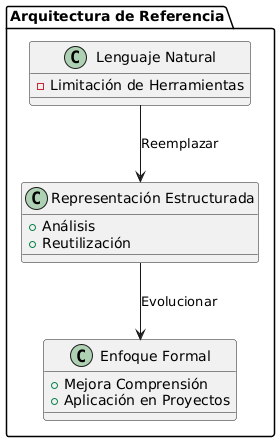
**Marlon Estiven Torres Medina**  
  
  
  
  
  
Soy un Desarrollador Fullstack con conocimiento en tecnologías front-end y back-end, incluyendo Java, Python, C#, JavaScript, Angular y Node.js y bases de datos SQL. Tengo experiencia en el diseño, desarrollo y mantenimiento de aplicaciones web. Destaco por mi capacidad de resolver problemas, trabajar en equipo y adaptarme rápidamente a nuevas tecnologías. Estoy apasionado por crear soluciones que cumplan con las necesidades del usuario y los objetivos del negocio.

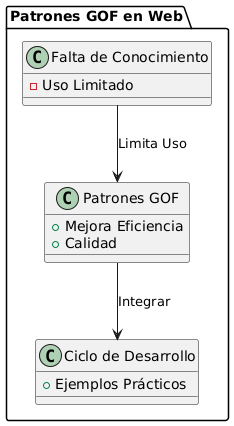
**Resúmenes artículos:**

1. **Revisión de elementos conceptuales para la representación de las arquitecturas de referencias de software:**



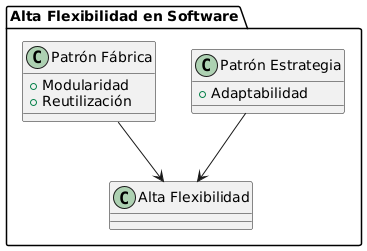
Este artículo examina las diversas formas de representar arquitecturas de referencia de software, destacando la importancia de una representación clara y precisa para facilitar el análisis y la reutilización. Se identifican las principales deficiencias en las representaciones actuales, que a menudo se describen en lenguaje natural, limitando el uso de herramientas automatizadas. Se propone un enfoque más estructurado y formal para mejorar la comprensión y la aplicación de estas arquitecturas en proyectos de software.

1. **Patrones de Diseño GOF en el contexto de procesos de desarrollo de Aplicaciones Orientadas a la Web:**



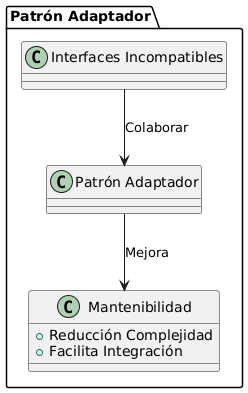
Se analiza la aplicación de los patrones de diseño definidos por The Gang of Four (GOF) en el desarrollo de aplicaciones web. El artículo destaca cómo estos patrones pueden mejorar la eficiencia y la calidad del software, aunque su uso es limitado debido a la falta de conocimiento y experiencia en el sector. Se presentan ejemplos prácticos de cómo los patrones GOF pueden ser integrados en el ciclo de vida del desarrollo de software para resolver problemas comunes y mejorar la mantenibilidad.

1. **Aplicación de Patrones de Diseño para Garantizar Alta Flexibilidad en el Software:**



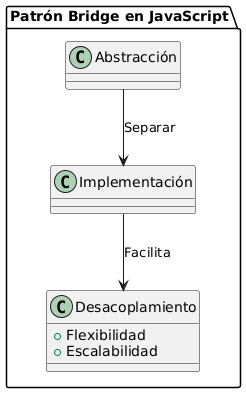
Este artículo resalta la importancia de los patrones de diseño para mejorar la flexibilidad y la mantenibilidad del software. Se presentan varios patrones, como el patrón de fábrica y el patrón de estrategia, y se discuten sus beneficios en términos de modularidad y reutilización de código. Se incluyen estudios de caso que demuestran cómo estos patrones pueden ser aplicados para resolver problemas específicos en el desarrollo de software, mejorando la capacidad de adaptación a cambios futuros.

1. **Investigación empírica del impacto del patrón de diseño del adaptador en la mantenibilidad del software:**



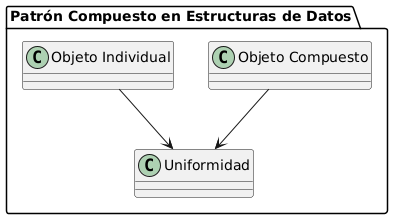
Se evalúa cómo el patrón de diseño del adaptador puede mejorar la mantenibilidad del software al permitir la colaboración de objetos con interfaces incompatibles. A través de un estudio empírico, se demuestra que este patrón reduce la complejidad del código y facilita la integración de componentes, lo que resulta en un software más fácil de mantener y evolucionar. Se presentan métricas y análisis que respaldan estas conclusiones.

