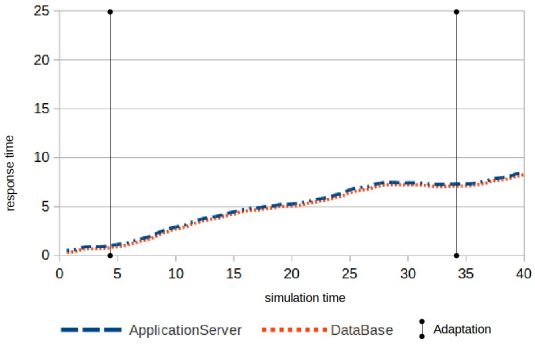
Figure 6.15c shows the response time average of the components ApplicationServer and DataBase in a simulation with the Ann2 adaptation specification. We observe the application of scale out adaptation operations at times 2.3 and 7.3. After these operations, the response time for both components decreased significantly, stabilizing with the ideal average of 0.3 t.u. for the DataBase component and varying between



(A) Resource usage queue with Ann1



(B) Resource usage with Ann1

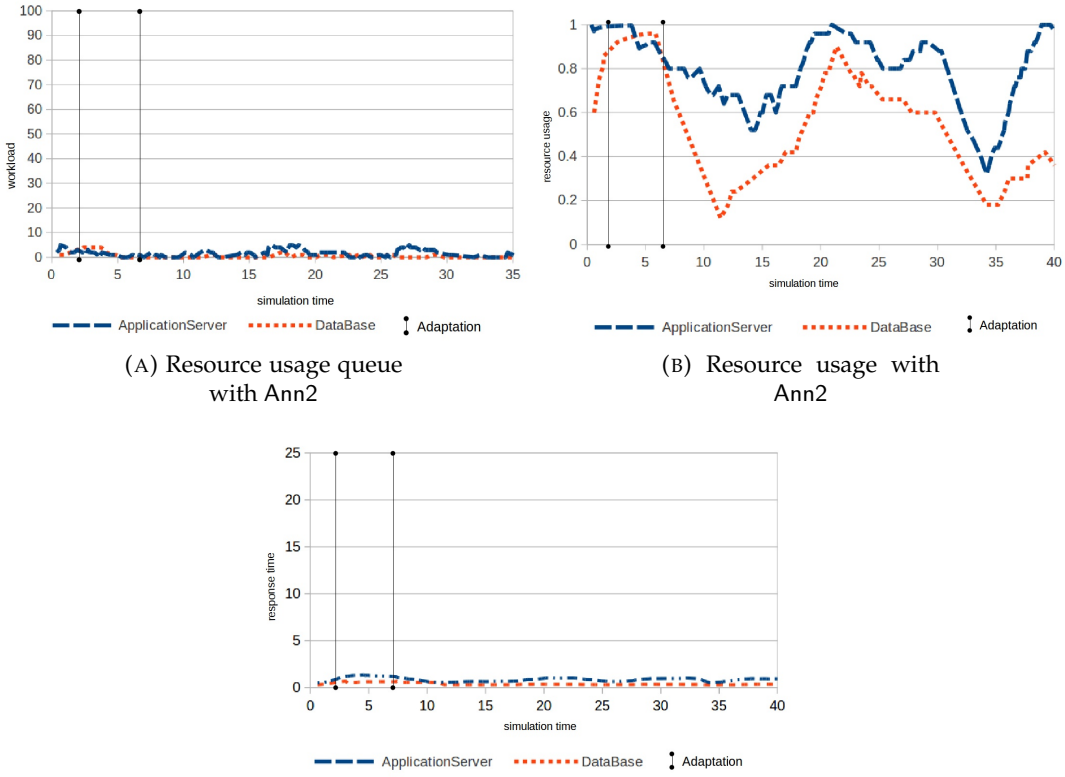


(C) Response time with Ann1

FIGURE 6.14: resource usage queue, resource usage and response time with Annotation Ann1

LISTING 6.4: Component-level SYBL annotation 2 for DataBase Component

```
@SYBL_ComponentContext(
  ComponentID = Component4;
  ComponentName = DataBase;
  constraints = "Co3: CONSTRAINT rt < 0.51;
                 Co4: CONSTRAINT rt > 0.21;
                 Priority(Co3) = 2,
                 Priority(Co4) = 1",
  monitoring = "Mo3: MONITORING rt = ObResponseTime",
  strategies = "St3: STRATEGY WHEN Violated(Co3): ScaleOut;
                St4: STRATEGY WHEN Violated(Co4): ScaleIn")
```



(A) Resource usage queue with Ann2

(B) Resource usage with Ann2

(C) Response time with Ann2

FIGURE 6.15: Resource usage queue, resource usage and response time and with Annotation Ann2

1 t.u. and 0.5 t.u. for the ApplicationServer component. The adaptation occurred only on the DataBase component, nevertheless, we notice the impact in the whole system. The impact in the resource usage in both components was also observed, as depicted in Figure 6.15b. The resource usage average in the nodes that allocate the DataBase component dropped below 60%. Despite the lack of stability, the results are promising, and show that an increase in the number of nodes that allocate the components offers a significant improvement in both quality metrics. When looking at the resource queues, we observe a decrease for both components: the ApplicationServer component presented an average of two works in the queue and the DataBase component presented an average of one work in the queue (Figure 6.15a).

In the end of the simulation using Ann2 four models were modified. Figure 6.16 shows the final configuration of the resource environment, where two new nodes were added. The same values for the CPU resources and a communication link was created for each new node (so that the link would not interfere with performance), but it is possible to both create new nodes with different configurations and share the communication link with other containers. The new node inserted reflects in the assembly context. Figure 6.17 shows the Assembly Model after simulation with Ann2, where two new assemblies context were added. Now, from the controller, the workflow can go to one of the assembly contexts. In the component model (Figure 6.18) two «Requires» are added from Controller to IDataBase (DataBase component interface). In the SEFF of the Controller component (Figure 6.18b), two probability branch are added and a new probability setting is established (now each DataBase

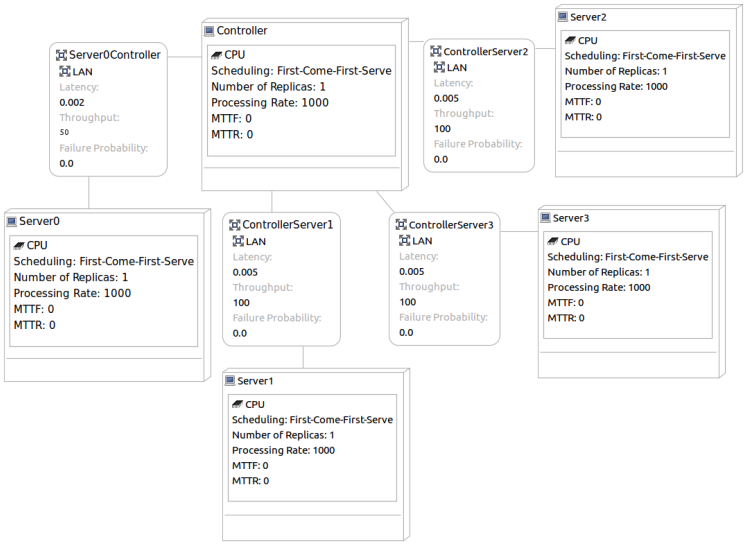


FIGURE 6.16: Resource Environment after Ann2

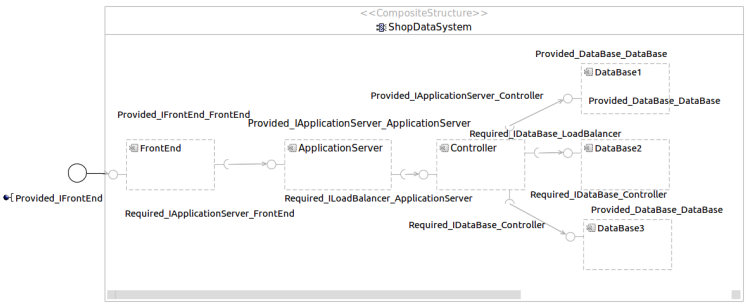


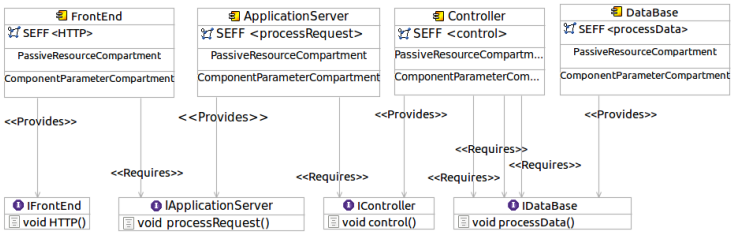
FIGURE 6.17: Assembly Model after Ann2

component node has a 33% probability of being accessed). Finally, the Figure 6.19 shows what the new allocation model looks like.

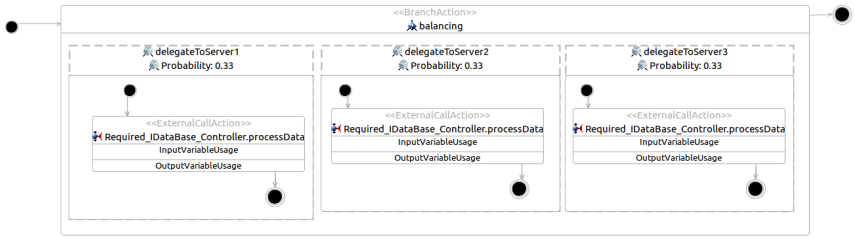
Analysis with (extra) Ann3: modifying the number of replicas in the DataBase's node

Figure 6.20 presents the resource usage queue and response time metrics with another scale to look in more detail at changes in metric values. Although there has been an improvement in the response time using Annotation Ann2 (Section 6.3.3), the response time of the application averaged 0.89 t.u., which is above what is specified in the non-functional requirements. To see how to improve this time, we have specified a (extra) annotation (Listing 6.5) – Annotation 3 (Ann3) – which inserts new nodes with instances of the DataBase component, operates scale-up adaptations on the DataBase component considering the response time.

Figure 6.21 shows the results obtained with the adjusted annotation. As we can observe, it brings an earlier stability in the response time of the DataBase component, and an average of 0.71 t.u. in the application response time. The response time with this SYBL annotation now satisfies the given requirements. The resource usage queue, which had already performed well on Ann2, dropped further and resource usage, which had also performed well on Ann2, was at 68% in the DataBase component node.



(A) Component Model after Ann2



(B) Branch Action of the Controller SEFF after Ann2

FIGURE 6.18: Component Model and Branch Action of the Controller SEFF after Ann2

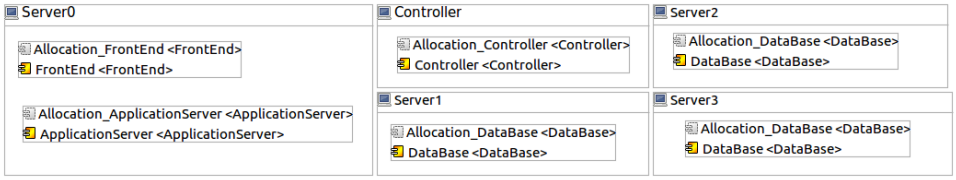
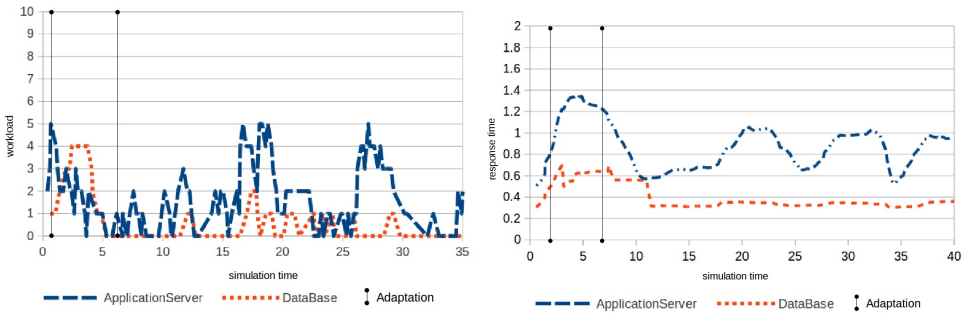


FIGURE 6.19: Allocation Model after Ann2

LISTING 6.5: Component-level SYBL annotation 3 for DataBase Component

```
1
2  @SYBL_ComponentContext(
3    ComponentID = Component4;
4    ComponentName = DataBase;
5    constraints = "Co3: CONSTRAINT rt < 0.71;
6                   Co4: CONSTRAINT rt > 0.21;
7                   Priority(Co5) = 2,
8                   Priority(Co6) = 1",
9    monitoring = "Mo3: MONITORING rt = ObResponseTime",
10   strategies = "St1: STRATEGY WHEN Violated(Co5): ScaleUp;
11                  St2: STRATEGY WHEN Violated(Co6): ScaleDown")
```



(A) Resource usage queue with Ann2 with scale 10

(B) Response time with Ann2 with scale 2

FIGURE 6.20: Resource usage queue and response time with Annotation Ann2 with other scales

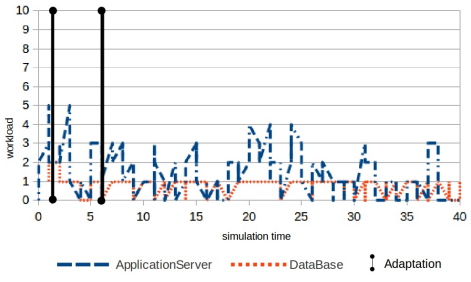
In the end of the simulation using the (extra) Ann3, the only modified model was the resource environment. Figure 6.22 shows the final configuration of the model where the Server1 container ends 4 CPU resource replicas.

