Análisis y Diseño de Sistemas 2

Proceso para arquitectura de software

Material de apoyo del curso Análisis y Diseño de Sistemas 2 de la USAC

Ing. Ricardo Morales

Primer semestre 2016

Análisis y Diseño de Sistemas 2

Proceso para arquitectura de software

Tabla de contenido

PROCESO PARA DEFINIR ARQUITECTURA DE SOFTWARE

Objetivos

Comprender las actividades de un proceso de definición de arquitectura

Aplicar el proceso de arquitectura en el proyecto del curso

Definición de proceso de definición de arquitectura

El término "definición de arquitectura" se refiere a:

- Un proceso por el cual las necesidades y temas de interés de los stakeholders son capturados,
- Se diseña una arquitectura para alcanzar esas necesidades y
- La arquitectura está descrita de forma clara y no ambigua a través de una descripción de arquitectura

El proceso incorpora elementos de diseño y de análisis de requerimientos

Diseño es una actividad enfocada en el espacio de la solución, dirigida primariamente a un grupo de gente: desarrolladores

El análisis de requerimientos, por otro lado, es una actividad enfocada en el espacio del problema, que ignora las necesidades y restricciones de grupos como desarrolladores y administradores de sistemas, porque define que se desea en vez de que es posible

Rol del arquitecto

Principio:

El arquitecto es responsable de diseñar, documentar y liderar la construcción de un sistema que satisfaga las necesidades de todos los stakeholders

Se ven 4 aspectos en este rol:

- Identificar e incluir stakeholders
- Entender y capturar los temas de interés de los stakeholders
- Crear y ser dueño de la definición de una arquitectura que solucione los temas de interés
- Tomar un rol de liderazgo en la realización de la arquitectura en un producto físico o sistema

Proceso de definición de arquitectura

Principios

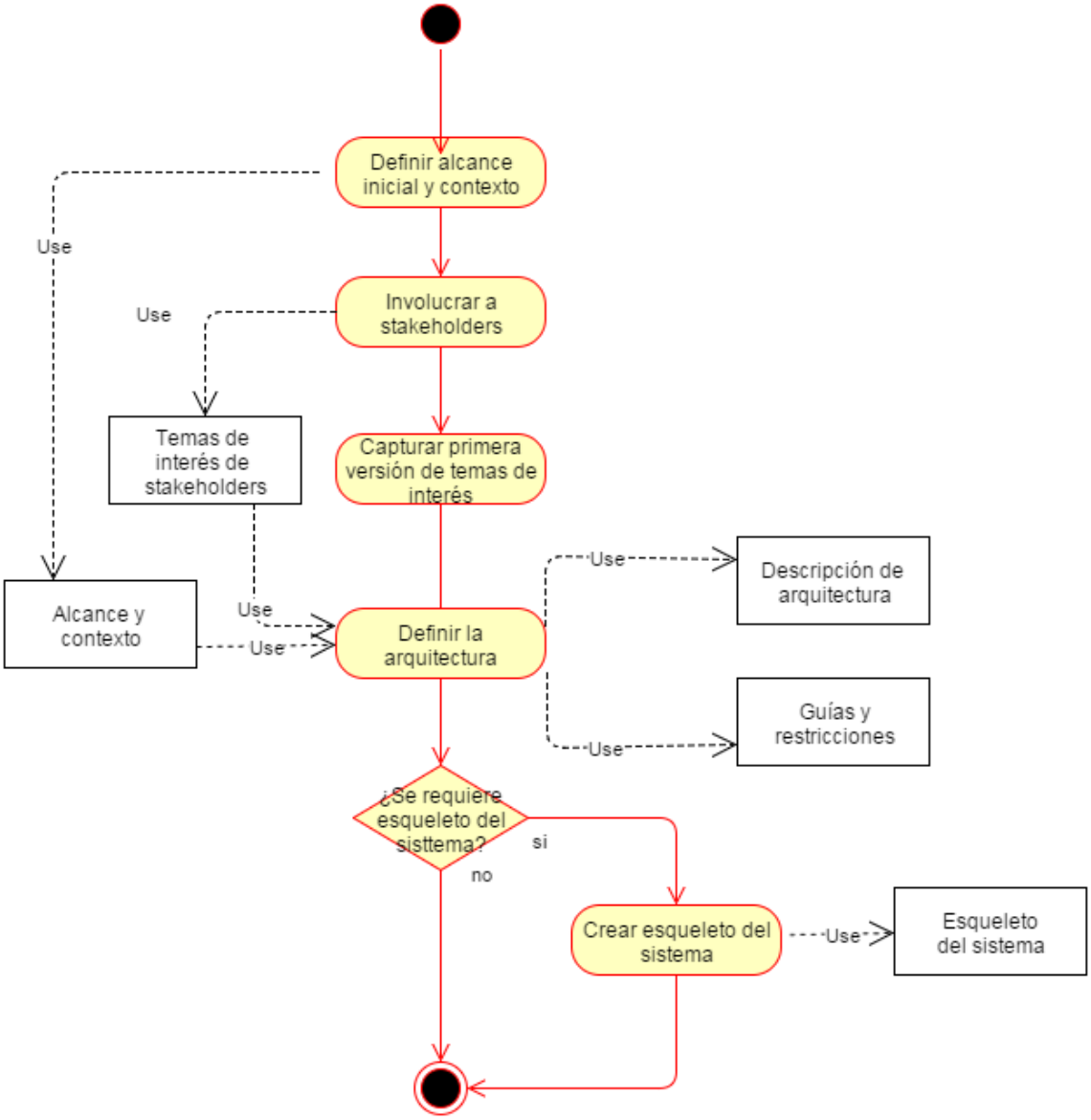
Para que un proceso de definición de arquitectura sea exitoso, se debe adherir a los siguientes principios:

- Debe ser dirigido por los temas de interés de los stakeholders
- Debe promoverse una comunicación efectiva de decisiones de arquitectura, principios y la solución misma a los stakeholders
- Debe asegurarse que las decisiones y principios de arquitectura son aplicadas y usadas durante todo el ciclo de vida hasta el despliegue
- Debe ser, en la medida de lo posible, estructurado
- Debe ser pragmático, es decir considerar temas de la realidad, como falta de dinero, falta de habilidades técnicas, etc.
- Debe ser agnóstico a la tecnología
- Debe integrarse con el ciclo de vida de desarrollo elegido
- Debe alinearse con las buenas prácticas de ingeniería de software y estándares de administración de calidad

Salidas del proceso

- Desarrollar la arquitectura
- Clarificación de requerimientos y otras entradas del proceso
- Administración de las expectativas de los stakeholders
- Identificación y evaluación de opciones de arquitectura
- Descripción de criterios de aceptación de la arquitectura (indirectamente)
- Creación de un conjunto de entradas de diseño (idealmente)