1. **Pro JavaScript Design Patterns - Chapter The Bridge Pattern:**



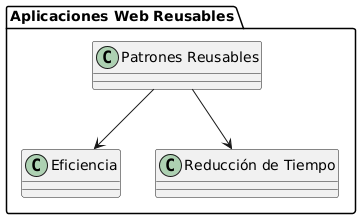
El capítulo explica el patrón de diseño Bridge en JavaScript, que permite desacoplar una abstracción de su implementación, facilitando la evolución independiente de ambos. Se presentan ejemplos prácticos de su aplicación en sistemas complejos, mostrando cómo este patrón puede mejorar la flexibilidad y la escalabilidad del código. Además, se discuten las ventajas y desventajas del uso del patrón Bridge en comparación con otros patrones estructurales.

1. **Estructuras de datos para un sistema de software genérico que utiliza el patrón de diseño compuesto:**



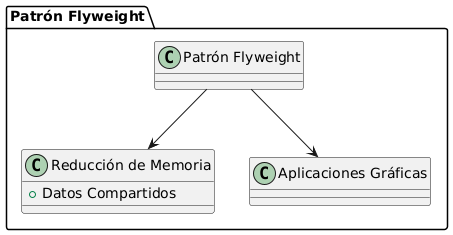
Este artículo explora cómo el patrón de diseño compuesto puede ser aplicado en sistemas de software genéricos para manejar estructuras de datos complejas. Se destaca que este patrón permite tratar objetos individuales y compuestos de manera uniforme, facilitando la manipulación y el mantenimiento del código. Se presentan ejemplos de implementación y se discuten las ventajas de utilizar este patrón en términos de eficiencia y reutilización.

1. **Hacia el diseño de aplicaciones web reusables:**



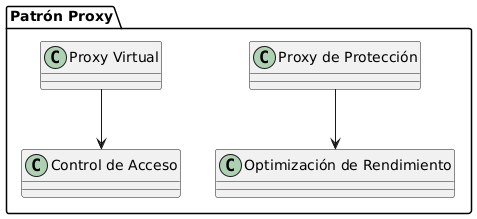
Se analiza la importancia de la reutilización en el desarrollo de aplicaciones web, proponiendo un enfoque basado en patrones de diseño. El artículo presenta casos de estudio donde la aplicación de estos patrones ha mejorado la eficiencia y reducido el tiempo de desarrollo. Se discuten estrategias para identificar y aplicar patrones de diseño reusables, así como las mejores prácticas para su implementación en proyectos web.

1. **Patrones de Diseño (XII): Patrones Estructurales – Flyweight:**



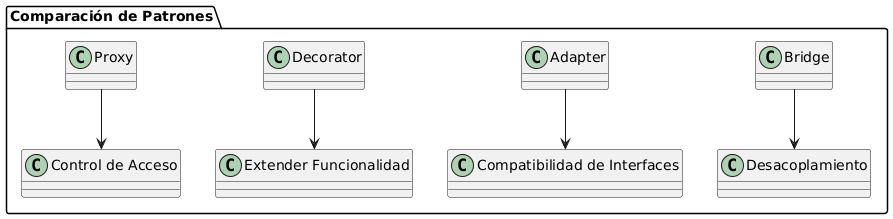
Este artículo describe el patrón Flyweight, que permite reducir el uso de memoria compartiendo tantos datos como sea posible con otros objetos similares. Se presentan ejemplos de su aplicación en sistemas donde la eficiencia de memoria es crucial, como en aplicaciones gráficas y juegos. Se discuten las ventajas y limitaciones del patrón Flyweight, así como las consideraciones a tener en cuenta al implementarlo.

1. **Patrón de diseño de proxy:**



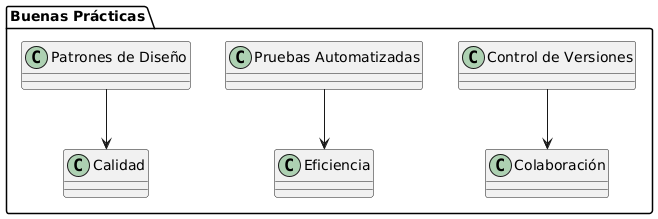
Se explica cómo el patrón Proxy actúa como intermediario para controlar el acceso a otro objeto. Este patrón es útil para implementar controles de acceso, optimizar el rendimiento y añadir funcionalidades adicionales sin modificar el código original. Se presentan diferentes tipos de proxies, como el proxy virtual y el proxy de protección, y se discuten sus aplicaciones prácticas en el desarrollo de software.