Conclusions from our Analysis

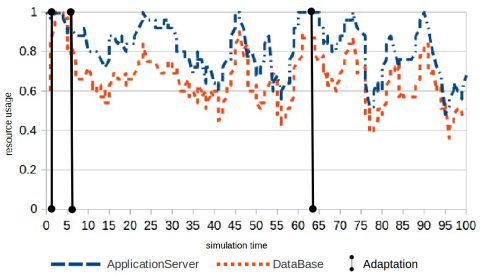
We could continue with the experiments, but with the three annotations we were able to define a final architectural model for our application. With the analysis of the simulations we can establish the conclusions listed below and define a new Annotation (Listing 6.6), containing all the possible adaptations for this application.

- The scale up adaptation strategy is added on the node where the ApplicationServer component is allocated when the CPU usage is greater than 65%;
- Adaptations in the node in which the DataBase component is allocated have priority over adaptations in the node in which the ApplicationServer component is allocated;
- Considering the average CPU usage obtained in the experiment with the adjusted specification, the new limit for the use of CPU on the nodes where DataBase component is allocated is 68%;
- Considering the average response time obtained in the experiment with the adjusted specification, the new response time limit for the application is 0.71 t.u;
- The modeled communication channel did not cause any kind of interference in the application's response time, not needing to apply any kind of adaptation using metrics related to it.

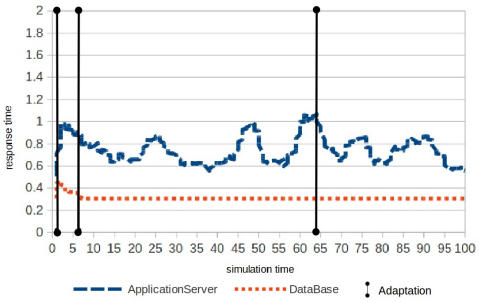
Figure 6.22 shows the proposed architecture model for the example application presented. The simulations showed that, in relation to the change in the initial model, two increments in the number of repetitions in the CPU resource of the nodes were sufficient in both cases (ApplicationServer and DataBase). These adaptations, combined with at least one more node, will bring good quality of service results for the example presented here. Thus, Figure 6.23 presents the final suggested resource environment for the example application. The assembly, component (and Controller SEFF) and allocation models are shown, respectively, in the Figures 6.24, 6.25 and 6.26.



(A) Resource usage queue with scale up adaptation



(B) Resource usage with scale up adaptation in DataBase component



(C) Response time with scale up adaptation in DataBase component

FIGURE 6.21: Resource usage queue, resource usage and response time in both ApplicatiomServer and DataBase components with scale up adaptation in DataBase component

The simulation and analysis procedure proposed and presented here proved to be useful to verify adjustments, initial configurations and adequate types of adaptation for the application being designed.

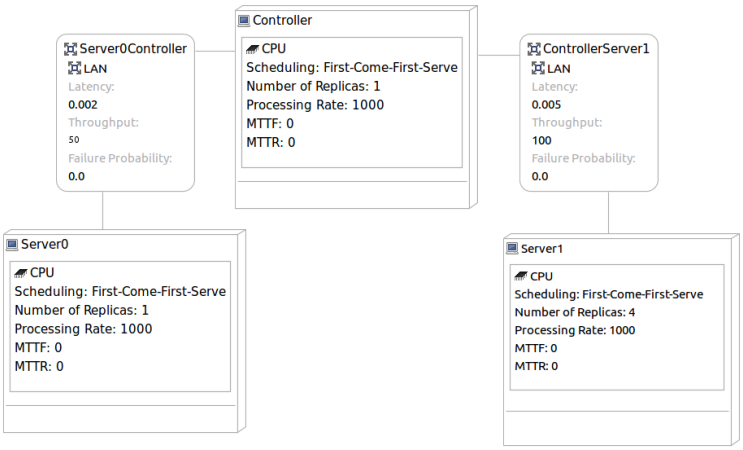


FIGURE 6.22: Resource Environment after Ann3

LISTING 6.6: Component-level SYBL annotation for Application-Server and DataBase Component after Simulation and Analysis

```
1
2  @SYBL_ComponentContext(
3    ComponentID = Component3;
4    ComponentName = ApplicationServer;
5    constraints = "Co1: CONSTRAINT cpuUsageAPP < 65;
6                   Co2: CONSTRAINT cpuUsageAPP > 30;
7                   Priority(Co1) = 6,
8                   Priority(Co2) = 5,
9    monitoring = "Mo1: MONITORING cpuUsageAPP = ObResourceUsage",
10   strategies = "St1: STRATEGY WHEN Violated(Co1): ScaleUp;
11                  St2: STRATEGY WHEN Violated(Co2): ScaleDown)
12
13 @SYBL_ComponentContext(
14   ComponentID = Component4;
15   ComponentName = DataBase;
16   constraints = "Co3: CONSTRAINT rt < 0.71;
17                  Co4: CONSTRAINT rt > 0.51;
18                  Co5: CONSTRAINT cpuUsageDB < 68;
19                  Co6: CONSTRAINT cpuUsageDB > 30;
20                  Priority(Co3) = 3,
21                  Priority(Co4) = 1,
22                  Priority(Co5) = 4,
23                  Priority(Co6) = 2",
24   monitoring = "Mo3: MONITORING rt = ObResponseTime",
25                "Mo4: MONITORING cpuUsageDB = ObResourceUsage",
26   strategies = "St1: STRATEGY WHEN Violated(Co5): ScaleUp;
27                  St2: STRATEGY WHEN Violated(Co6): ScaleDown;
28                  St3: STRATEGY WHEN Violated(Co3): ScaleOut;
29                  St4: STRATEGY WHEN Violated(Co4): ScaleIn")
```

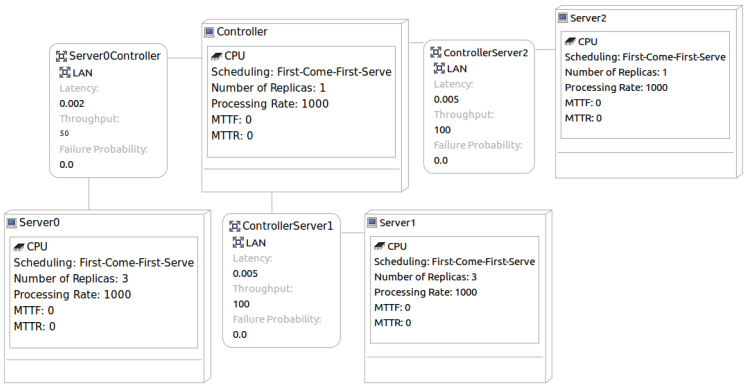


FIGURE 6.23: Resource Environment after Simulation and Analysis

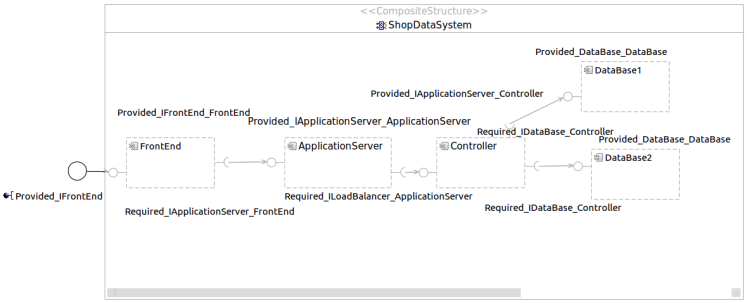
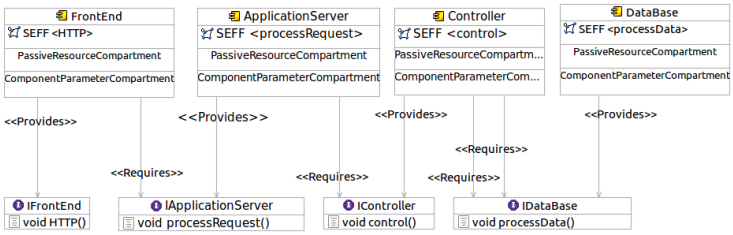
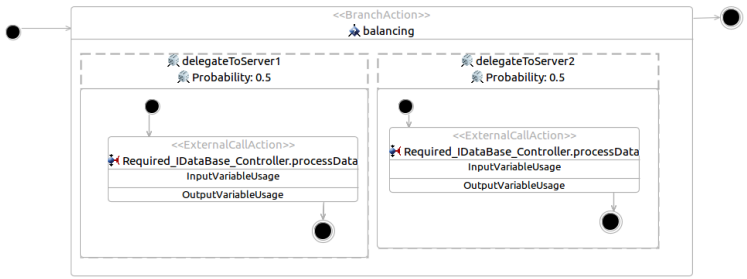


FIGURE 6.24: Assembly Model after Simulation and Analysis



(A) Component Model after Simulation and Analysis



(B) Branch Action of the Controller SEFF after Simulation and Analysis

FIGURE 6.25: Models after Simulation and Analysis

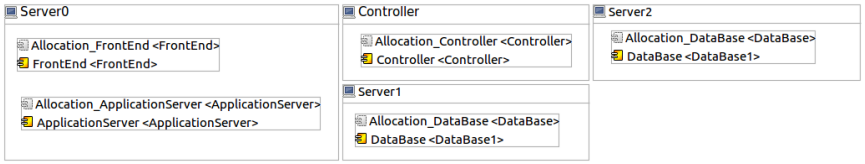


FIGURE 6.26: Allocation Model after Simulation and Analysis

7 Flexible System Modeling: An Example of Modeling of an Application for Smart Cities

One of the advantages of our approach is the flexibility regarding the possibility of modeling different behaviors, adaptations and metrics. This is possible due to the integration of different tools during the specification process. In this chapter, we will illustrate how the modeling process based on functional and non-functional requirements takes place using these tools applied to a very dynamic scenario which involves Cloud and IoT services. In fact, we will model and define the behavior and adaptation mechanisms of an application in the context of Smart Cities.

7.1 Understanding the Application Domain

In applications for the context of Smart Cities, large volumes of data can be a "threat" to their performance. Thus, the proposal to use heterogeneous architectures brought a way to dynamically manage Smart City applications and consider the use of different technologies. The use of Cloud Computing and Edge Computing can compose a heterogeneous architecture that seeks, in a self-adaptive way, to overcome possible performance problems of this type of application.

The Cloud dynamics brings numerous benefits, among them we can highlight its elasticity, which allows self-adaptation on demand. However, Edge architectures still need to be more dynamic, just like Cloud architectures, and to allow dynamic access to IoT devices to maintain good system performance. In a Smart City application that considers a heterogeneous architecture, IoT devices (or Edge devices) can move, connect, disconnect and send information to Edge servers and Cloud servers.

Modeling an application that uses this heterogeneous architecture is a challenging task, but the possibility of modeling and analyzing its performance at design-time, considering its dynamism, would be quite advantageous, since the complexity of the architecture does not allow us to carry out sufficiently concise tests (and to consider possible adaptations) before their implementation.