Actividades de soporte de definición de arquitectura



Actividad: Definir alcance inicial y contexto

Objetivo

- Definir claramente los límites de comportamiento y responsabilidades del sistema, y el contexto operacional y organizacional dentro del cual existe el sistema

Salidas

- Definiciones iniciales de las metas del sistema
- Que está incluido y excluido de sus responsabilidades
- Definición inicial del contexto del sistema

Entradas

- Necesidades y visión del adquiriente
- Estrategia organizacional
- Arquitectura empresarial de TI

Actividad: Involucrar a los stakeholders

Objetivo

- Identificar los stakeholders importantes del sistema y crear una relación de trabajo con ellos

Salidas

- Definición de cada uno de los grupos de stakeholders, con uno o mas nombres de involucrados que representan al grupo

Entradas

- Alcance y contexto a partir de le vista inicial de contexto
- Estructura organizacional

Actividad: Capturar primera versión de temas de interés

Objetivo

- Entender claramente los temas de interés que cada grupo de stakeholders tiene acerca del sistema y las prioridades de tema de interés

Salidas

- Definición inicial de un conjunto de temas de interés priorizados para cada grupo de stakeholders

Entradas

- Lista de stakeholders
- Alcance y contexto

Actividad: Definir la arquitectura

Objetivos

- Crear la descripción de arquitectura para el sistema

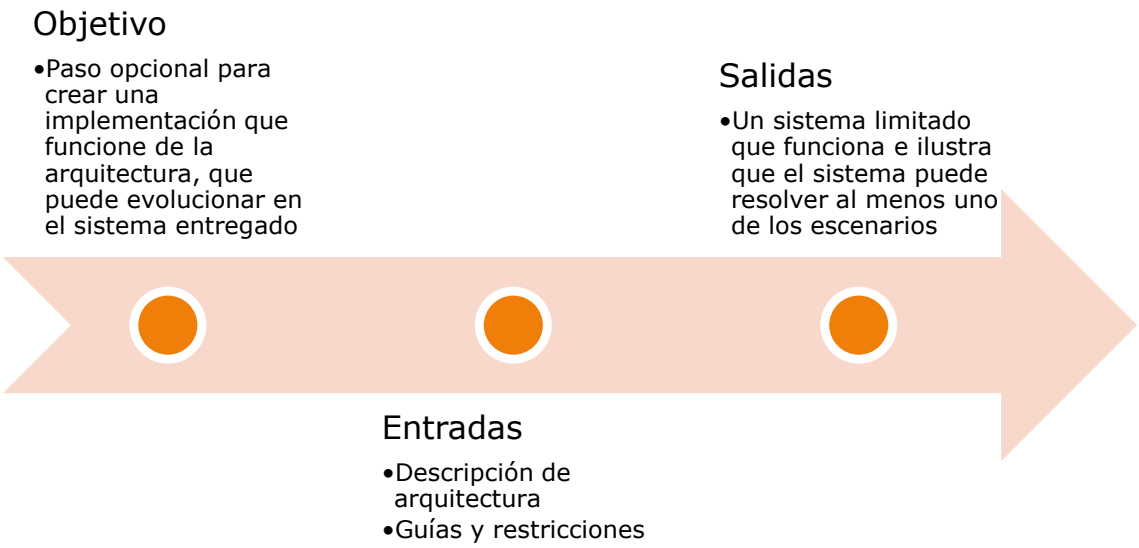
Salidas

- Descripción de arquitectura
- Guías y restricciones

Entradas

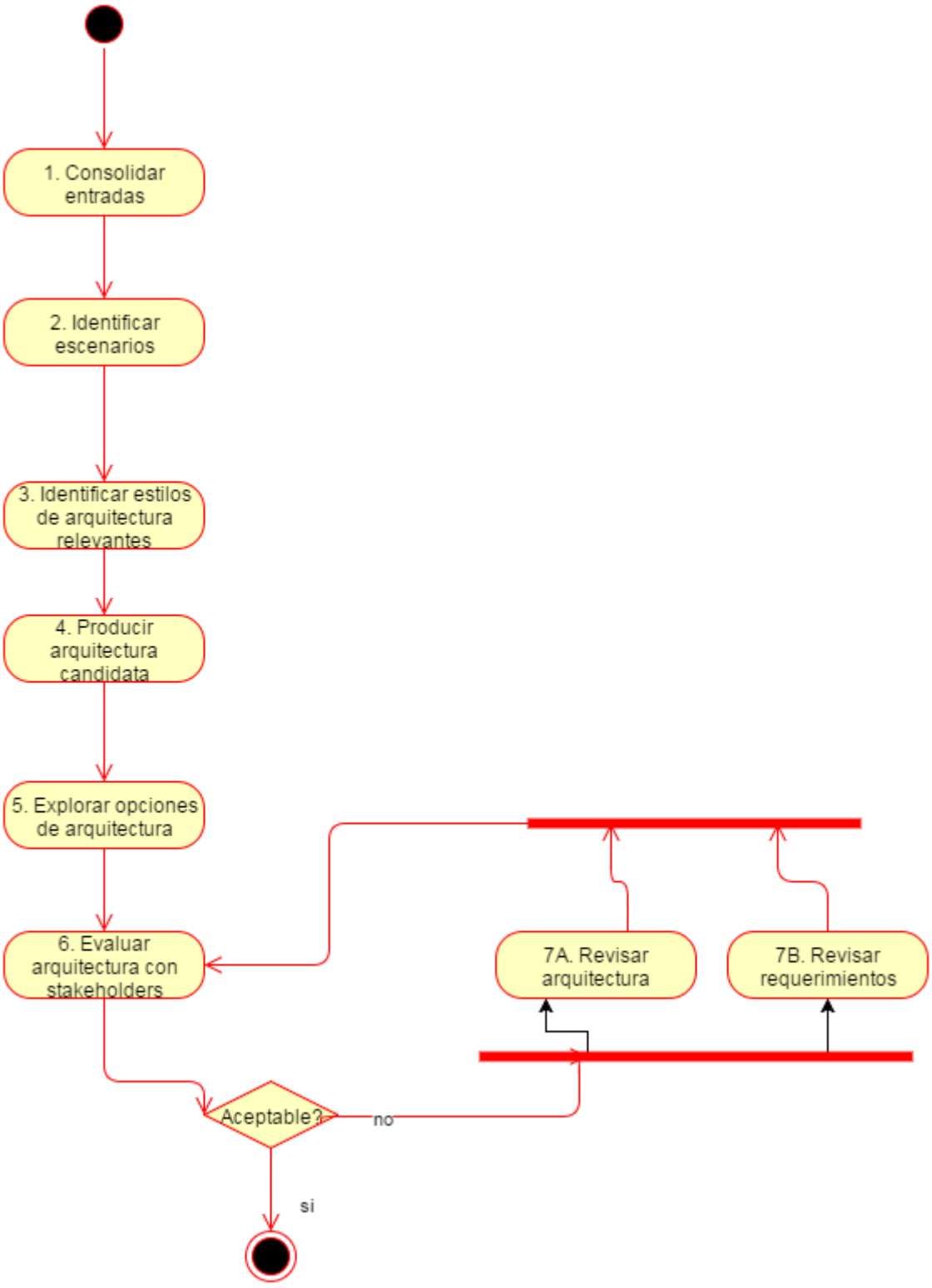
- Lista de stakeholders
- Alcance y contexto