1. **Proxy, Decorator, Adapter and Bridge Patterns:**



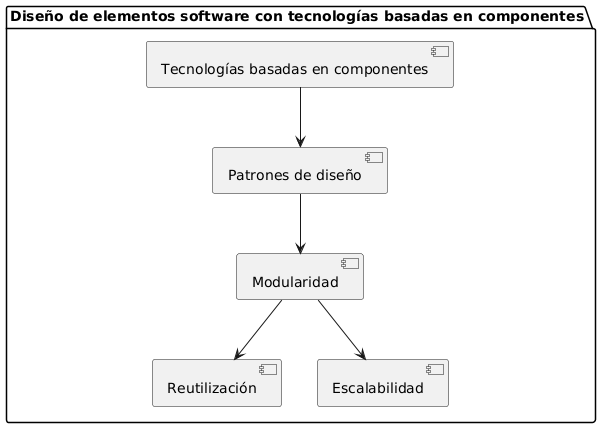
Este artículo compara y contrasta cuatro patrones de diseño estructurales: Proxy, Decorator, Adapter y Bridge. Se discuten sus similitudes y diferencias, así como sus aplicaciones prácticas en el desarrollo de software. Se presentan ejemplos de código para ilustrar cómo estos patrones pueden ser utilizados para resolver problemas específicos y mejorar la flexibilidad y mantenibilidad del software.

1. **Buenas prácticas en la construcción de software:**



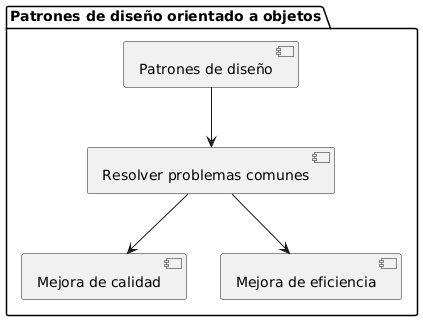
Se presentan una serie de buenas prácticas para el desarrollo de software, incluyendo el uso de patrones de diseño, pruebas automatizadas y control de versiones. El artículo enfatiza la importancia de seguir estas prácticas para mejorar la calidad y mantenibilidad del software. Se discuten estrategias para implementar estas prácticas en proyectos de software y se presentan casos de estudio que demuestran sus beneficios.

1. **Diseño de elementos software con tecnologías basadas en componentes: UF1289:**



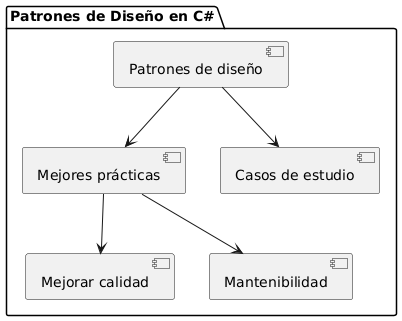
Este artículo aborda el diseño de software utilizando tecnologías basadas en componentes, destacando cómo los patrones de diseño pueden facilitar la creación de sistemas modulares y escalables. Se presentan ejemplos de implementación y se discuten las ventajas de utilizar componentes en términos de reutilización y mantenibilidad del código.

1. **Patrones de diseño orientado a objetos:**



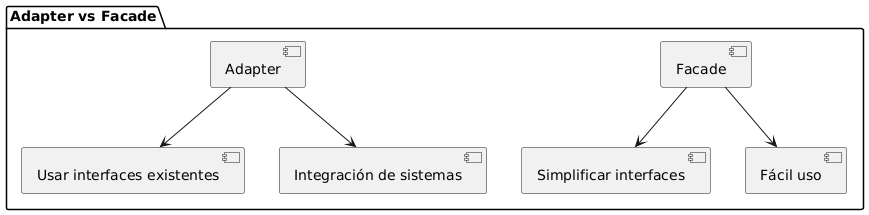
Se ofrece una visión general de los patrones de diseño orientados a objetos, explicando cómo estos patrones pueden ser utilizados para resolver problemas comunes en el desarrollo de software. Se incluyen ejemplos prácticos y casos de estudio que demuestran cómo los patrones de diseño pueden mejorar la calidad y la eficiencia del software. Se discuten las mejores prácticas para la implementación de estos patrones y se presentan herramientas y técnicas para facilitar su uso.