7.1.1 Domain properties

The use of Edge Computing, expanding a Cloud Computing environment, has been pointed out as an alternative to overcome several challenges of Internet of Things (IoT) applications (Nardelli et al., 2017), due to the fact that Edge nodes are closer

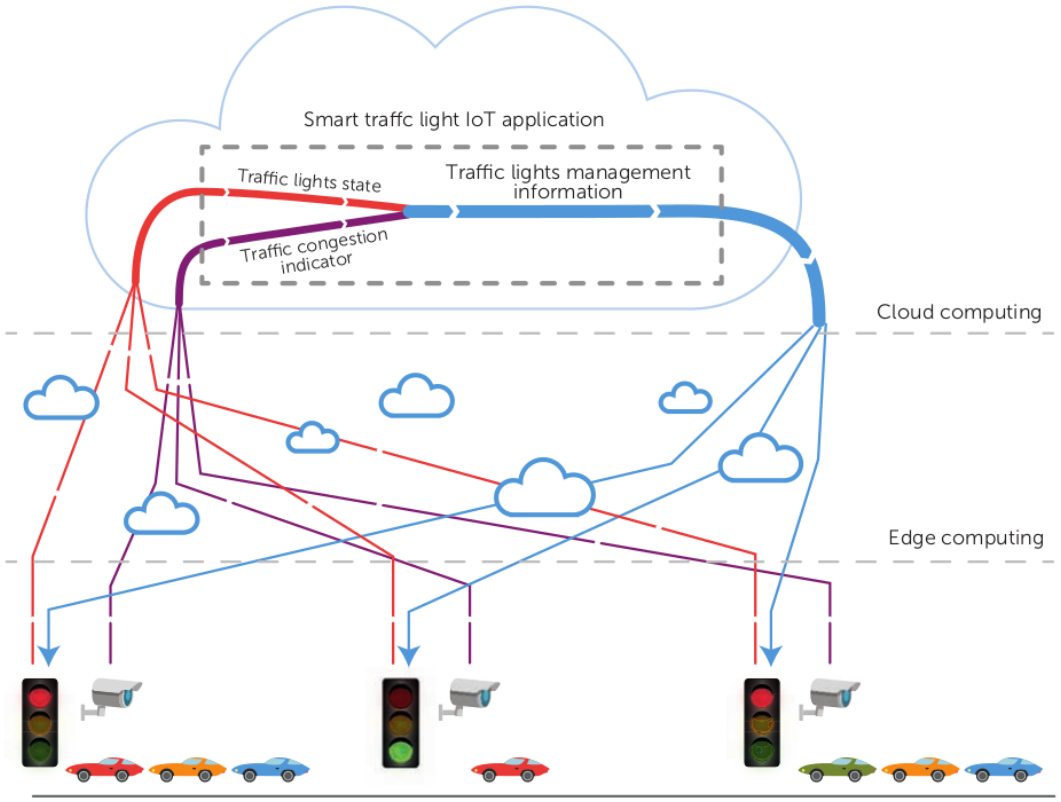


FIGURE 7.1: A high-level description of the smart traffic light IoT workflow application. Extracted from Nardelli et al., 2017

to the IoT devices, reducing latency, and due to the fact that they may be on the user's administrative premises, making the environment more secure. In this sense, systems models and architectures for IoT must consider complex and heterogeneous infrastructures capable of communicating with each other and adapting to different needs.

Considering that one of the essential aspects of IoT applications is the communication medium in which they are inserted, in IoT applications it is also necessary to consider the configuration and management of network services. Techniques such as SDN (Software-Defined Networking) and VNF (Virtual Network Functions) can allow network service providers to use software to install, configure and manage network services automatically and dynamically.

Considering these specificities, we will propose a scenario considering several technologies based on the example presented in the article by Nardelli et al., 2017, shown in 7.1. In the example we can identify two Edge elements, servers (edge computing) and IoT devices (edge device).

From the example presented, we developed a scheme to represent the scenario. Figure 7.2 presents the IoT scenario scheme that will help us to model the application in Palladio, in the construction of the behavior rules of the proposed technologies and in the definition of possible adaptations for this application domain. The flow of the scheme starts with IoT devices making requests to an SDN Switch. The Switch will have a controller (which will be modeled using behavior rules) that, in this first

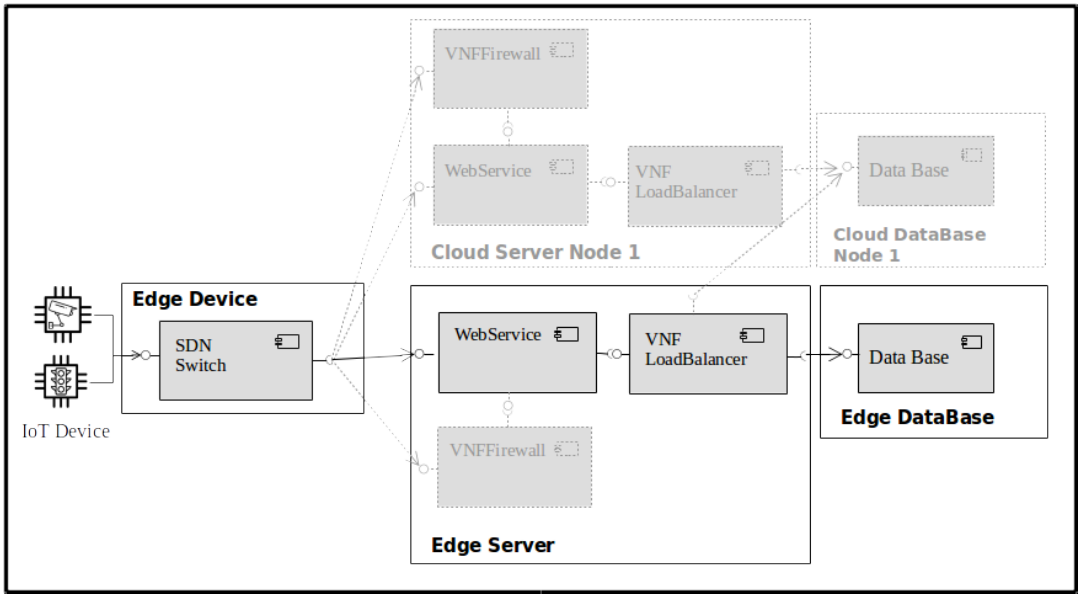


FIGURE 7.2: IoT Scenario Scheme

moment, will return the port to route. At the application modeling level, the flow to the defined port will be possible using Palladio's Guarded Branch.

Following the application flow, the Switch will make a request to the WebService, which can be on an Edge server or on a Cloud server. The choice of one of the flows will be done through adaptation rules using SYBL and the monitoring response will define the appropriate port for the Switch. New instances of the WebService can be created for both Edge and Cloud levels.

The application flow from the WebService can either go to a VNF Firewall or to a VNF Load Balancer. The flow will be defined using Palladio's Guarded Branch, behavior rules that model the VNF provider and the requirements defined in SYBL. The flow will go to a VNF Firewall if the SYBL definitions show the need to apply security policies in case - for example, if there is a sudden increase in the workload. If it is not necessary to go up the VNF Firewall, the flow will normally follow the VNF Load Balancer, which will probabilistically choose which Data Base to access - if there is more than one. The number of databases can increase or decrease if a scale out or scale in adaptation is made. The scheme suggests a starting scene in darker shades and solid lines and the possible scenarios in lighter shades and dotted lines.

7.1.2 Requirement and Adaptations Specifications

From the understanding of the domain and the definition of its properties, we can define the functional and non-functional requirements that will guide its modeling process and adaptation definitions.

Functional Requirements

We have defined the following functional requirements (FR1-FR4) which will be important to guide the behavior modeling process:

FR1: There will be an SDN Controller that must indicate to the SDN Switch the port for routing of incoming packets or must indicate the discard of the packets;

LISTING 7.1: Component-level SYBL annotation for Application-Server Component

```
@SYBL_ComponentContext(
  ComponentID = Component1;
  ComponentName = Switch;
  constraints = "Co1: CONSTRAINT rt < 1.0;
                 Co2: CONSTRAINT rt > 0.5;
                 Priority(Co1) = 2,
                 Priority(Co2) = 1,
  monitoring = "Mo1: MONITORING cpuUsageAPP = ObResponseTime",
  strategies = "St1: STRATEGY WHEN Violated(Co1): RouteToCloud;
                St2: STRATEGY WHEN Violated(Co2): RouteToEdge)
```

FR2: The Switch will have a flow table that will contain a queue of incoming works;
FR3: There will be a VNF provider, linked to the WebService, which will store the set of VNFs;
FR4: Initially there will be a Firewall type VNF and a Load Balancer type VNF;
FR5: The WebService should be monitored for the need to add security policies.

Non-Functional Requirements

We have defined the following non-functional requirements (NFR1-NFR4) which will be used as a basis for the SYBL annotation:

NFR1: Average response time must be less than or equal to 1.0 time unit (t.u.);
NFR2: If NFR1 is not met, that is, if the average response time is greater than 1.0 t.u., the system should restore an average response time of 1.0 t.u. or less as soon as possible;
NFR3: Resource usage queue must not exceed 5 works on average;
NFR4: If NFR3 is not met, that is,if the resource usage queue exceeds 5 works, the system should activate security policies as soon as possible.

Adaptations Specified at SYBL Annotation

Considering the NFR2 requirement, we will specify an annotation to guarantee this condition. For this case, we propose a route diversion adaptation, which will act on the Switch. The Switch initially sends the work to the Edge server. If the NFR2 condition is not met, it will divert the route to the Cloud server until the system returns to the condition imposed in the non-functional requirements. To achieve the NFR4 requirement, we propose the adaption of adding VNF Firewall and remove it when the requirement is satisfied again. Listings 7.1 and 7.2 show the SYBL Annotation resulting from these specifications.

7.2 Results

In this section we will present the modeling of the application and its scenario in Palladio, as well as the definitions of behaviors needed using graph transformation, and, finally, we will discuss how the model evolves with these definitions considering different perspectives.

LISTING 7.2: Component-level SYBL annotation for DataBase Component

```
@SYBL_ComponentContext(
  ComponentID = Component2;
  ComponentName = WebService;
  constraints = "Co3: CONSTRAINT queueWS < 5;
                 Co4: CONSTRAINT queueWS > 2;
                 Priority(Co3) = 2,
                 Priority(Co4) = 1,
  monitoring = "Mo3: MONITORING rt = ObResourcesUsage",
               St3: STRATEGY WHEN Violated(Co3): AddFirewall;
               St4: STRATEGY WHEN Violated(Co4): RemoveFirewall")
```

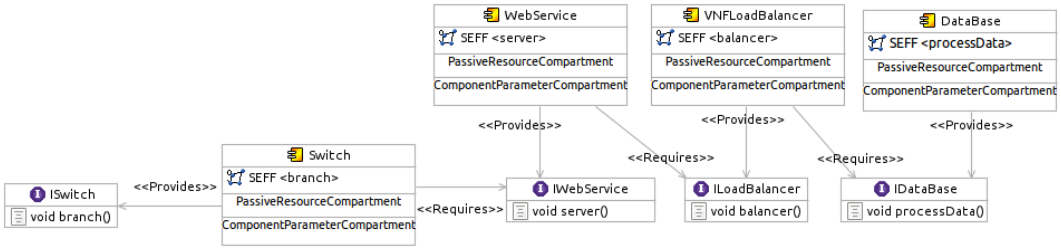


FIGURE 7.3: Initial IoT Palladio Repository

7.2.1 Modeling of the Application

Figure 7.3 represents the initial repository model, where we have four components: Switch, WebService, VNFLoadBalancer and DataBase. The Switch and WebService components have in their SEFFs a branch of the Guarded type (the example of a Guarded branch is illustrated in Figure 7.4), the VNFLoadBalancer component has in its SEFF a branch of the Probabilistic type (the example of a Probabilistic branch is illustrated in Figure 7.5) and the DataBase component has in its SEFF an Internal Action with 100 CPU units of resource demand. Figure 7.6 presents the application's composition structure, which has the following assembly contexts: SDN_Switch, Edge_WebService, Edge_VNF_LoadBalancer and Edge_Database.