Actividad: Crear el esqueleto del sistema

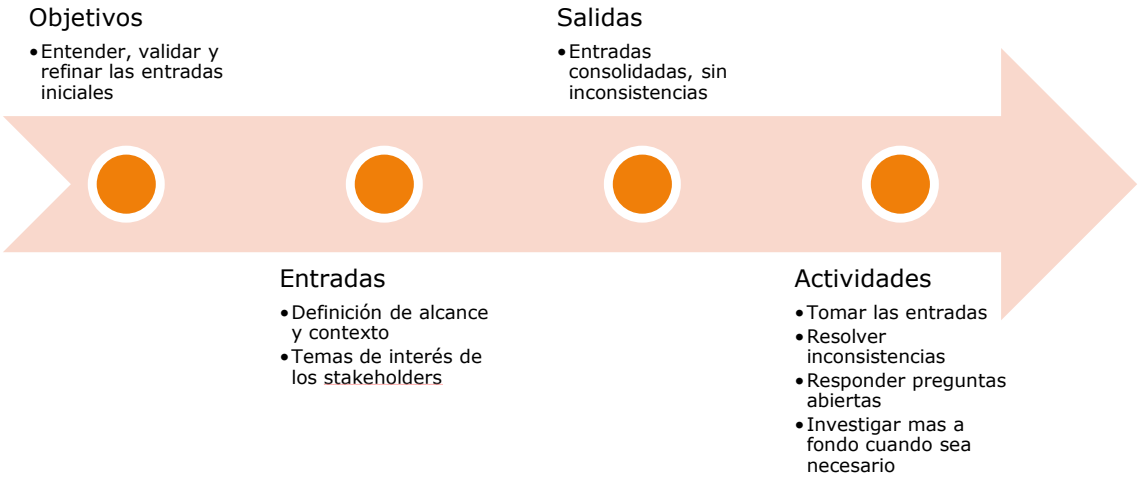


Detalle de actividad definir la arquitectura

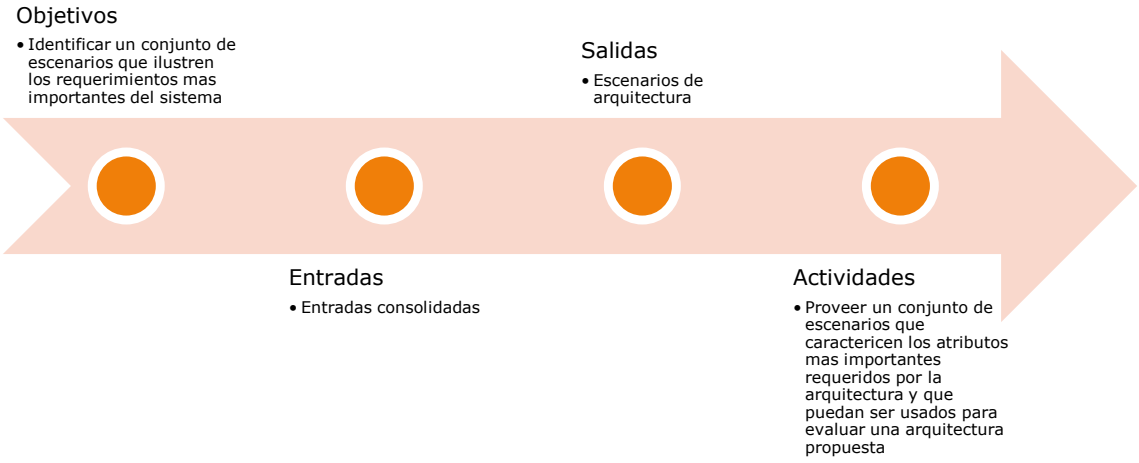
En esta sección se presentan con detalle los pasos de la actividad para definir la arquitectura de un sistema.


1. Consolidar entradas
2. Identificar escenarios
3. Identificar estilos de arquitectura relevantes
4. Producir arquitectura candidata
5. Explorar opciones de arquitectura
6. Evaluar arquitectura con stakeholders
7A. Revisar arquitectura
7B. Revisar requerimientos
Aceptable?
no
si