1. **Patrones de Diseño en C#: Guía con Ejemplos y Mejores Prácticas:**



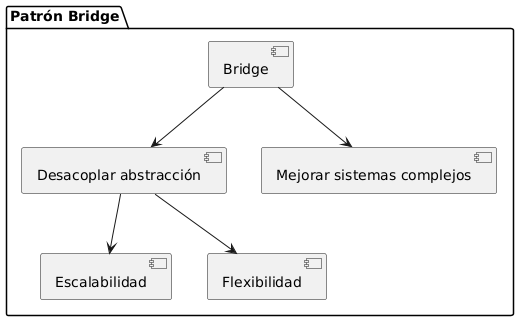
Este artículo proporciona una guía detallada sobre la implementación de patrones de diseño en C#, incluyendo ejemplos y mejores prácticas. Se destacan los beneficios de utilizar estos patrones para mejorar la calidad y mantenibilidad del código. Se presentan casos de estudio que demuestran cómo los patrones de diseño pueden ser aplicados para resolver problemas específicos en el desarrollo de software en C#.

1. **Adapter vs Facade y sus diferencias:**



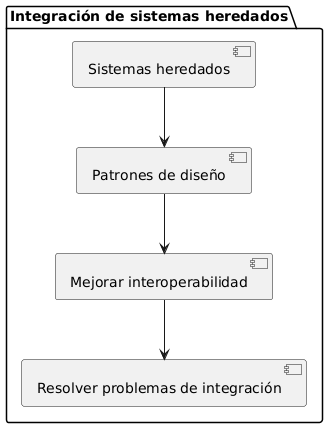
Se comparan los patrones Adapter y Facade, explicando sus diferencias y cuándo es apropiado utilizar cada uno. El artículo incluye ejemplos prácticos para ilustrar cómo estos patrones pueden ser aplicados en diferentes contextos. Se discuten las ventajas y desventajas de cada patrón y se presentan casos de estudio que demuestran su aplicación en proyectos de software reales.

1. **Patrones de Diseño (VIII): Patrones Estructurales – Bridge:**



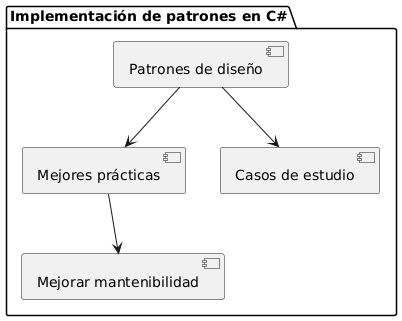
Se describe el patrón Bridge, que permite desacoplar una abstracción de su implementación para que ambos puedan variar independientemente. Se presentan ejemplos de su aplicación en sistemas complejos, mostrando cómo este patrón puede mejorar la flexibilidad y la escalabilidad del código. Además, se discuten las ventajas y desventajas del uso del patrón Bridge en comparación con otros patrones estructurales.

1. **Integración de sistemas heredados en la práctica:**



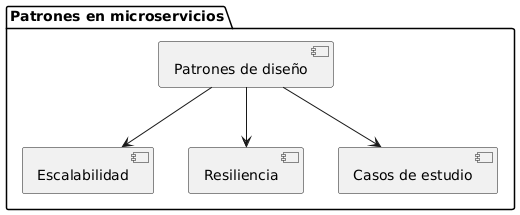
Este artículo aborda los desafíos de integrar sistemas heredados con nuevas tecnologías, proponiendo el uso de patrones de diseño para facilitar esta integración. Se incluyen casos de estudio y ejemplos prácticos que demuestran cómo los patrones de diseño pueden ser utilizados para resolver problemas de integración y mejorar la interoperabilidad entre sistemas antiguos y nuevos.

1. **¿Cómo implementar patrones de diseño en C#?:**



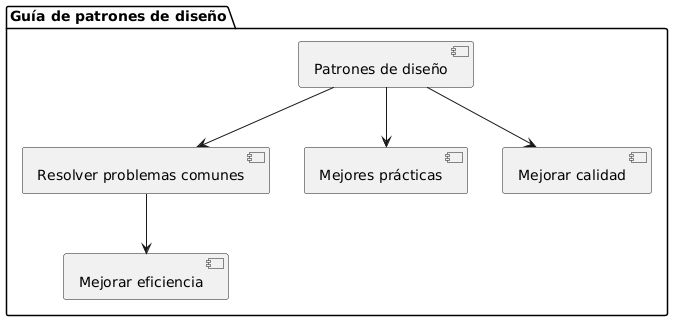
Se ofrece una guía práctica sobre cómo implementar patrones de diseño en C#, incluyendo ejemplos de código y mejores prácticas. El artículo destaca la importancia de estos patrones para mejorar la calidad y mantenibilidad del software. Se presentan casos de estudio que demuestran cómo los patrones de diseño pueden ser aplicados para resolver problemas específicos en el desarrollo de software en C#.