7.2.2 Modeling of the Scenario

Figure 7.7 presents the initial resources model, where we have three containers, with their respective resource configurations and their communication links: EdgeDevice,

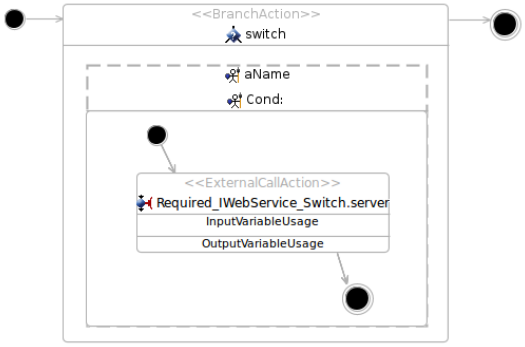


FIGURE 7.4: Initial IoT Palladio Switch Branch

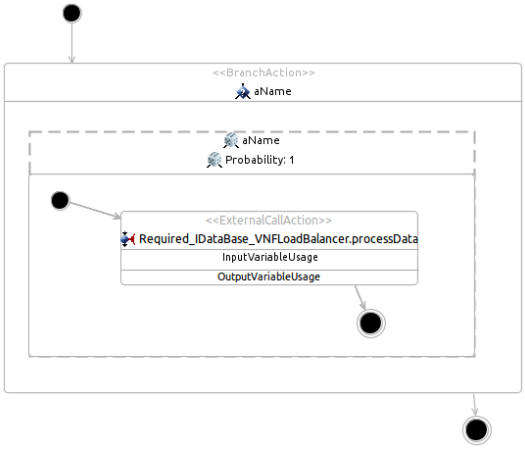


FIGURE 7.5: Initial IoT Palladio Load Balancer Branch

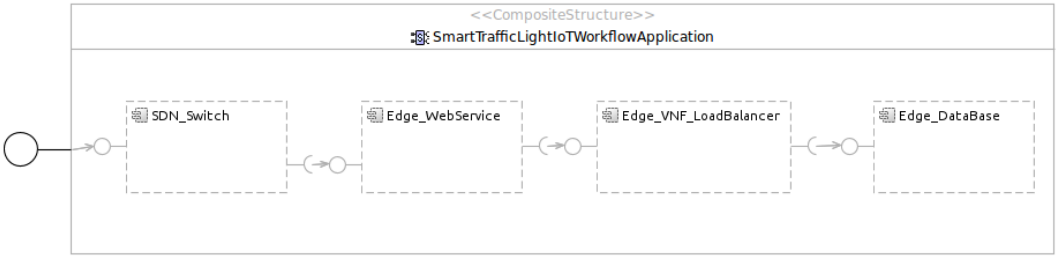


FIGURE 7.6: Initial IoT Palladio Assembly Context

EdgeServer and EdgeDatabase. Each of these containers allocates one or more components, as shown in Figure 7.8; the Edge Device container allocates the Switch component, the EdgeServer container allocates the Edge_WebService and Edge_VNF_LoadBalancer components and the EdgeDataBase container allocates the Edge_DataBase component.

7.2.3 Behavior Rules

Considering the specified functional requirements, a rule was defined to model the FR1 requirement, which consists of the behavior of an SDN controller, which indicates to the SDN Switch where to forward the work. For this, we will use the Guarded branch, modeled on Palladio in the Switch component. The rule checks the condition of each Guarded branch. These conditions are defined as the adaptation strategies, defined using SYBL. The rule checks which strategy is active to forward to one branch or another. The Switch object defined in the metamodel includes an

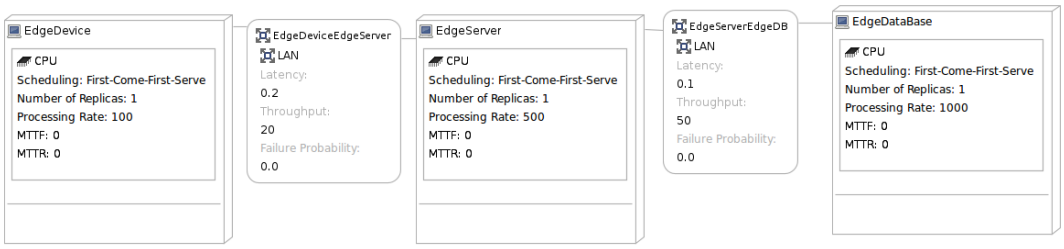


FIGURE 7.7: Initial IoT Palladio Resource Environment

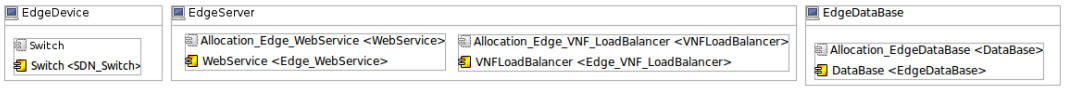


FIGURE 7.8: Initial IoT Palladio Allocation Context

attribute that corresponds to the flow table, which is linked to the resource usage observer and checks the observer's resource usage queue attribute, thus meeting the FR2 requirement. To meet the FR3 requirement, a "VNF provider" object was created in the metamodel, which is linked to the VNF object and the Palladio component of the WebService type. The created VNF object has an attribute "type" to guarantee the FR4 requirement. Finally, to fulfill the FR5 requirement, the object that corresponds to the WebService component must be linked to a monitor and SYBL annotations. In addition to the rules for meeting the defined requirements, for this application, we created the following rules: remove and add external action from a branch, add components and assign conditionals to a Guarded branch.

7.2.4 Model Evolution

In this section we will discuss three of the possible changes that can occur for the evolution of the initial model to illustrate the expressive power of our approach:

1. **Creation of a new component:** to illustrate the adaptation we will use the example of a VNF Firewall, which is activated according to security conditions in the Edge environment (Figure 7.9);
2. **Creation of load-balanced components:** we will illustrate an adaptation where, in an Edge environment, the VNF Load Balancer can choose to store data either in a Cloud Database or in an Edge Database (Figure 7.15);
3. **Creation of replicas of components:** we will show the adaptation of where the SDN Switch can choose to send a job to the Cloud environment or to the Edge environment (Figure 7.21).

Firewall

The first change we are considering is the activation of the Firewall (Figure 7.9), in which, instead of the Switch sending the requests directly to the WebService, due to an adaptation made, according to the specification, the works first go through a Firewall that will indicate whether to drop or follow the workflow. Figures 7.11, 7.10, 7.12, 7.13 and 7.14 show the changes made to the model when there is an adaptation to add the Firewall.

The SEFF of the Switch component (Figure 7.10) will have a branch that will forward the works to the VNFFirewall component. Figure 7.11) shows the Repository which presents the new component, the VNFFirewall, and its iFirewall interface. For this new component there will be an external action which will have the role of forwarding requests to the iWebService interface of the WebService component. The SEFF of the VNFFirewall component (Figure 7.12) will have a branch with security-related conditionals. If the security conditions are favorable, the works will be forwarded to the WebService component, otherwise, they will be discarded.

With the modifications made to the repository model, the assembly model (Figure 7.13) automatically changes, giving it an assembly context Edge_VNF_Firewall for the VNFFirewall component. In the same way, the allocation model (Figure 7.14)

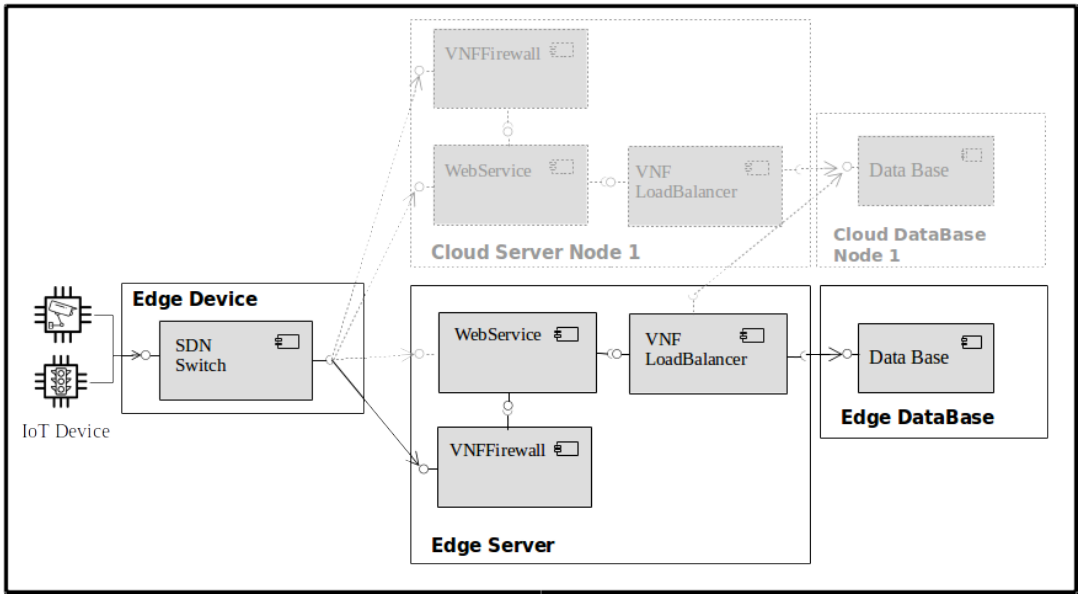


FIGURE 7.9: IoT Scenario Scheme With Firewall Edge

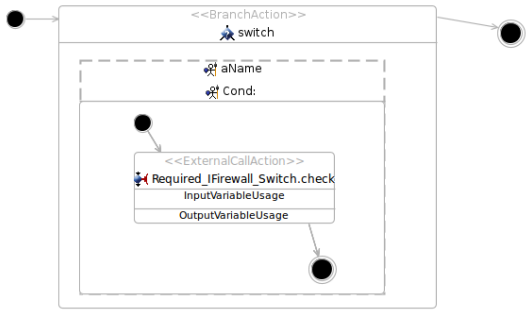


FIGURE 7.10: IoT Palladio Switch Branch with Firewall Edge

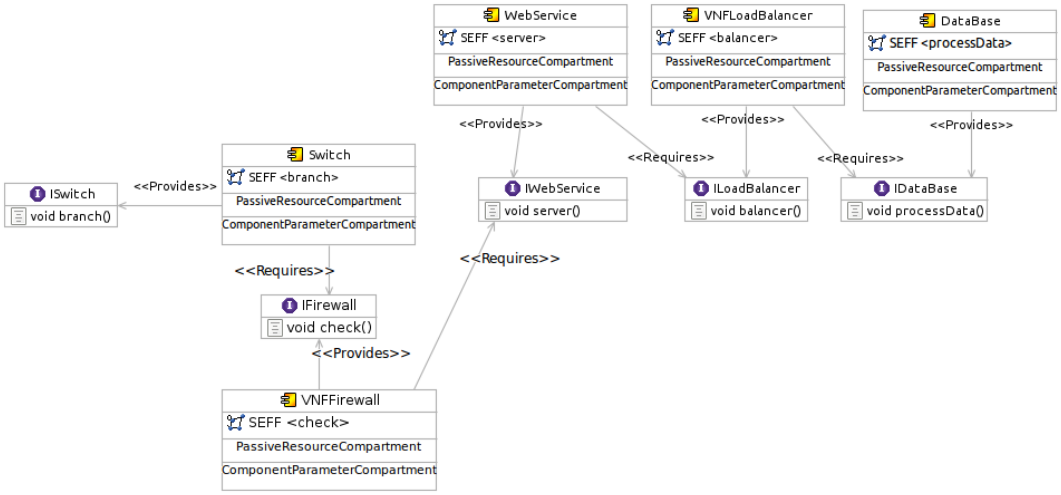


FIGURE 7.11: IoT Palladio Repository with Firewall Edge

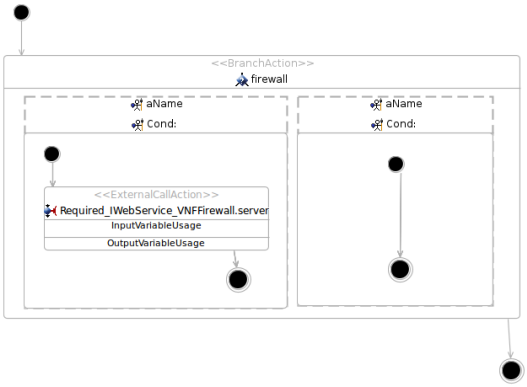


FIGURE 7.12: IoT Palladio Firewall Branch

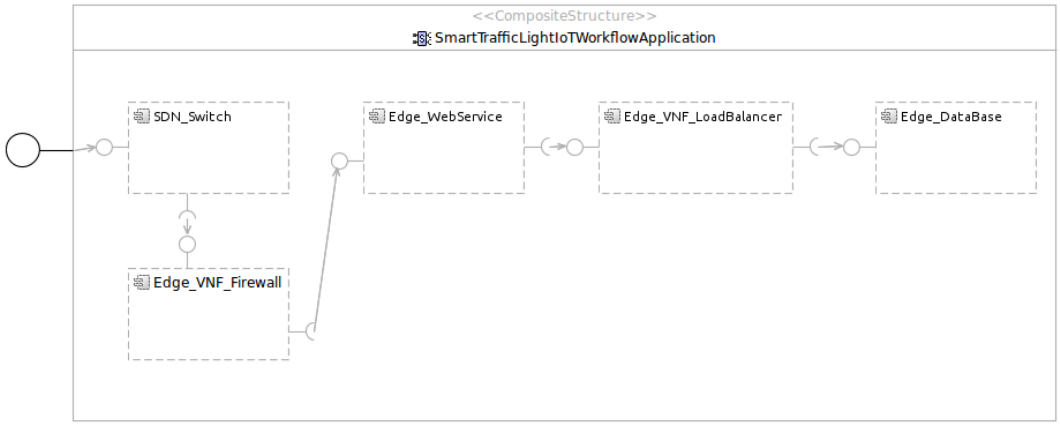


FIGURE 7.13: IoT Palladio Assembly Context with Firewall Edge

is modified, with the allocation of the VNFFirewall component in the EdgeServer container.