1. Consolidar las entradas



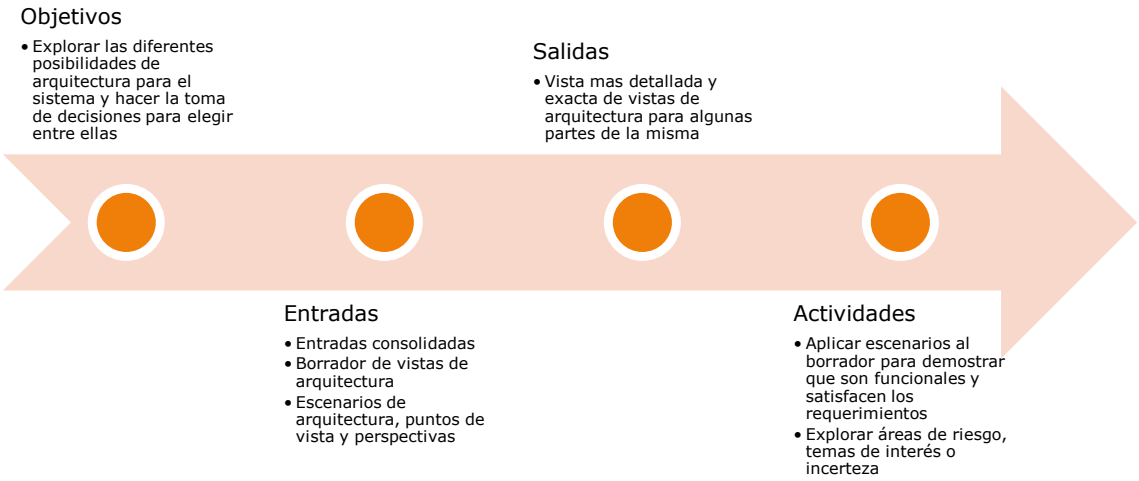
2. Identificar escenarios



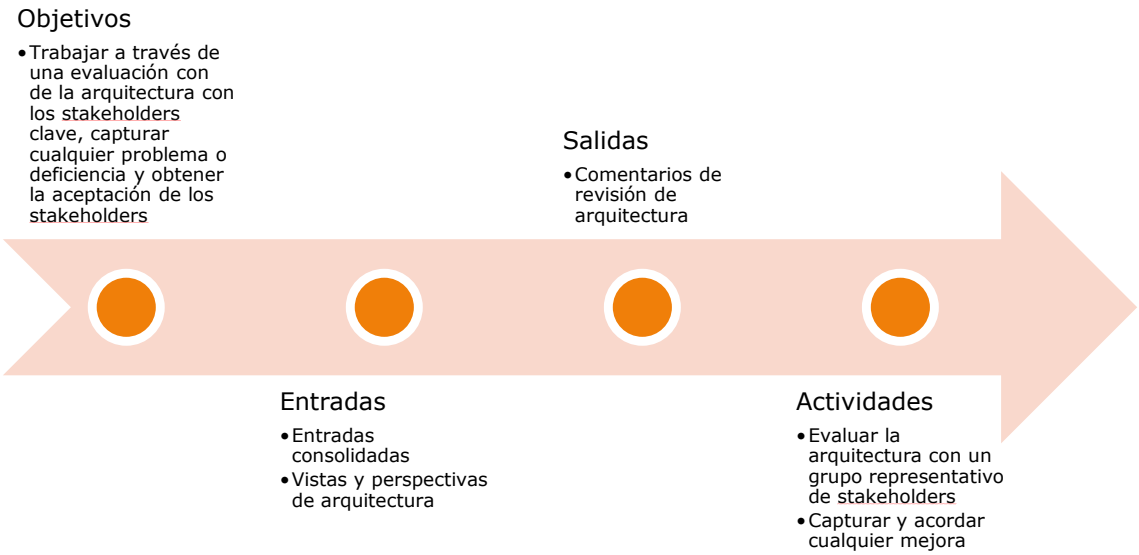
3. Identificar los estilos de arquitectura relevantes

4. Producir una arquitectura candidata

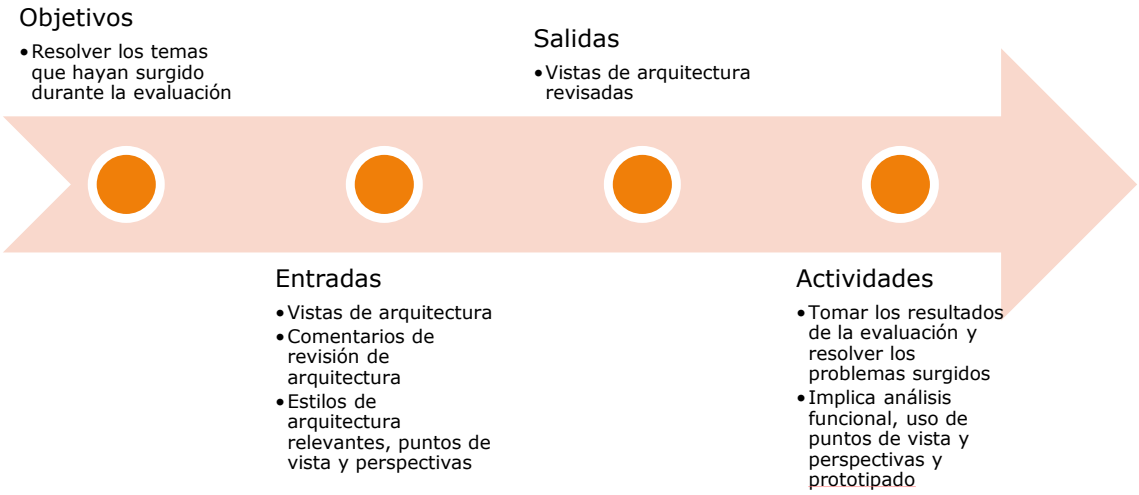
5. Explorar las opciones de arquitectura



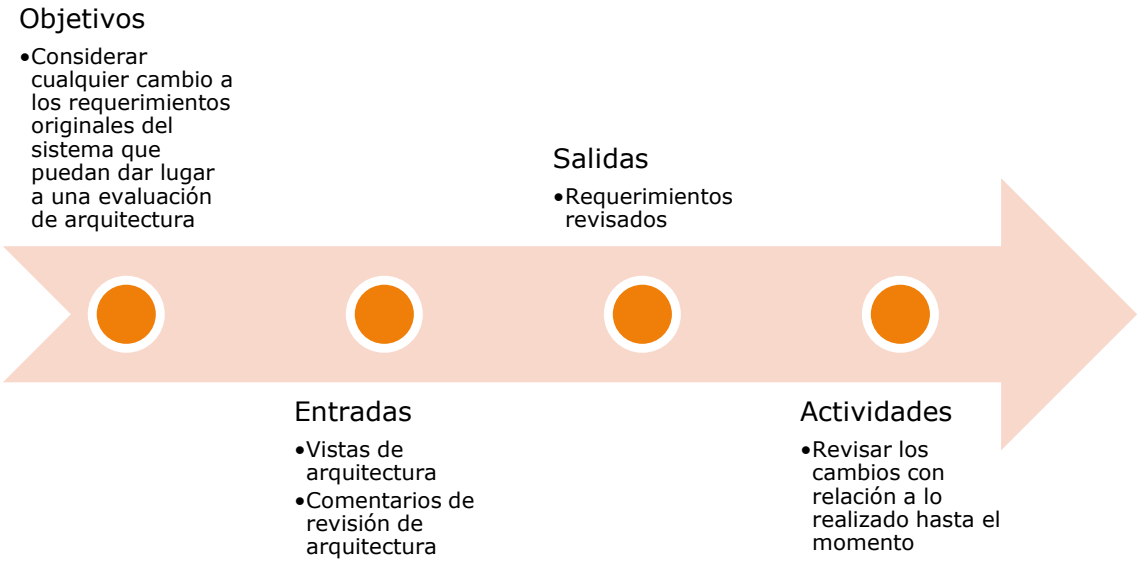
6. Evaluar la arquitectura con los stakeholders



7A. Revisar la arquitectura



7B. Revisar los requerimientos



Criterio de éxito del proceso

Principio

La definición de arquitectura puede considerarse completa una vez que los riesgos materiales que el sistema enfrenta hayan sido mitigados, lo cual se puede juzgar por la ausencia de comentarios después de la evaluación.