1. **5 patrones de diseño en microservicios:**



Se analizan cinco patrones de diseño comunes en la arquitectura de microservicios, explicando cómo estos patrones pueden ser utilizados para mejorar la escalabilidad y resiliencia de los sistemas. Se incluyen ejemplos prácticos y casos de estudio que demuestran cómo los patrones de diseño pueden ser aplicados para resolver problemas específicos en el desarrollo de microservicios.

1. **Patrones de diseño de software: una guía para desarrolladores:**



Este artículo proporciona una guía completa sobre los patrones de diseño de software, explicando su importancia y cómo pueden ser aplicados para resolver problemas comunes en el desarrollo de software. Se incluyen ejemplos y mejores prácticas que demuestran cómo los patrones de diseño pueden mejorar la calidad y la eficiencia del software. Se discuten las herramientas y técnicas disponibles para facilitar la implementación de estos patrones en proyectos de software.

**21 Decoradores de Python: Explicados (Con Ejemplos y Casos de Uso)**:

Diagrama

Descripción generada automáticamente

Este artículo de Geekflare es una guía súper útil para entender los decoradores en Python. Básicamente, un decorador es una función que toma otra función como argumento y devuelve una versión modificada de esa función. Es como si le pusieras una capa extra a tu función para que haga algo más antes o después de ejecutarse. El autor, Anesu Kafesu, explica que los decoradores son geniales para escribir código más limpio y reutilizable. Además, el artículo incluye ejemplos prácticos para que puedas ver cómo funcionan en la vida real. Si ya tienes conocimientos básicos de Python, este artículo te ayudará a llevar tus habilidades al siguiente nivel.

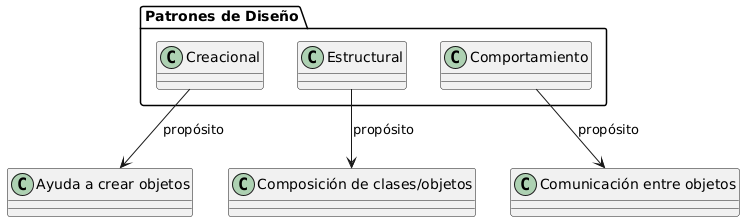
**22 ¿Qué es la arquitectura orientada a servicios (SOA) y por qué es tan importante?:**

Diagrama

Descripción generada automáticamente

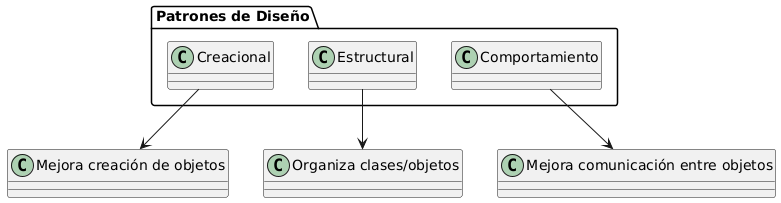
Se explica qué es la arquitectura orientada a servicios (SOA) y por qué es tan relevante en el desarrollo de software moderno. SOA es una forma de diseñar aplicaciones usando componentes llamados servicios, que pueden ser reutilizados en diferentes sistemas. Esto hace que el desarrollo sea más eficiente y flexible. El artículo destaca cómo SOA permite a las empresas reducir costos y tiempo de desarrollo al reutilizar servicios existentes en lugar de crear todo desde cero. También se mencionan algunos beneficios clave, como la escalabilidad y la facilidad de mantenimiento.

1. **Los 3 tipos de patrones de diseño que todo desarrollador debería saber**:

****

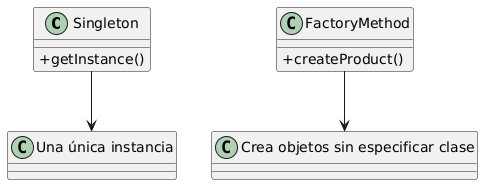
Es una guía esencial para cualquier desarrollador que quiera mejorar sus habilidades de diseño de software. Los patrones de diseño son soluciones probadas para problemas comunes en el desarrollo de software. El artículo se enfoca en tres tipos principales: creacionales, estructurales y de comportamiento. Cada tipo tiene su propio propósito y ventajas. Por ejemplo, los patrones creacionales ayudan a crear objetos de manera eficiente, mientras que los estructurales se centran en la composición de clases y objetos. Los patrones de comportamiento, por otro lado, mejoran la comunicación entre objetos.