DataBase

We could also apply another adaptation, the Scale Out one (Figure 7.15), in which a DataBase is added to balance the load - this DataBase could be added in Cloud. Figures 7.16 , 7.17, 7.18, 7.19 and 7.20 present the changes made to the model in case a Scale Out adaptation occurred.

In addition to VNFFirewall component and its iFirewall interface, the DataBaseReplica component and its iDataBaseReplica interface were added to the repository (Figure 7.16), which receives requests from the VNFLoadBalancer component. With the addition of the DataBaseReplica component, the VNFLoadBalancer component now has two branches with a probability of 50% each, that is, 50% of the works will be

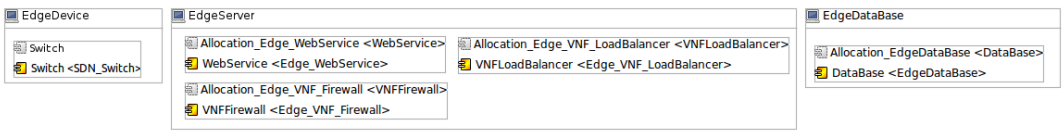


FIGURE 7.14: IoT Palladio Allocation Context with Firewall Edge

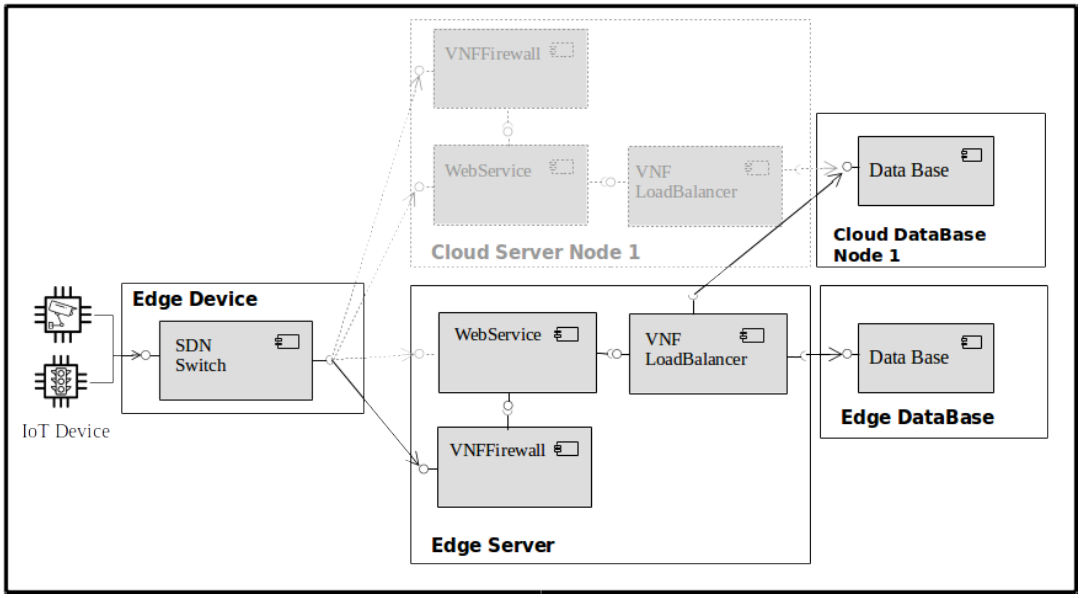


FIGURE 7.15: IoT Scenario Scheme 2 DB

sent to the DataBase component, in Edge, and other 50% will be sent to the new DataBaseReplica component , in Cloud.

With the modifications made to the repository model, the assembly model (Figure 7.18) automatically changes, giving it an assembly context Cloud_DataBase for the DataBaseReplica component. In the same way, the allocation model (Figure 7.19) and, in this case, the resource environment (Figure 7.20) are modified. A new Cloud-DataBase container is added, which will allocate the new DataBaseReplica component.

Edge and Cloud

Another possible change we can discuss in the model is when applying the adaptation specified in the Switch, making it start to route to the Cloud Server (Figure 7.21). Figures 7.22, 7.23, 7.24 and 7.25 show the changes made to the case model when the adaptation takes place.

In addition to the components already presented in the previous modifications, with these changes, the new repository (Figure 7.22) gains new components: VNFFirewallReplica component and its iFirewallReplica interface, WebServiceReplica and its iWebServiceReplica interface, and LoadBalancerReplica and its iLoadBalancer-Replica interface. With the addition of the VNFFirewallReplica component, the SEFF of the Switch component (Figure 7.23) has been modified and now has two branches: one for the VNFFirewall component and another one for the VNFirewall-Replica component.

The modifications of the repository model bring changes to the assembly model (Figure 7.24), which now has the assembly context of the new components: Cloud_VNF_Firewall, Cloud_WebService and Cloud_VNF_LoadBalancer. In the same way, the allocation model and also the resource environment (Figure 7.25) are modified. A new Cloud-Server container is added, which will allocate the replicas of the components.