Estrategia

Inclúyase a si mismo como revisor de la descripción de arquitectura y no termine la definición inicial de arquitectura hasta que esté satisfecho que no hay problemas significativos con la arquitectura.

Buscar producir una descripción de arquitectura que es suficientemente buena para satisfacer las necesidades de los usuarios, en vez de una versión perfecta.

ESCENARIOS DE ARQUITECTURA

Objetivos

Comprender el concepto de estilos de arquitectura

Evaluar estilos de arquitectura existentes y comprender escenarios para su aplicación

Escenario

Un escenario de arquitectura es una descripción bien definida de la interacción entre una entidad externa y el sistema. Define el evento que dispara el escenario, la interacción iniciada por la entidad externa y la respuesta requerida por el sistema.

Los tipos de escenarios que se pueden construir son:

- Escenarios funcionales. Casi siempre son definidos en términos de una secuencia de eventos externos a los cuales el sistema debe responder en una forma particular
- Escenarios de calidad del sistema. Son definidos en términos de cómo el sistema debe reaccionar a un cambio en su ambiente de manera que exhiba una o mas propiedades de calidad

Los escenarios pueden ser útiles para:

- Proveer entradas para la definición de arquitectura
- Definir y validar el alcance del sistema
- Evaluar la arquitectura
- Comunicarse con stakeholders
- Encontrar requerimientos perdidos
- Dirigir el proceso de pruebas

Las fuentes para obtener información de escenarios son: requerimientos, stakeholders y experiencia.

Los escenarios deben ser priorizados, para lo cual se pueden considerar los siguientes criterios:

- Importancia que el stakeholder le da al escenario

- Riesgo probable que se considera en la implementación del escenario

Escenarios funcionales

Los elementos que definen un escenario funcional son:

Resumen	• Descripción breve de lo que el escenario trata de ilustrar
Estado del sistema	• Estado del sistema antes que el escenario ocurra (si es relevante)
Ambiente del sistema	• Cualquier observación significativa acerca del ambiente en el cual está corriendo el sistema • Tales como no disponibilidad de sistemas externos, comportamiento particular de infraestructura, restricciones de tiempo, etc.
Estimulo externo	• Una definición de que causa que el escenario ocurra • Tales como llegada de datos a una interfaz, ingreso del usuario, paso del tiempo, etc.
Respuesta requerida del sistema	• Una explicación, desde la perspectiva de un observador externo, de cómo el sistema debe responder al escenario

Ejemplo escenario funcional, actualización incremental de estadísticas

Resumen	• Como el sistema maneja cambios a algunos datos base
Estado del sistema	• Ya existen las estadísticas resumidas para el cuarto (quarter) a que se refieren las estadísticas • La base de datos cuenta con suficiente espacio para procesar la actualización requerida
Ambiente del sistema	• El ambiente de deployment está operando normalmente, sin problemas
Estimulo externo	• Una actualización a las transacciones de ventas del cuarto anterior arriba vía la interfaz de carga de datos
Respuesta requerida del sistema	• Los nuevos datos deben disparar automáticamente un procesamiento estadístico para actualizar las estadísticas resumidas para el cuarto afectado, para reflejar los datos actualizados • Los datos de estadísticas resumidas anteriores deben permanecer disponibles hasta que las nuevas estén listas

Escenarios de calidad del sistema

Los elementos que definen un escenario de calidad del sistema son:

Resumen	• Descripción breve de lo que el escenario trata de ilustrar
Estado del sistema	• Estado del sistema antes que el escenario ocurra (si es relevante)
Ambiente del sistema	• Cualquier observación significativa acerca del ambiente en el cual está corriendo el sistema • Tales como no disponibilidad de sistemas externos, comportamiento particular de infraestructura, restricciones de tiempo, etc.
Cambio en el ambiente	• Una explicación de que ha cambiado en el ambiente del sistema que causa que el escenario ocurra • Tales como cambios o fallas de infraestructura, cambios en comportamiento de otro sistema, ataques de seguridad, etc.
Respuesta requerida del sistema	• Una definición de cómo se debe comportar el sistema en respuesta a un cambio en su ambiente

Ejemplo escenario de calidad del sistema, actualización diaria triplica tamaño

Resumen	• Como se debe comportar el procesamiento de fin de día del sistema cuando el volumen de datos crece bastante
Estado del sistema	• El sistema tiene estadísticas resumidas en la base de datos para datos ya procesados y los elementos de procesamiento del sistema tienen una carga ligera
Ambiente del sistema	• El ambiente de deployment está operando normalmente, sin problemas • Los datos están llegando a una tasa constante de 1,000 a 1,500 transacciones por minuto
Cambio en el ambiente	• La tasa de actualización de datos para un día particular se incrementa a 4,000 transacciones por minuto
Respuesta requerida del sistema	• Cuando el procesamiento de fin de día inicia, el sistema debe procesar los datos del día hasta alcanzar el límite de tiempo configurado en el sistema • En ese punto, el sistema debe descartar el proceso, dejar disponibles las estadísticas resumidas anteriores y loguear un mensaje a la consola operacional del sistema

ESTILOS DE ARQUITECTURA

Un *patrón* es una solución reutilizable a un problema recurrente.

Un *estilo de arquitectura* es una clase de patrón que ocurre al nivel de arquitectura y se aplica a elementos de arquitectura como componentes y módulos.

Un estilo de arquitectura define un lenguaje consistente de tipos de elementos (módulos, componentes y conectores) y restricciones (que restringen como pueden ser usados los tipos de elementos).

Ventajas

- Conjunto prefabricado de restricciones
  Se puede pensar en un estilo como un conjunto prefabricado de restricciones con beneficios e inconvenientes conocidos. Como cualquier cosa prefabricada, se ahorra el trabajo de diseñarlo y debugearlo.
- Consistencia y entendimiento
  La consistencia provocada por las restricciones del estilo pueden fomentar una evolución limpia del sistema, que hace el mantenimiento mas fácil.
- Comunicación
  La comunicación entre desarrolladores es mejorada por el simple nombre del estilo, como publish-suscribe, que transmite concisamente el intento de diseño a otros desarrolladores.
- Reutilización de diseño
  Cuando se usa un estilo, se reutiliza un conjunto prefabricado de restricciones. Como resultado, cualquier ingeniero tiene el beneficio de reutilizar el conocimiento de diseño del ingeniero que lo inventó o seleccionó.
- Asegurar atributos de calidad
  Un problema con el código sin restricciones y arbitrario es que puede hacer cualquier cosa. Si se necesita que el código tenga cierta cualidad, como mantenibilidad, escalabilidad o seguridad, se debe restringir.