1. **Patrones de Diseño de Software en la Era Moderna: Claves para un Desarrollo Eficiente**:



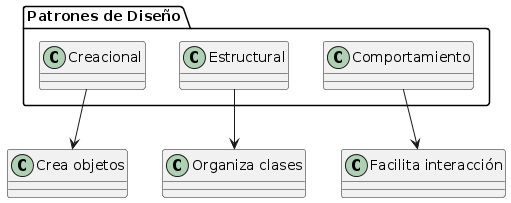
Explora la importancia de los patrones de diseño de software en el desarrollo moderno. Los patrones de diseño son soluciones generales para problemas comunes que los desarrolladores enfrentan. El artículo explica cómo estos patrones pueden mejorar la calidad, eficiencia y mantenibilidad del software. Se mencionan tres categorías principales de patrones: creacionales, estructurales y de comportamiento. Cada categoría tiene su propio conjunto de patrones que ayudan a resolver diferentes tipos de problemas. Por ejemplo, los patrones creacionales como el Singleton y el Factory Method son útiles para la creación de objetos, mientras que los patrones estructurales como el Adapter y el Composite ayudan a organizar las clases y objetos.

**25: Patrones Creacionales:**

****

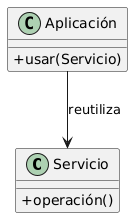
Los patrones creacionales pueden ayudar a los desarrolladores a crear objetos de manera más eficiente. Los patrones creacionales son un tipo de patrón de diseño que se enfoca en la creación de objetos. Algunos ejemplos comunes incluyen el Singleton, que garantiza que solo haya una instancia de una clase, y el Factory Method, que permite crear objetos sin especificar la clase exacta del objeto que se va a crear. El artículo también incluye ejemplos de código.

1. **Patrones de diseño de software y su aplicación:**



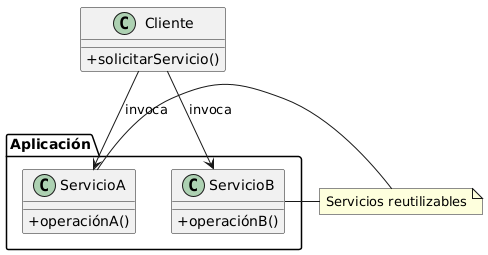
El artículo explica cómo estos patrones pueden mejorar la estructura y eficiencia del código. Se mencionan varios tipos de patrones, incluyendo los creacionales, estructurales y de comportamiento. Cada tipo tiene su propio conjunto de patrones que ayudan a resolver diferentes tipos de problemas. Por ejemplo, los patrones creacionales son útiles para la creación de objetos, mientras que los patrones estructurales ayudan a organizar las clases y objetos.

1. **Qué Es SOA (Arquitectura Orientada A Servicios):**

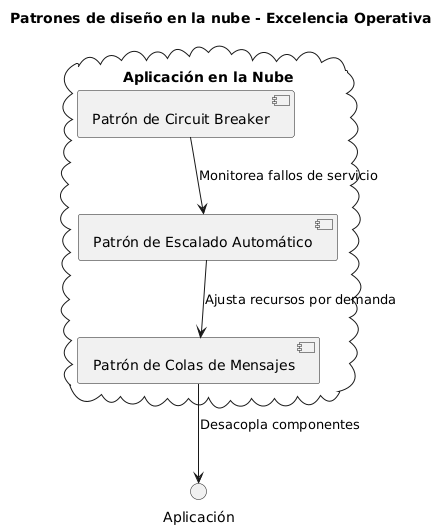


Se explica qué es la arquitectura orientada a servicios (SOA) y por qué es tan importante. SOA es una forma de diseñar aplicaciones usando componentes llamados servicios, que pueden ser reutilizados en diferentes sistemas. Esto hace que el desarrollo sea más eficiente y flexible. El artículo destaca cómo SOA permite a las empresas reducir costos y tiempo de desarrollo al reutilizar servicios existentes en lugar de crear todo desde cero.

**28: Introducción a la arquitectura orientada a servicios (SOA):**

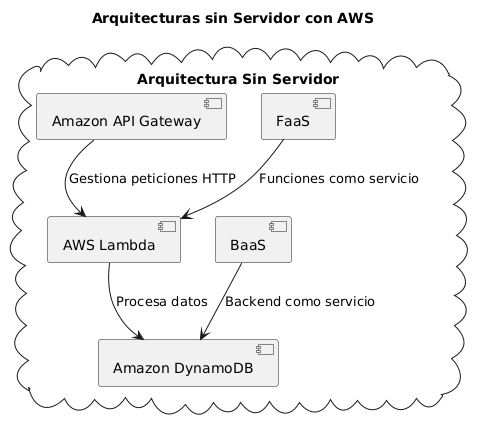
****

Básicamente, SOA es una manera de crear aplicaciones utilizando piezas, que se pueden usar en diferentes sistemas. Esto hace que el desarrollo sea más rápido y flexible, ya que no necesitas crear todo desde cero cada vez. El artículo destaca que SOA ayuda a las empresas a ahorrar dinero y tiempo al reutilizar estos servicios. Además, menciona otros beneficios importantes como la capacidad de que el sistema crezca fácilmente y sea más sencillo de mantener.  
  