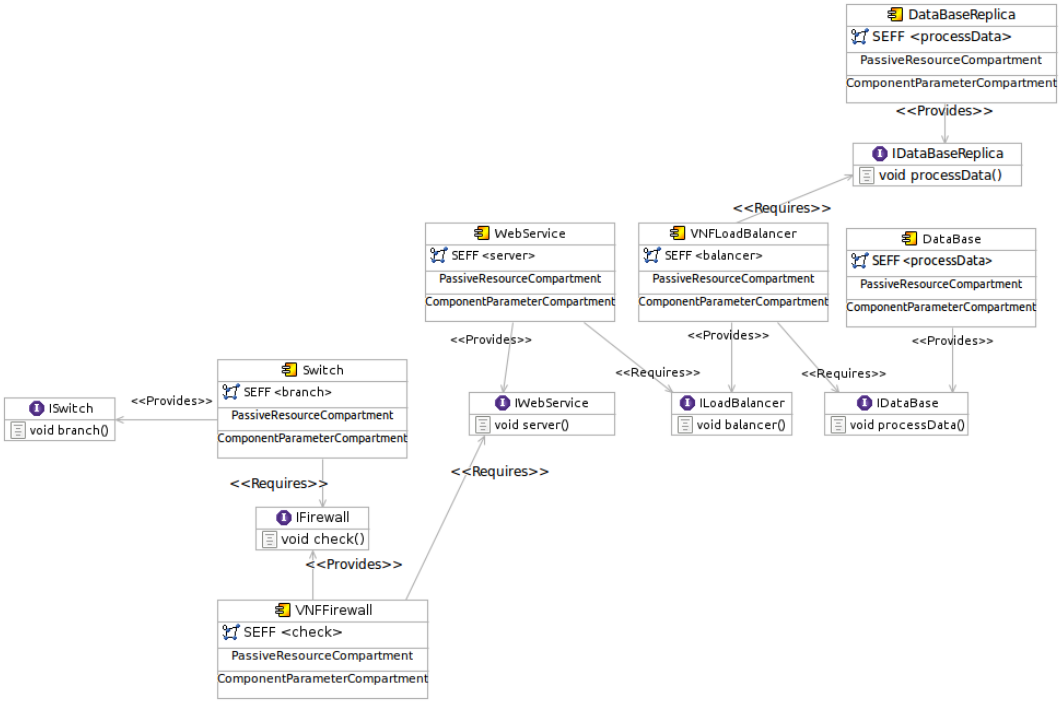


FIGURE 7.16: IoT Palladio Repository 2 DB

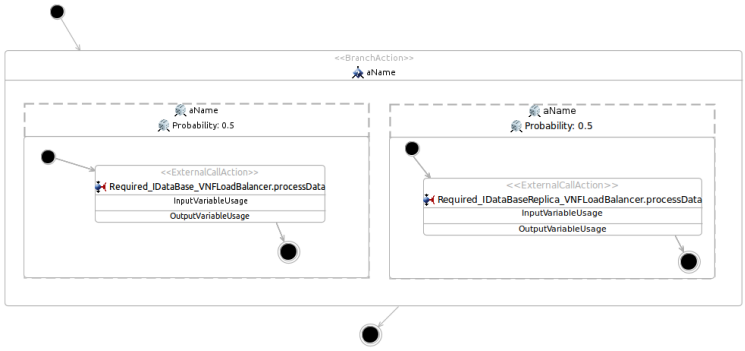


FIGURE 7.17: IoT Palladio Load Balancer Branch 2 DB

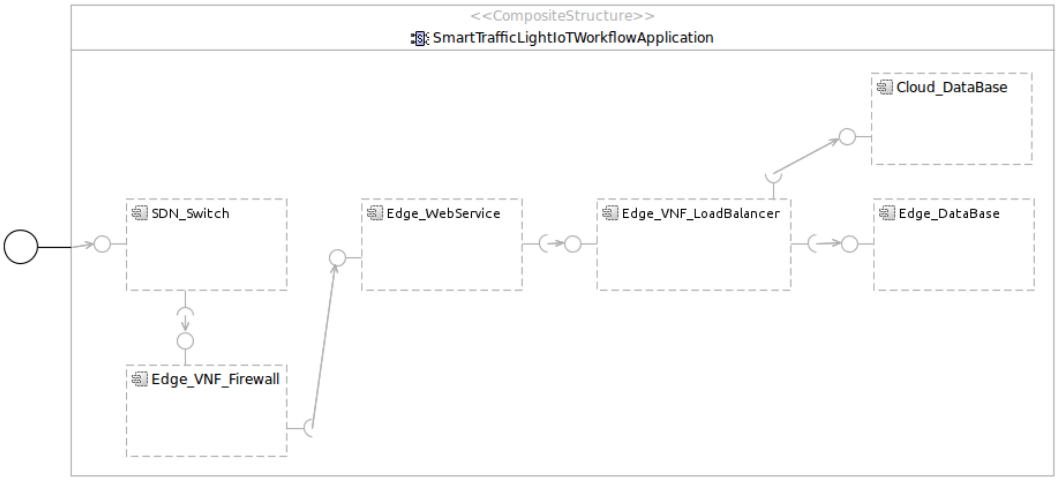


FIGURE 7.18: IoT Palladio Assembly Context 2 DB

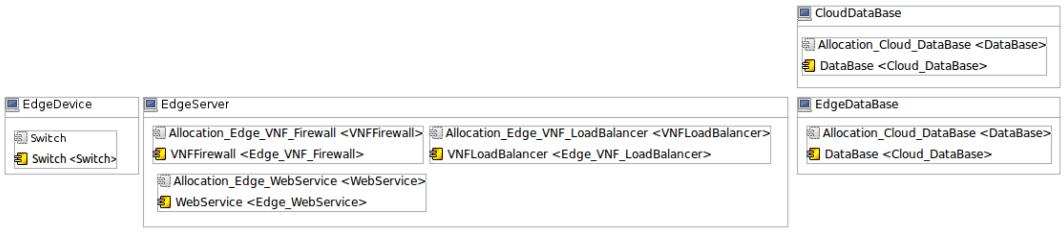


FIGURE 7.19: IoT Palladio Allocation Context 2 DB

FIGURE 7.20: IoT Palladio Resource Environment 2 DB

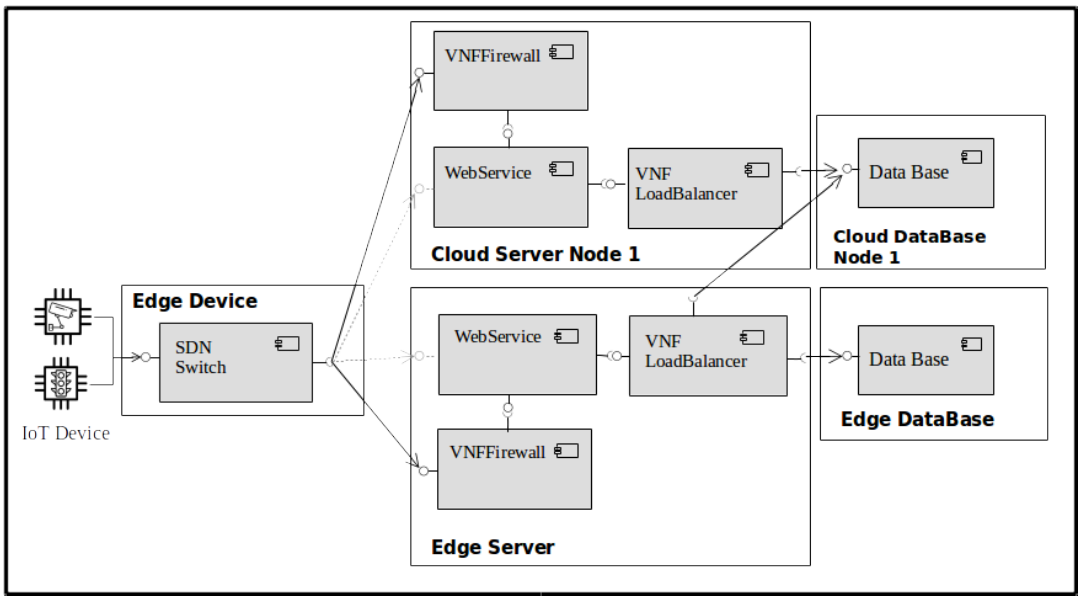


FIGURE 7.21: IoT Scenario Scheme Edge and Cloud

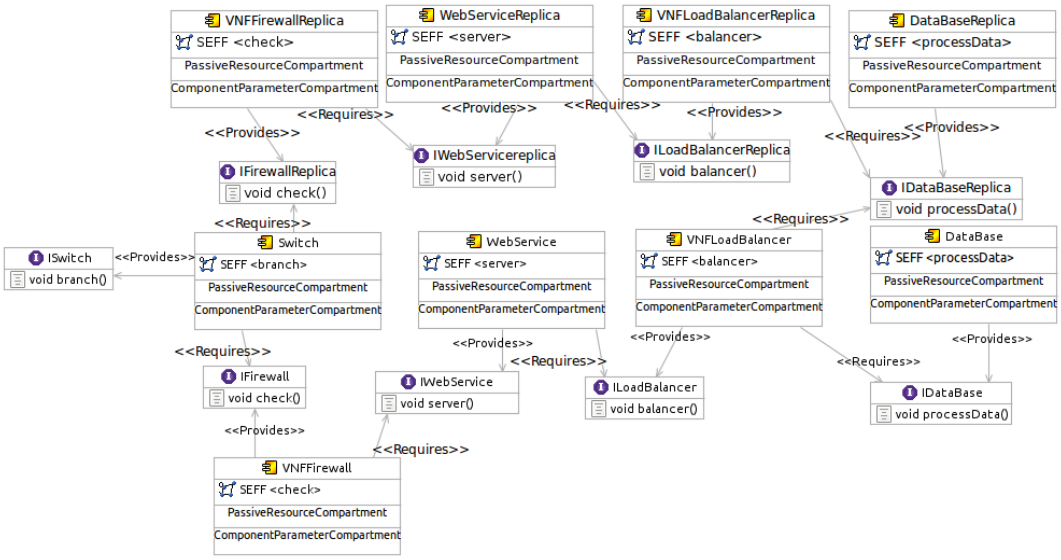


FIGURE 7.22: IoT Palladio Repository Edge and Cloud

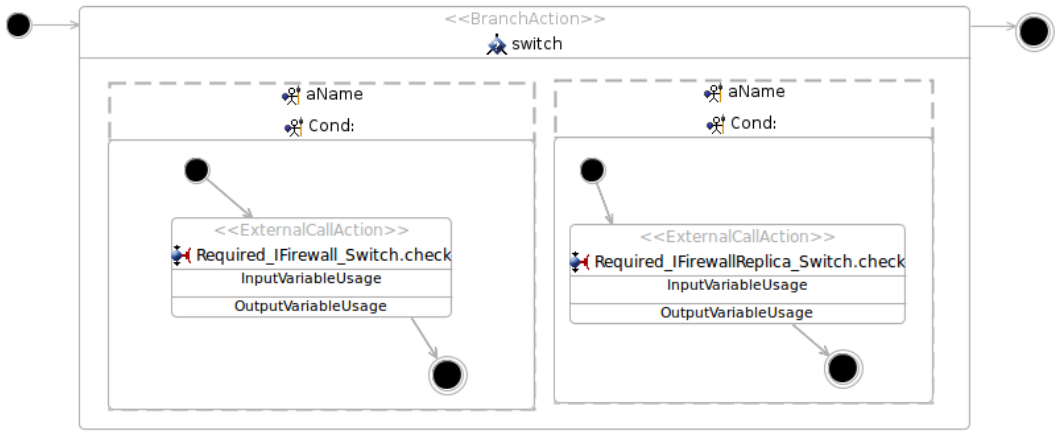


FIGURE 7.23: IoT Palladio Switch Branch Edge and Cloud

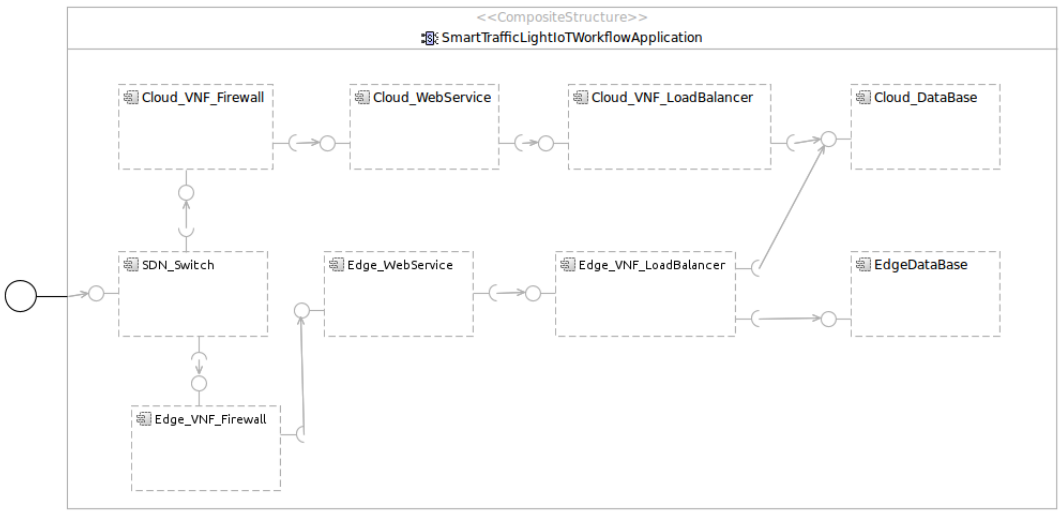


FIGURE 7.24: IoT Palladio Assembly Context Edge and Cloud

FIGURE 7.25: IoT Palladio Resource Environment Edge and Cloud

8 Discussion of the Obtained Results

We have presented a design-time approach that allows us to adjust and remodel our system specifications, their models, and even the behavior of systems, and can lead to performance prediction once the results presented in the design-time analysis are validated in real application runs.

Our approach allows us to conduct analysis of models of self-adaptive systems. This thesis clarifies several points to be considered in order to conduct modeling and analysis of models at design time that consider the dynamism of self-adaptive systems, that is, the need for dynamic change in the execution of analysis of models, and the need for readjustments and reanalysis of the models executed. To proceed with these steps, a structure capable of offering the possibility of modifying each stage of the design of this type of system is necessary.

Building adaptation rules and a metamodel that is capable of acting on the simulation of self-adaptive systems is not a trivial task: it requires time and development effort. However, once done, it can contribute to the advancement of research in the area, with the structure and initial modeling available. It is necessary to thoroughly analyse the behavior to be modelled and identify the elements necessary for the construction of each metamodel. This task can only be performed from a detailed survey of functionalities and real behavior, either to model a system, such as Palladio, or to model any type of adaptation, action or language that you want to include in a simulation.

To model a complete or even complex self-adaptive system is not a trivial task to be addressed at design time either. The approach presented in this thesis seeks to provide ways to conduct the analysis of system components.

This approach represents a significant contribution and a qualitative gain for practical applications. Considering that, by anticipating instruments and mechanisms of analysis and adaptation at design-time, it makes it possible to: (1) establish an understanding of the changing nature of self-adaptive systems through the modeling of services and their adaptations, also enabling the simulation of these models; and (2) establish guidelines capable of assisting in the construction of systems, facilitating the set of decisions at design-time, such as the specification of requirements that consider restrictions, the monitoring and adaptation strategies, as well as the modeling of the system and its behavior, and design-time analysis to build a model with better quality of service results.

The results of the simulations using our procedural and flexible approach showed us that it is possible to incorporate changes in models of adaptive systems and, mainly, that our approach is useful to model and measure the unpredictability of these systems.

With the Covid-19 pandemic, the use of cyber-physical systems has become essential for everyone's daily lives. More and more systems were used to make everyday life easier. What once consisted in solutions that were used by a small number of users became demands by a variety of customers, with even some solutions developed to meet new demands. An example of this was Brazil's financial aid policy, which was meant to mitigate the economic and social effects of the Covid crisis. The aid would be made accessible for citizens through two applications initially designed to serve 73 million people. Within weeks of the launch of the apps 95 million people requested sign up, and that number rose to 107 million a few months later (article available in: *Brazil Financial Aid Program*). However, the applications had problems due to the high demand for simultaneous access. The users had to wait on "virtual queues" (article available in: *Virtual Queues 'Caixa Tem'*) in order to access the "Caixa Tem" solution, since the developers had established a maximum access policy of 5,000 users per minute (article available in: *Access Policy 'Caixa Tem'*). Developing solutions that have the predictability of high demand can have benefits using self-adaptive policies that are modeled and analyzed at design time. With our approach, applications with this characteristic can not only reach the feasibility of foreseen policies, but also verify unpredictability that can cause system unavailability, slowdowns and long waiting time.

With the advent of the Smart Cities proposal, it is envisioned that solutions for cities are accessed by a large number of users and that a large volume of data is generated daily. These characteristics can bring problems related to the non-functional requirements of the solutions developed for this context. In an application for Smart Cities that considers a heterogeneous architecture, Internet of Things - IoT devices (or edge devices) can move, connect, disconnect and send information to both Edge and Cloud servers. With the approach presented here, it is possible: to specify (phase 1) the non-functional requirements and adaptation strategies as well as the dynamic context change (from Cloud to Edge or from Edge to Cloud as needed); to model a system using Palladio (phase 2), considering its component, container, allocation and communication link; to create rules in e-Motions (phase 3) that incorporate changes from an Edge context to a Cloud context where a high demand for a service can increase throughput impacting the latency of edge services or even create behavior rules with other paradigms such as SDN (Software-defined networking) and VNF (Network Functions Virtualization) to deal with the demands of communication between containers and services; and finally, to analyze the system architecture using Maude (phase 4).

9 Conclusions and Future Work

9.1 Summary and Conclusions

Designing software systems, which are becoming increasingly larger and more complex, has become a challenge due to the need to take into account not only the overall structure of the systems themselves, but also their allocation, functionality, and the communication of their components. In this regard, the Software Engineering community should concentrate efforts on building approaches that aim to establish the systems' architectures for their implementation, but that are also suitable to the analysis of their quality attributes, such as performance.