Estilo "gran bola de lodo" (big ball of mud)

Si el estilo de capas es el estilo mas buscado, este estilo es el que mas se alcanza.

Se caracteriza por la ausencia de cualquier estructura evidente o tal vez vestigios de una estructura erosionada.

También es típico compartir información de forma promiscua, a veces al grado que las estructuras de datos se vuelven globales.

Las reparaciones y mantenimiento son difíciles e implican parches en vez de soluciones elegantes. No es sorprendente que tales sistemas tienen pobre mantenibilidad y extensibilidad.

Los desarrolladores "que entienden" estos sistemas sienten seguridad y los que podrían limpiar "el lodo" generalmente se alejan.

Estilo en capas

Descripción

Tal vez es el estilo mas común, tan común que muchos desarrolladores asumen que todos los sistemas son o deben ser en capas.

Elementos y restricciones

El elemento esencial es una *capa* y la relación esencial es una relación *uses*, una especialización de la relación de dependencia.

Consiste de una pila de capas donde cada capa actúa como una *máquina virtual* para las capas arriba de ella y su ordenamiento forma un grafo dirigido acíclico.

Una capa solo puede usar la capa directamente bajo ella.

Calidades resultantes

Las restricciones llevan directamente a los atributos de calidad que promueve: *modificabilidad, portabilidad y reutilización*.

Ya que una capa depende solo de la capa directamente bajo ella, las capas subsecuentes pueden ser cambiadas o emuladas.

Pilas mas altas de capas llevan a mas oportunidades de sustitución, posiblemente a expensas de una ejecución eficiente (*rendimiento*).

Variantes

Variantes del estilo doblan la restricción de manera que una capa podría usar capas mas abajo.

Otra variante implica tener capas compartidas, donde cada capa puede usar esas capas compartidas (verticales).

Notas

El estilo puede variar grandemente de su forma platónica a la incorporada.

Generalmente se viola la restricción de comunicación de una capa a otra, lo que afecta los atributos de calidad.

Estilo pipe and filter

Descripción

Los datos fluyen a través de pipes (tuberías) a filtros que trabajan sobre los datos, similar a la forma en que un fluido podría fluir a través de pipes en una planta de procesamiento químico. Una característica clave es que la red de pipe and filter está procesando datos continua e incrementalmente.

Elementos y restricciones

El estilo consiste de 4 elementos: *pipes (tuberías), filtros, puertos de lectura y puertos de escritura.*

Cuando opera, un filtro lee alguna entrada de uno o mas puertos de entrada, hace algún procesamiento, y escribe la salida a uno o mas puertos de salida.

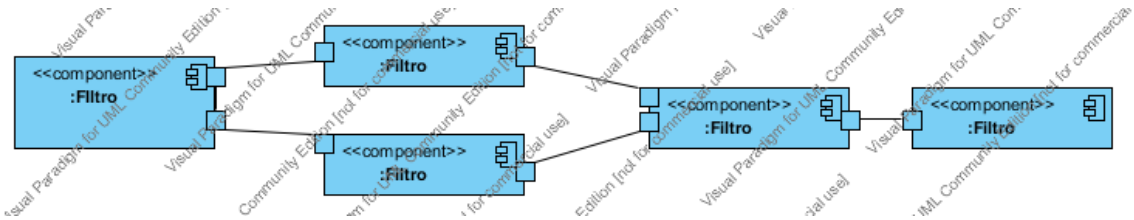
Los filtros pueden enriquecer, refinar o transformar los datos, pero los pipes solo deben transportar datos en una dirección, sin cambios y en orden. Se puede pensar que cada filtro como aplicar una *función* a su entrada.

En la red mas simple de pipe and filter, una lineal, los datos fluyen de una *fuente* a través de los pipes y filtros hasta que alcanza un *pozo* (sink).

Se requiere que los filtros sean independientes, no pueden interactuar entre sí, aún indirectamente, excepto a través de pipes y no pueden compartir estado entre sí.

Un filtro debe leer incrementalmente la entrada que recibe y, conforme procesa esa entrada, incrementalmente escribe su salida. La intención de esta restricción es mantener la red de pipe and filter funcionando en todo momento.

Lo correcto de la red de pipe and filter debe ser determinístico con respecto a la concurrencia. Independientemente que la implementación maneje la concurrencia, una entrada dada siempre debe producir la misma salida.



Calidades resultantes

Permite (re)*composición* tardía de una red, permitiendo *modificabilidad* o *reconfigurabilidad.*

Se podría no proveer una red, sino solo una colección de filtros hechos previamente para que sean ensamblados por otros. Estos filtros podrían ser *reutilizados* por usuarios.

Al trabajar con este estilo las oportunidades para concurrencia son mejoradas ya que cada filtro puede correr en su propio thread o proceso.

En general, este estilo es inapropiado para aplicaciones interactivas.

Variantes

Algunas veces la red es restringida a ser lineal.

Las redes son usualmente grafos dirigidos acíclicos, pero se pueden introducir loops.

Los filtros pueden obtener o poner datos de sus puertos de entrada.

Notas

Cuando se implementa una red se debe poner atención en cómo se detendrá.

En lo abstracto, los pipes son infinitamente rápidos y grandes. En la práctica los pipes son implementados con un buffer de tamaño limitado, lo cual puede impactar el rendimiento.

También habrá diferencias de rendimiento si los filtros están todos en el mismo espacio de memoria o en máquinas separadas.

Es importante distinguir 2 roles:

- Desarrollador de filtro

- Desarrollador de red de pipes and filter

Estilo batch-sequential

Descripción

Los datos fluyen de etapa a etapa y son procesados incrementalmente.

En contraste con pipe and filter cada etapa completa todo su procesamiento antes de escribir su salida.

Elementos y restricciones

Los componentes de procesamiento se llaman de diferentes formas*: etapas o pasos.*

Una tarea simple que fluye a través del sistema batch-secuencial es llamada un *batch o job.*

Este estilo tiene restricciones similares al estil pipe and filter. En particular, cada etapa es similarmente independiente.

Una etapa depende de los datos que toma, pero no de las etapas previas.

Generalmente es un sistema lineal de una serie de etapas. Ningún trabajo es hecho en los conectores, que simplemente pasan datos sin alterar a la siguiente etapa.

Calidades resultantes

Promueve los mismos atributos de calidad que el estilo pipe and filter, especialmente *modificabilidad* ya que las etapas son independientes entre sí.