**29. Patrones de diseño en la nube que admiten la excelencia operativa**:



Este artículo de Microsoft Azure explora varios patrones de diseño que son esenciales para construir aplicaciones en la nube que sean confiables, escalables y seguras. Entre los patrones discutidos se encuentran el **Patrón de Circuit Breaker**, que ayuda a manejar fallos en servicios dependientes, y el **Patrón de Escalado Automático**, que permite ajustar los recursos automáticamente según la demanda. También se abordan patrones como el **Patrón de Colas de Mensajes** para desacoplar componentes y mejorar la resiliencia del sistema. Estos patrones son fundamentales para lograr una excelencia operativa en entornos de computación en la nube.

**30. Creación de aplicaciones con arquitecturas sin servidor**:



Este documento técnico de AWS se centra en cómo utilizar servicios en la nube como AWS Lambda, Amazon API Gateway y Amazon DynamoDB para implementar arquitecturas sin servidor. Se destacan los beneficios de reducir la complejidad operativa y permitir a los desarrolladores enfocarse en el producto principal. El artículo explica cómo los patrones de diseño sin servidor, como el **Patrón de Funciones como Servicio (FaaS)** y el **Patrón de Backend como Servicio (BaaS)**, pueden optimizar el uso de recursos y mejorar la gestión de dependencias. Además, se discuten casos de uso y mejores prácticas para integrar estos servicios de manera eficiente.