When considering the scalability, elasticity and adaptability of systems, dynamism and autonomy are challenging requirements, and considering them at design-time is still a difficult task. Emerging behaviors and opportunistic interactions cannot yet be predicted at design-time. However, these efforts should focus mainly on solutions related to the establishment of models that understand the changing nature of these systems in order for projects to model services and its adaptations more efficiently. These models must be tested so that it is possible to find guidelines capable of assisting in the construction of systems so that they mitigate delays and failures, and that do not directly impact the users' experience. These guidelines concern a set of decisions at design-time, such as the specification of requirements that consider constraints, monitoring and adaptation strategies, as well as the modeling of systems and their behavior, and the system analysis at design-time in order to build models with better quality of service results.

This work presented a proposal that uses Palladio, e-Motions, Maude and SYBL in such a way that it enables expressiveness and flexibility in the specification, modeling, behavioral definition and performance analysis of self-adaptive models using a procedural approach. Such procedural organisation has helped us to understand the formulation of each of the steps, which makes the approach flexible and consistently reproducible.

We model adaptation mechanisms as generic adaptation rules. We have illustrated our approach by modeling in/out scale and up/down scale rules, triggered in response to breaches of restrictions. We specify the elasticity requirements of the systems, allowing their adjustment. The verification of the specification and the adaptation in different components in the system allowed us to verify that previously

specified strategies may be misleading and our approach proposes that the readjustment and analysis be carried out at design-time. With the development of the communication channel behavior rules, it is possible to consider the latency variation and adaptations considering the communication link.

The procedure and the practical experience have allowed us to validate our hypothesis. The initial specifications, model and adaptations were tested at simulation time, and the results obtained through their use during a simulation were submitted for re-simulation and reanalysis, which led us to a more precise diagnosis of the system and, eventually, to a readjustment towards better results. This was possible thanks to the possibility of modifying the model during simulation time, the facilitation of the writing of the requirements specifications (constraints, monitoring and strategies) using a simple language like SYBL, and the subsequent analysis of the impact of the adaptations on the system.

9.2 Publications

The development of this thesis have given rise to the following publications (listed by category and in reverse chronological order):

Journal paper

- ARAÚJO-DE-OLIVEIRA, PATRÍCIA; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. A procedural and flexible approach for specification, modeling, definition, and analysis for self-adaptive systems. Softw Pract Exper. 2021;51:1387–1415. https://doi.org/10.1002/spe.2962

Book chapters

- DE OLIVEIRA, PATRÍCIA ARAÚJO; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. Towards the Performance Analysis of Elastic Systems with e-Motions. In: Cerone A., Roveri M. (eds) Software Engineering and Formal Methods. SEFM 2017. Lecture Notes in Computer Science, vol 10729, p. 475-490. Springer, Cham.
- DE OLIVEIRA, PATRÍCIA ARAÚJO; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. An Approach to Predictive Analysis of Self-Adaptive Systems in Design Time. In: Braubach L. et al. (eds) Service-Oriented Computing – ICSOC 2017 Workshops. ICSOC 2017. Lecture Notes in Computer Science, vol 10797, p. 363-368. Springer, Cham.

International conference

- DE OLIVEIRA, PATRÍCIA ARAÚJO. Predictive Analysis of Cloud Systems (Extended Abstract). In: International Conference on Software Engineering (Companion Volume), 2017: 483-484. Buenos Aires, Argentina.

Iberoamerican conference

- DE OLIVEIRA, PATRÍCIA ARAÚJO; MORENO-DELGADO, ANTÔNIO; DURÁN, FRANCISCO; PIMENTEL, ERNESTO. Towards the predictive analysis of cloud systems with e-Motions. In: Ibero-American Conference on Software Engineering, 2017: 169-182. Buenos Aires, Argentina.

Spanish conference

- DE OLIVEIRA, PATRÍCIA ARAÚJO.Towards the model-based predictive performance analysis of Cloud adaptive systems with e-Motions. In: Jornadas sobre PROgramación y LEnguajes (PROLE), 2017, Tenerife. Jornadas de la Asociación de Ingeniería del Software y Tecnologías de Desarrollo de Software (SISTEDES), 2017.

9.3 Future Work

We intend to advance the analysis of this approach by integrating the characteristics of Cloud, Fog and Edge Computing to create a model to simulate Smart Cities scenarios, advancing in the proposal for modeling SDNs and VNFs. For this, it is necessary to overcome technological limitations still present. The e-Motions tool leaves a significant part of the computation on the conditions of rules. Although this greatly simplifies the rules and their transformation, it results in a high computational price during the simulations. Given the way in which these rules are executed, a significant number of rule matches are discarded when evaluating such conditions. This may hamper the scalability of the proposal, and will need to be overcome if the tool is to be applied on complex systems.

This approach can advance even further in modeling the paradigms Software-Defined Networks (SDNs) or even the Virtualization of Network Functions (VNFs), but the field itself still has challenges to be overcome, such as the pursuit of ways to model an SDN that interacts with the model of a system in order to represent minimal behavior of a real network, or even, to model the behavior of a VNF in the best possible way, since this is an NP-difficult problem and several proposals can be presented.

Bibliography

Abuseta, Yousef and Khaled Swesi (2015). "Towards a framework for testing and simulating self adaptive systems". In: *6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. Ed. by M Surendra Prasad Babu and Wenzheng Li, pp. 70–76.

Access Policy 'Caixa Tem'. https://economia.uol.com.br/noticias/redacao/2020/05/31/auxilio-emergencial-r-600-caixa-tem-fila-de-espera.htm. Accessed: 2021-10-05.

Andries, Marc et al. (1999). "Graph transformation for specification and programming". In: *Science of Computer programming* 34.1, pp. 1–54.

Arcelli, Davide (2020). "Exploiting Queuing Networks to Model and Assess the Performance of Self-Adaptive Software Systems: A Survey". In: *Procedia Computer Science* 170, pp. 498 –505. DOI: 10.1016/j.procs.2020.03.108.

Balsamo, Simonetta et al. (2004). "Model-Based Performance Prediction in Software Development: A Survey". In: *IEEE Trans. Software Eng.* 30.5, pp. 295–310.

Becker, Matthias, Steffen Becker, and Joachim Meyer (2013). "SimuLizar: Design-Time Modeling and Performance Analysis of Self-Adaptive Systems". In: *Software Engineering* 213, pp. 71–84.

Becker, Steffen, Gunnar Brataas, and Sebastian Lehrig (2017). *Engineering Scalable, Elastic, and Cost-Efficient Cloud Computing Applications*. Springer.

Becker, Steffen, Heiko Koziolek, and Ralf Reussner (2009a). "The Palladio component model for model-driven performance prediction". In: *J. of Systems and Software* 82.1, pp. 3 –22.

– (2009b). "The Palladio component model for model-driven performance prediction". In: *Journal of Systems and Software* 82.1, pp. 3–22. ISSN: 0164-1212. DOI: 10.1016/j.jss.2008.03.066. URL: http://www.sciencedirect.com/science/article/pii/S0164121208001015.

Becker, Steffen, Heiko Koziolek, and Ralf H. Reussner (2007). "Model-Based Performance Prediction with the Palladio Component Model". In: *6th Intl. Workshop on Software and Performance (WOSP)*. Ed. by Vittorio Cortellessa, Sebastián Uchitel, and Daniel Yankelevich. ACM, pp. 54–65.

Bernardi, Simona et al. (2018). "Towards a model-driven engineering approach for the assessment of non-functional properties using multi-formalism". In: *Software & Systems Modeling*, pp. 1–24.

Bezerra, C. I. M. et al. (Sept. 2016). "DyMMer: a measurement-based tool to support quality evaluation of DSPL feature models". In: *20th International Systems and Software Product Line Conference (SPLC)*. Ed. by Hong Mei. ACM, pp. 314–317. DOI: 10.1145/2934466.2962730.

Bezerra, Carla I. M. et al. (2018). "Aggregating Measures Using Fuzzy Logic for Evaluating Feature Models". In: *12th International Workshop on Variability Modelling of Software-Intensive Systems*. Ed. by Rafael Capilla, Malte Lochau, and Lidia Fuentes. ACM, 35–42. DOI: 10.1145/3168365.3168375.

Brazil Financial Aid Program. https://feed.itsrio.org/brazils-auxílio-emergencial-and-caixa-tem-apps-6a4a5de68468. Accessed: 2021-10-05.

Bucchiarone, Antonio et al. (2015). "Rule-based modeling and static analysis of self-adaptive systems by graph transformation". In: *Software, services, and systems*. Ed. by Rocco De Nicola and Rolf Hennicker. Springer, pp. 582–601.

Business Process Model and Notation (BPMN) – Version 2.0. December 2011. OMG. URL: \url{https://www.omg.org/spec/BPMN/}.

Cao, Yinzhi and Junfeng Yang (2015). "Towards Making Systems Forget with Machine Unlearning". In: *2015 IEEE Symposium on Security and Privacy*, pp. 463–480. DOI: 10.1109/SP.2015.35.

Casimiro, Maria et al. (2021). "Self-Adaptation for Machine Learning Based Systems". In: *Proceedings of the 1st International Workshop on Software Architecture and Machine Learning (SAML)*. Springer.

Chen, B. et al. (2019). "Architecture-Based Behavioral Adaptation with Generated Alternatives and Relaxed Constraints". In: *IEEE Transactions on Services Computing* 12.1, pp. 73–87.

Ciccozzi, Federico, Antonio Cicchetti, and Andreas Wortmann (July 2020). "Editorial to theme section on interplay of model-driven and component-based software engineering". In: *Software and Systems Modeling* 19.6, pp. 1461–1463. DOI: 10.1007/s10270-020-00812-7. URL: https://doi.org/10.1007/s10270-020-00812-7.

Ciccozzi, Federico et al. (Mar. 2017). "Editorial to theme issue on model-driven engineering of component-based software systems". In: *Software & Systems Modeling* 18.1, pp. 7–10. DOI: 10.1007/s10270-017-0589-6. URL: https://doi.org/10.1007/s10270-017-0589-6.

Claus, Volker, Hartmut Ehrig, and Grzegorz Rozenberg (1979). *Graph-grammars and their application to computer science and biology: international workshop, Bad Honnef, October 30-November 3, 1978*. Vol. 1. Springer Science & Business Media.

Clavel, Manuel et al. (2007). *All About Maude*. Vol. 4350. Lecture Notes in Computer Science. Springer.

Combemale, Benoit et al. (Nov. 2016). *Engineering Modeling Languages: Turning Domain Knowledge into Tools*. Chapman and Hall/CRC, p. 398. URL: https://hal.inria.fr/hal-01355374.

Copil, Georgiana et al. (2013). "SYBL: An extensible language for controlling elasticity in cloud applications". In: *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Ed. by Pavan Balaji, Dick Epema, and Thomas Fahringer. IEEE, pp. 112–119.

Criado, Javier et al. (2016). "Exploring Quality-Aware Architectural Transformations at Run-Time: The ENIA Case". In: *6th International Conference on Model and Data Engineering (MEDI)*. Ed. by Ladjel Bellatreche et al. Vol. 9893. Lecture Notes in Computer Science. Springer, pp. 288–302. DOI: 10.1007/978-3-319-45547-1_23. URL: https://doi.org/10.1007/978-3-319-45547-1_23.

Criado, Javier et al. (2018). "Quality-aware Architectural Model Transformations in Adaptive Mashups User Interfaces". In: *Fundam. Inform.* 162.4, pp. 283–309. DOI: 10.3233/FI-2018-1726. URL: https://doi.org/10.3233/FI-2018-1726.

de Sousa, Amanda Oliveira et al. (2019). "Quality Evaluation of Self-Adaptive Systems: Challenges and Opportunities". In: *XXXIII Brazilian Symposium on Software Engineering (SBES)*. Ed. by Ivan Machado and Rodrigo Souza. ACM, 213–218. DOI: 10.1145/3350768.3352455.

Durán, Francisco, Antonio Moreno-Delgado, and José M. Álvarez-Palomo (2016). "Statistical Model Checking of e-Motions Domain-Specific Modeling Languages". In: *19th Intl. Conf. Fundamental Approaches to Software Engineering (FASE)*. Ed. by Perdita Stevens and Andrzej Wasowski. Vol. 9633. Lecture Notes in Computer Science. ETAPS. Springer, pp. 305–322.

e-Motions Examples. http://atenea.lcc.uma.es/projects/E-motions.html. Accessed: 2021-10-05.