Una diferencia es que donde el estilo pipe and filter produce su salida incrementalmente, este estilo la produce al final en cuyo caso la solución está completa o ausente, que impacta en la *usabilidad.*

Otra diferencia es que se tienen menos oportunidades de *concurrencia* ya que las etapas no pueden ser ejecutadas en paralelo.

Pueden tener un mayor *desempeño* (throughput).

Estilo centrado en modelo

Descripción

Componentes independientes interactúan con un modelo central (también llamado *data store o repositorio*) en vez de interactuar entre sí.

Puede usar una base de datos relacional, pero es usado mas a menudo en memoria

Un ejemplo es un IDE donde un modelo central representa el estado del programa editado y se presenta al usuario con varias vistas y controles

Elementos y restricciones

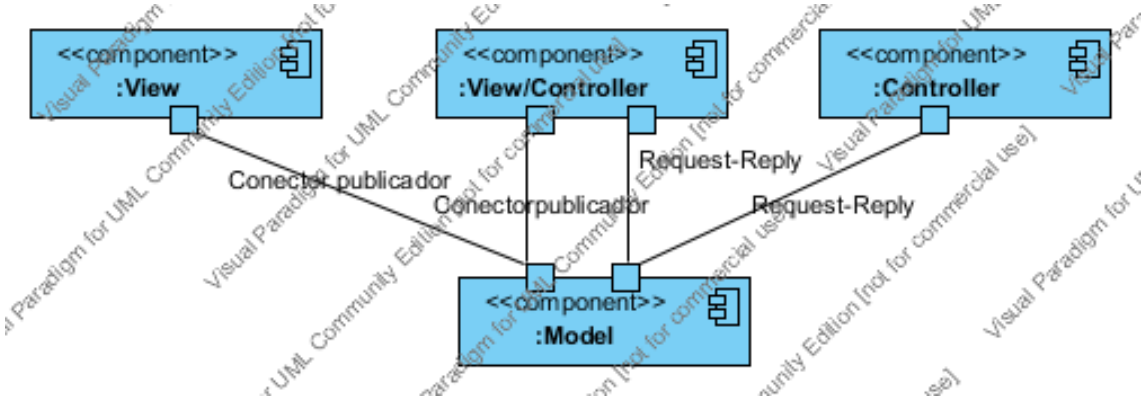
Cada sistema tiene un componente *modelo* y uno o mas componentes *view, controller o view controller.*

Los tipos de conectores pueden variar. Si el modelo implementa el patrón observador entonces los conectores notificarán a las vistas de los cambios, pero las vistas también pueden consultar el modelo.

Si se usa una base de datos relacional, se pueden usar triggers para causar notificaciones de actualizaciones.

Las vistas y controladores dependen solo del modelo, no entre sí. Hay un solo modelo compartido, muchas vistas y controladores.

Como en el MVC vistas y controles especiales pueden comunicarse directamente, sin pasar por el modelo, esto empaña su independencia con el beneficio de mejor rendimiento.



Calidades resultantes

Es altamente *modificable* por la independencia de los componentes vista y controlador del componente modelo y dependencias mínimas. También debido a que el productor y consumidor de información están desacoplados.

El sistema es *extensible* ya que es fácil agregar vistas y controladores no anticipados.

La *concurrencia* puede ser promovida ya que las vistas y controladores pueden correr en sus propios threads o procesos, o aún en diferente hardware.

Notas

Un punto de variación importante es si el modelo está estructurado o no previamente.

Algunas variantes hacen disponible una pila de datos no estructurados que son limpiados incrementalmente por las vistas y controladores.

Otras variantes proporcionan datos estructurados, pero no saben cómo serán usados por las vistas y controladores.

Estilo publish-suscribe

Descripción

Componentes independientes publican eventos y se suscriben a ellos.

Un componente que publica ignora la razón en el contexto general de porque un evento es publicado, los componentes suscritos no saben porque o quien publicó el evento.

Elementos y restricciones

Define 2 tipos de puertos, puertos que publican y puerto suscritos y un conector, un bus de eventos.

Cualquier tipo de componente puede publicar eventos (o suscribirse a ellos) mientras use un puerto que publica (o suscribe).

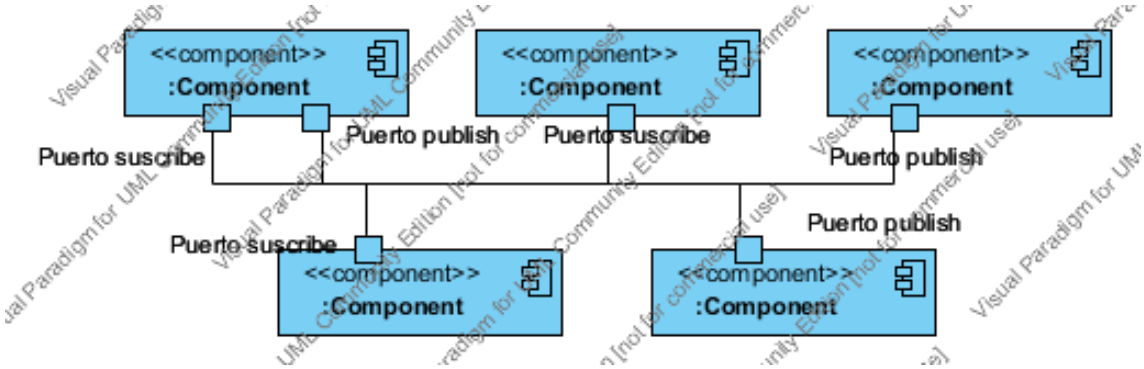
El bus de eventos es un conector en n vías en el cual varios puertos pueden agregarse a ellos. Consecuentemente, un componente puede publicar un evento y varios componentes pueden suscribirse a el.

Nótese que este estilo el conector es la "estrella", no los componentes y es responsable la mayor parte del trabajo. Es responsable de entregar eventos.

Los componentes que publican confían que los eventos son entregados a los suscriptores y los suscriptores confían que recibirán cualquier evento al que estén suscritos.

Los suscriptores dependen solo en el evento, no en el publicador de eventos. El publicador podría cambiar, sin afectar al suscriptor.

De forma similar el publicador es ajeno al consumo de los eventos.



Calidades resultantes

El beneficio principal del estilo es que desacopla publicadores y consumidores de eventos. Como consecuencia el sistema es mas *mantenible y evolutivo.*

El bus de eventos agrega una capa de indirección entre productores y consumidores, lo cual puede afectar el *rendimiento*. Sin embargo, el considerar utilizar componentes ya desarrollados para este propósito puede ser una opción.

Variantes

Algunas variantes requieren que los suscriptores se registren y desgregistren por eventos.