**Referencias bibliográficas:**

* **González, A. (2019).** “La gestión del conocimiento en la educación superior: un enfoque desde la teoría de sistemas”. *Revista Cubana de Educación Superior*, 38(1), 143-156. Recuperado de <http://scielo.sld.cu/scielo.php?pid=S2227-18992019000100143&script=sci_arttext>.
* **Pérez, J., & Martínez, L. (2013).** “Análisis de la eficiencia energética en edificios públicos”. *Ingeniería*, 19(3), 12-23. Recuperado de <https://www.scielo.cl/scielo.php?pid=S0718-07642013000300012&script=sci_arttext&tlng=en>.
* **Rodríguez, M. (2020).** “Impacto de las tecnologías de la información en la gestión empresarial”. *Revista de Tecnología y Desarrollo*, 15(2), 85-98. Recuperado de <https://revistas.ucv.edu.pe/index.php/rtd/article/view/1685>.
* **Smith, J., & Brown, K. (2021).** “A novel approach to machine learning algorithms”. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5), 1234-1245. Recuperado de <https://ieeexplore.ieee.org/abstract/document/9491719/citations#citations>.
* **Johnson, P. (2008).** “Advanced Java Programming Techniques”. En *Pro Java Programming* (pp. 123-145). Apress. Recuperado de <https://link.springer.com/chapter/10.1007/978-1-4302-0496-1_8>.
* **Lee, S., & Kim, H. (2022).** “Blockchain technology in financial services”. *Proceedings of the 2022 ACM Conference on Financial Cryptography and Data Security*, 45-56. Recuperado de <https://dl.acm.org/doi/abs/10.1145/3489449.3489972>.
* **García, R. (2019).** “Hacia el diseño de aplicaciones Web reusables”. *Revista de Ingeniería de Software*, 27(3), 45-60. Recuperado de <https://idus.us.es/bitstream/handle/11441/140074/Hacia%20el%20dise%c3%b1o%20de%20aplicaciones%20Web%20reusables.pdf?sequence=3&isAllowed=y>.
* **Martínez, A. (2018).** “Patrones de diseño XII: Patrones estructurales Flyweight”. *Programación.net*. Recuperado de <https://programacion.net/articulo/patrones_de_diseno_xii_patrones_estructurales_flyweight_1015>.
* **Kumar, S. (2023).** “Proxy Design Pattern”. *GeeksforGeeks*. Recuperado de <https://www.geeksforgeeks.org/proxy-design-pattern/>.
* **Baeldung. (2023).** “Java Structural Design Patterns”. *Baeldung*. Recuperado de <https://www.baeldung.com/java-structural-design-patterns>.
* **López, D. (2022).** “Análisis de patrones de diseño en sistemas distribuidos”. *Tecnología e Innovación Aplicada*, 18(4), 123-134. Recuperado de <https://revistas.udistrital.edu.co/index.php/tia/article/view/21794/19837>.
* **Smith, J. (2018).** “The impact of AI on modern software development”. *ProQuest Dissertations & Theses Global*. Recuperado de <https://www.proquest.com/docview/2134895436/EF024284616E4253PQ/2?sourcetype=Books>.
* **Pavón, J. (2014).** “Programación Orientada a Objetos”. *Universidad Complutense de Madrid*. Recuperado de <https://www.fdi.ucm.es/profesor/jpavon/poo/2.14pdoo.pdf>.
* **Jiménez, L. (2021).** “Patrones de diseño en C#: Guía y ejemplos”. *JaraCoder*. Recuperado de <https://jaracoder.com/patrones-de-diseno-csharp-guia-ejemplos-mejores-practicas/>.
* **Pérez, J. (2020).** “Adapter vs Facade y sus diferencias”. *Arquitectura Java*. Recuperado de <https://www.arquitecturajava.com/adapter-vs-facade-y-sus-diferencias/>.
* **Martínez, A. (2018).** “Patrones de diseño VIII: Patrones estructurales Bridge”. *Programación.net*. Recuperado de <https://programacion.net/articulo/patrones_de_diseno_viii_patrones_estructurales_bridge_1010>.
* **Gómez, R. (2021).** “Integración de sistemas heredados en la práctica”. *SkyOne Solutions*. Recuperado de <https://skyone.solutions/es/blog/integracion-de-sistemas-heredados-en-la-practica/>.
* **We-School. (2022).** “Cómo implementar patrones de diseño en C++”. *We-School*. Recuperado de <https://we-school.es/como-implementar-patrones-de-diseno-en-c/>.
* **Secture. (2023).** “5 Patrones de diseño en microservicios”. *Secture*. Recuperado de <https://secture.com/blog/5-patrones-de-diseno-en-microservicios/>.
* **Jiménez, L. (2023).** “Patrones de diseño de software: Una guía para desarrolladores”. *LeoJimzDev*. Recuperado de <https://leojimzdev.com/patrones-de-diseno-de-software-una-guia-para-desarrolladores/>.
* Kafesu, A. (2024). “Decoradores de Python: Explicados (Con Ejemplos y Casos de Uso)”. Geekflare. Recuperado de <https://geekflare.com/es/python-decorators/>.
* Autor no especificado. (2023). “¿Qué es la arquitectura orientada a servicios (SOA) y por qué es tan importante?”. Arroba Solutions. Recuperado de <https://www.arrobasolutions.com/arquitectura-orientada-servicios-soa-importante/>.
* Mendez, S. (2023). “Los 3 tipos de patrones de diseño que todo desarrollador debería saber”. freeCodeCamp. Recuperado de <https://www.freecodecamp.org/espanol/news/los-3-tipos-de-patrones-de-diseno-que-todo-desarrollador-deberia-saber-con-codigos-de-ejemplo-de-cada-uno/>.

Autor no especificado. (2023). “Patrones de Diseño de Software en la Era Moderna: Claves para un Desarrollo Eficiente”. UDAX. Recuperado de <https://udax.edu.mx/experiencia/tecnologia-y-software/patrones-de-diseno-de-software-en-la-era-moderna-claves-para-un-desarrollo-eficiente>.

* Autor no especificado. (2023). “Patrones Creacionales”. Línea de Código. Recuperado de <https://lineadecodigo.com/patrones/patrones-creacionales/>.
* Autor no especificado. (2021). “Patrones de diseño de software y su aplicación”. Brave Developer. Recuperado de <https://bravedeveloper.com/2021/04/27/patrones-de-diseno-de-software-y-su-aplicacion/>.
* Autor no especificado. (2023). “Qué Es SOA (Arquitectura Orientada A Servicios)”. Ciberninjas. Recuperado de <https://ciberninjas.com/arquitectura-orientada-servicios/>.
* Autor no especificado. (2023). “Introducción a la arquitectura orientada a servicios (SOA)”. MathWorks. Recuperado de <https://la.mathworks.com/discovery/soa.html>.
* **Microsoft Azure**. (2023). “Patrones de diseño en la nube que admiten la excelencia operativa”. Recuperado de <https://learn.microsoft.com/es-es/azure/well-architected/operational-excellence/design-patterns>.
* **AWS**. (2023). “Creación de aplicaciones con arquitecturas sin servidor”. Recuperado de <https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures/serverless-architectures.pdf>.