Edwards, R. and N. Bencomo (2018). "DeSiRE: Further Understanding Nuances of Degrees of Satisfaction of Non-functional Requirements Trade-Off". In: *IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Ed. by Jesper Andersson and Danny Weyns, pp. 12–18.

Ehrig, Hartmut, Manfred Nagl, and Grzegorz Rozenberg, eds. (1983). *Graph-Grammars and Their Application to Computer Science*. Springer Berlin Heidelberg. DOI: 10.1007/bfb0000094. URL: https://doi.org/10.1007/bfb0000094.

Ehrig, Hartmut, Michael Pfender, and Hans Jürgen Schneider (1973). "Graph-grammars: An algebraic approach". In: *14th Annual Symposium on Switching and Automata Theory (swat 1973)*. IEEE, pp. 167–180.

Falkner, Katrina, Claudia Szabo, and Vanea Chiprianov (2016). "Model-driven performance prediction of systems of systems". In: *19th Intl. Conf. on Model Driven Engineering Languages and Systems*. Ed. by Jörg Kienzle and Alexander Pretschner. ACM/IEEE, pp. 44–44.

Franck, Reinhold (1976). "PLAN2D - Syntactic Analysis of Precedence Graph Grammars". In: POPL '76. Atlanta, Georgia: Association for Computing Machinery, 134–139. ISBN: 9781450374774. DOI: 10.1145/800168.811547. URL: https://doi.org/10.1145/800168.811547.

Franco, João M. et al. (May 2016). "Improving Self-Adaptation Planning through Software Architecture-Based Stochastic Modeling". In: *J. Syst. Softw.* 115.C, 42–60. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.01.026. URL: https://doi.org/10.1016/j.jss.2016.01.026.

Grassi, Vincenzo, Raffaela Mirandola, and Enrico Randazzo (2009). "Model-driven assessment of QoS-aware self-adaptation". In: *Software Engineering for Self-Adaptive Systems*. Ed. by B.H.C. Cheng et al. Springer, pp. 201–222.

Gu, Tianyu et al. (2019). "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks". In: *IEEE Access* 7, pp. 47230–47244. DOI: 10.1109/ACCESS.2019.2909068.

Habel, Annegret (1992). "Hyperedge replacement: grammars and languages". In:

Hannachi, Mohamed Amine et al. (2013). "GMTE: A Tool for Graph Transformation and Exact/Inexact Graph Matching". In: *Graph-Based Representations in Pattern Recognition*. Springer Berlin Heidelberg, pp. 71–80. DOI: 10.1007/978-3-642-38221-5_8. URL: https://doi.org/10.1007/978-3-642-38221-5_8.

Happe, Jens, Heiko Koziolek, and Ralf Reussner (2011). "Facilitating Performance Predictions Using Software Components". In: *IEEE Software* 28.3, pp. 27–33. ISSN: 0740-7459.

Heinrich, Robert (Feb. 2016). "Architectural Run-time Models for Performance and Privacy Analysis in Dynamic Cloud Applications". In: *SIGMETRICS Perform. Eval. Rev.* 43.4, pp. 13–22. ISSN: 0163-5999. DOI: 10.1145/2897356.2897359. URL: http://doi.acm.org/10.1145/2897356.2897359.

Himsolt, Michael (1990). "GraphEd: An Interactive Tool For Developing Graph Grammars". In: *Proceedings of the Fourth International Workshop on Graph-Grammars and Their Application to Computer Science*. Ed. by Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 532. Lecture Notes in Computer Science. Springer-Verlag, pp. 61–65. ISBN: 3-540-54478-X. DOI: 10.1007/BFb0017378.

Huang, Ling et al. (2011). "Adversarial machine learning". In: *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58.

Huber, Nikolaus et al. (2012). "S/T/A: Meta-modeling run-time adaptation in component-based system architectures". In: *9th Intl. Conf. on e-Business Engineering (ICEBE)*. Ed. by Kuo-Ming Chao. IEEE, pp. 70–77.

Johnsen, Einar Broch, Jia-Chun Lin, and Ingrid Chieh Yu (2016). "Comparing AWS Deployments Using Model-Based Predictions". In: *7th Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 9953. Lecture Notes in Computer Science. Springer, pp. 482–496.

König, Barbara and Vitali Kozioura (2005). "Augur - A Tool for the Analysis of Graph Transformation Systems". In: *Bull. EATCS* 87, pp. 126–137.

Koussaifi, Maroun et al. (May 2020). *Putting the End-User in the Loop in Smart Ambient Systems: an Approach based on Model-Driven Engineering*. Research Report IRIT/RR–2020–06–FR. IRIT - Institut de Recherche en Informatique de Toulouse. URL: https://hal.archives-ouvertes.fr/hal-03120776.

Koziolek, Heiko (2010). "Performance evaluation of component-based software systems: A survey". In: *Perform. Eval.* 67.8, pp. 634–658.

Krach, Sebastian Dieter and Max Scheerer (2018). "SimuLizar NG: An extensible event-oriented simulation engine for self-adaptive software architectures". In: *9th Symposium on Software Performance (SSP)*. Ed. by Holger Eichelberger and Klaus Schmid.

Löwe, Michael and Martin Beyer (1993). "Agg — An implementation of algebraic graph rewriting". In: *Rewriting Techniques and Applications*. Ed. by Claude Kirchner. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 451–456.

McIlroy, M Douglas et al. (1968). "Mass-produced software components". In: *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pp. 88–98.

Mian, Natash Ali and Farooq Ahmad (2017). "Modeling and Analysis of MAPE-K loop in Self Adaptive Systems using Petri Nets". In: *Int. J. Comput. Sci. Netw. Secur.* 17.12, pp. 158–163.

Miller, Brad et al. (2016). "Reviewer Integration and Performance Measurement for Malware Detection". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer International Publishing, pp. 122–141. DOI: 10.1007/978-3-319-40667-1_7. URL: https://doi.org/10.1007/978-3-319-40667-1_7.

Monshizadeh Naeen, Hossein, Esmaeil Zeinali, and Abolfazl Toroghi Haghighat (2020). "Adaptive Markov-based approach for dynamic virtual machine consolidation in cloud data centers with quality-of-service constraints". In: *Software: Practice and Experience* 50.2, pp. 161–183. DOI: 10.1002/spe.2764. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2764. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2764.

Moreno-Delgado, Antonio et al. (2014). "Modular DSLs for flexible analysis: An e-Motions reimplementation of Palladio". In: *10th European Conference on Modelling Foundations and Applications, ECMFA*. Ed. by Jordi Cabot and Julia Rubin. Vol. 8569. Lecture Notes in Computer Science. STAF. Springer, pp. 132–147.

Münch, Manfred (2000). "PROgrammed Graph REwriting System PROGRES". In: *Applications of Graph Transformations with Industrial Relevance*. Ed. by Manfred Nagl, Andreas Schürr, and Manfred Münch. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 441–448.

Nardelli, Matteo et al. (2017). "Osmotic Flow: Osmotic Computing + IoT Workflow". In: *IEEE Cloud Computing* 4.2, pp. 68–75. DOI: 10.1109/MCC.2017.22.

Palladio Simulator Website. https://www.palladio-simulator.com. Accessed: 2021-10-05.

Pfaltz, John L. and Azriel Rosenfeld (1969). "Web Grammars". In: *Proceedings of the 1st International Joint Conference on Artificial Intelligence*. IJCAI'69. Washington, DC: Morgan Kaufmann Publishers Inc., 609–619.

Pinto, Fábio, Marco O. P. Sampaio, and P. Bizarro (2019). "Automatic Model Monitoring for Data Streams". In: *ArXiv* abs/1908.04240.

Plasmeijer, Rinus, Marko Van Eekelen, and MJ Plasmeijer (1993). *Functional programming and parallel graph rewriting*. Vol. 857. Addison-wesley Reading.

Quionero-Candela, Joaquin et al. (2009). *Dataset Shift in Machine Learning*. The MIT Press. ISBN: 0262170051.

Rabanser, Stephan, Stephan Günnemann, and Zachary Lipton (2019). "Failing loudly: An empirical study of methods for detecting dataset shift". In: *Advances in Neural Information Processing Systems* 32.

Raibulet, Claudia et al. (Jan. 2017). "An Overview on Quality Evaluation of Self-Adaptive Systems". In: *Managing Trade-offs in Adaptable Software Architectures*. Ed. by Ivan Mistrik et al. Morgan Kaufmann, pp. 325–352. DOI: 10.1016/B978-0-12-802855-1.00013-7.

Rivera, Jose E, Francisco Durán, and Antonio Vallecillo (2009). "A graphical approach for modeling time-dependent behavior of DSLs". In: *IEEE Symp. on Visual Languages and Human-Centric Computing (VL/HCC)*. Ed. by Robert DeLine, Mark Minas, and Martin Erwig. IEEE, pp. 51–55.

Rivera, José Eduardo, Francisco Durán, and Antonio Vallecillo (2009). "Formal Specification and Analysis of Domain Specific Models Using Maude". In: *Simulation* 85.11-12, pp. 778–792.

Rodrigues da Silva, Alberto (2015). "Model-driven engineering: A survey supported by the unified conceptual model". In: *Computer Languages, Systems Structures* 43, pp. 139–155. ISSN: 1477-8424. DOI: https://doi.org/10.1016/j.cl.2015.06.001. URL: https://www.sciencedirect.com/science/article/pii/S1477842415000408.

Rozenberg, Grzegorz (1997). *Handbook of graph grammars and computing by graph transformation*. Vol. 1. World scientific.

Sanislav, Teodora, George Mois, and Liviu Miclea (Dec. 2015). "An Approach to Model Dependability of Cyber-Physical Systems". In: *Microprocessors and Microsystems* 41. DOI: 10.1016/j.micpro.2015.11.021.

Saputri, Theresia Ratih Dewi and Seok-Won Lee (2020). "The Application of Machine Learning in Self-Adaptive Systems: A Systematic Literature Review". In: *IEEE Access* 8, pp. 205948–205967. DOI: 10.1109/ACCESS.2020.3036037.

Serral, Estefanía, Paolo Sernani, and Fabiano Dalpiaz (Nov. 2017). "Personalized adaptation in pervasive systems via non-functional requirements". In: *Journal of Ambient Intelligence and Humanized Computing* 9. DOI: 10.1007/s12652-017-0611-4.

Sinreich, David (2005). *An architectural blueprint for autonomic computing*. Tech. rep. IBM.

Smith, Connie U. and Lloyd G. Williams (2002). *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Object-Technology Series. Addison-Wesley. ISBN: 0-201-72229-1.

Troya, Javier et al. (2013). "Model-driven performance analysis of rule-based domain specific visual models". In: *Information & Software Technology* 55.1, pp. 88–110.

Vale, Tassio et al. (Jan. 2016). "Twenty-eight years of component-based software engineering". In: *Journal of Systems and Software* 111, pp. 128–148. DOI: 10.1016/j.jss.2015.09.019. URL: https://doi.org/10.1016/j.jss.2015.09.019.

Virtual Queues 'Caixa Tem'. https://oglobo.globo.com/economia/caixa-tem-volta-funcionar-mas-usuarios-enfrentam-fila-virtual-24786935. Accessed: 2021-10-05.

Vogel, Thomas and Holger Giese (2018). "Model-Driven Engineering of Self-Adaptive Software with EUREMA". In: *CoRR* abs/1805.07353. arXiv: 1805.07353. URL: http://arxiv.org/abs/1805.07353.

Wanke, Egon (1990). "PLEXUS: Tools for Analyzing Graph Grammars". In: *Graph-Grammars and Their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5-9, 1990, Proceedings*. Ed. by Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg. Vol. 532. Lecture Notes in Computer Science. Springer, pp. 68–69. DOI: 10.1007/BFb0017381. URL: https://doi.org/10.1007/BFb0017381.

Weyns, Danny (2020). *An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective*. John Wiley & Sons.

Weyns, Danny and Usman Iftikhar (2016). "Model-based simulation at runtime for self-adaptive systems". In: *International Conference on Autonomic Computing (ICAC)*. Ed. by Samuel Kounev, Holger Giese, and Jie Liu. IEEE, pp. 1–9.

Wu, Yinjun, Edgar Dobriban, and Susan B. Davidson (2020). "DeltaGrad: Rapid retraining of machine learning models". In: DOI: 10.48550/ARXIV.2006.14755. URL: https://arxiv.org/abs/2006.14755.