Otra variación implica creación dinámica de tipos de eventos, publicadores y suscriptores. Este es un ejemplo de una arquitectura dinámica.

Los buses de eventos varían en cuanto a las propiedades que soportan. Algunos son durables, en el sentido que garantizan que los mensajes no se perderán en caso de una falla. Esto se alcanza escribiendo eventos a un almacenamiento confiable, con el tradeoff de latencia.

Notas

Desde el punto de vista de mantenimiento y evolución, el estilo desacopla publicadores de eventos de los consumidores, pero no confundir esto con las intenciones y conocimiento del desarrollador. Solo un diagrama del patrón indicaría que todos los componentes se comunican con todos y no el tema de desacoplamiento.

Estilo cliente servidor y n capas

Descripción

Los clientes requieren servicios de los servidores.

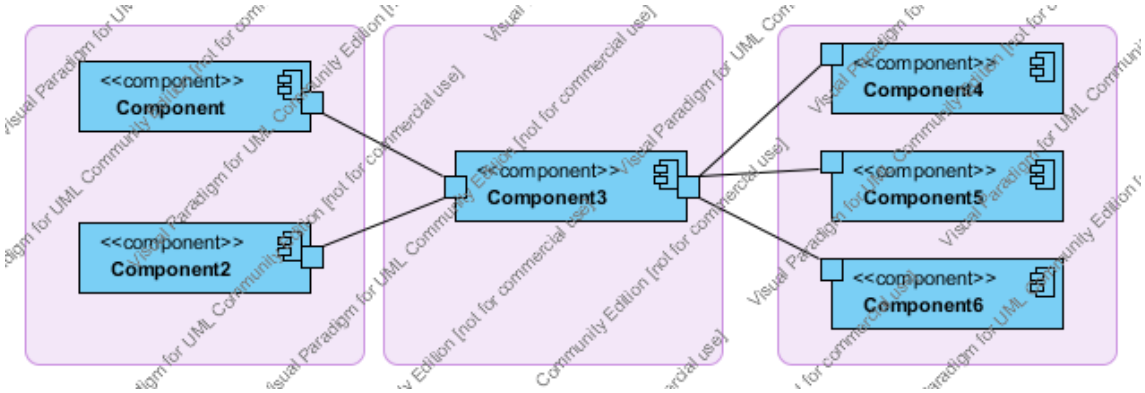
La solicitud es usualmente síncrona y a través de un conector request-reply, pero puede variar.

Elementos y restricciones

Contiene componentes cliente y componentes servidor y, usualmente, un conector request-reply y puertos.

Los clientes pueden iniciar comunicación pero no los servidores.

El servidor no sabe la identidad del cliente hasta que es contactado, pero los clientes saben la identidad del servidor o saben cómo buscarlo.



Calidades resultantes

Establece una poderosa relación asimétrica entre clientes y servidores en cuanto a quien puede iniciar el procesamiento, esto lleva a que el servidor tenga mas influencia ya que provee el servicio.

La *mantenibilidad* se promueve ya que se puede cambiar la implementación del servidor en un solo lugar, en vez de en todos los clientes.

Este control central ayuda a *la evolución* del sistema.

El estilo también puede ser usado para *integrar* sistemas existentes creando una fachada sobre el sistema existente y tratándolo como un servidor.

Variantes

El estilo tiene varios puntos de variación. Los conectores pueden ser síncronos o asíncronos, pueden existir límites en el número de clientes o servidores, las conexiones pueden ser con estado o sin estado (sesiones) y la topología del sistema puede ser dinámica o estática.

Una variante del estilo permite que el servidor, luego de ser contactado primero por el cliente, le envíe actualizaciones subsecuentes.

Otra variante es el estilo n-capas. Este estilo usa 2 o mas instancias del estilo cliente servidor para formar una serie de capas. Las solicitudes deben fluir en una sola dirección.

Un caso común es el sistema de 3 capas donde la capa de interfaz de usuario actúa como cliente de la capa de lógica de negocio, quien actúa como cliente de la capa de persistencia.

Notas

El estilo es similar al estilo centrado en el modelo, pero este tiene la restricción adicional que los componentes de vista y controlador no interactúan.

En la práctica los clientes en un sistema cliente servidor raramente interactúan, pero el estilo no lo prohíbe.

Estilo peer to peer

Descripción

Los nodos se comunican entre si como pares y las relaciones jerárquicas se prohíben.

Cada nodo tiene la habilidad, pero no la obligación, de actuar como cliente y como servidor.

El resultado es una red de nodos operando como pares, donde cualquier nodo puede solicitar o proveer servicios a otro nodo.

Elementos y restricciones

Los elementos en este estilo son similares a los del estilo cliente servidor. Sin embargo, donde el conector cliente-servidor forza un rol de cliente y de servidor, este estilo el conector tienen roles idénticos en cualquier extremo y permite solicitudes y respuestas.

La habilidad de actuar como servidor para cualquier otro nodo y hacer solicitudes a cualquier otro nodo es requerido pero no significa que todos los nodos deban estar interconectados entre sí.

Calidades resultantes

Las redes de pares son usadas a menudo para proveer acceso a recursos, como archivos en la red BitTorrent, con copias redundantes de los archivos mantenidos en nodos múltiples.

Consecuentemente la *disponibilidad* es promovida.

También promueve la *resistencia*, ya que es menos probable que los fallos de nodos individuales perjudiquen al sistema.

La red es altamente *escalable y extensible.*

Notas

Algunas de las fortalezas de las redes de pares se derivan de la interconexión entre nodos, pero esto puede verse afectado si un ciclo de nodos está disjunto de la red principal: una isla.

Estilo map-reduce

Descripción

Es apropiado para procesar grandes conjuntos de datos, como los encontrados en sistemas de escala de internet tales como motores de búsqueda o sitios de redes sociales.

Conceptualmente programas simple, como ordenar o buscar, pueden ejecutarse lentamente sobre grandes conjuntos de datos si se usara una sola computadora.

Conforme aumenta el número de computadoras usadas, la probabilidad de que una de ellas falle también se incrementa, de manera que el estilo permite la recuperación de tales fallos.

El gran conjunto de datos se parte en datasets (llamados splits) mas pequeños y almacenados en un filesystem global.

Uno o mas de estos datasets es procesado (mapeado) por un componente map worker y los resultados intermedios son escritos al filesystema local. Los resultados intermedios y los map workers son independientes, de manera que ningún map worker puede leer la salida de otro map worker.

Un componente reduce worker lee los resultados locales desde múltiples filesystem locales y combina (reduce) los resultados para producir un resultado final completo, que es almacenado en el filesystem global. Como en el caso de los map workers, los reduce workesrs son independientes.

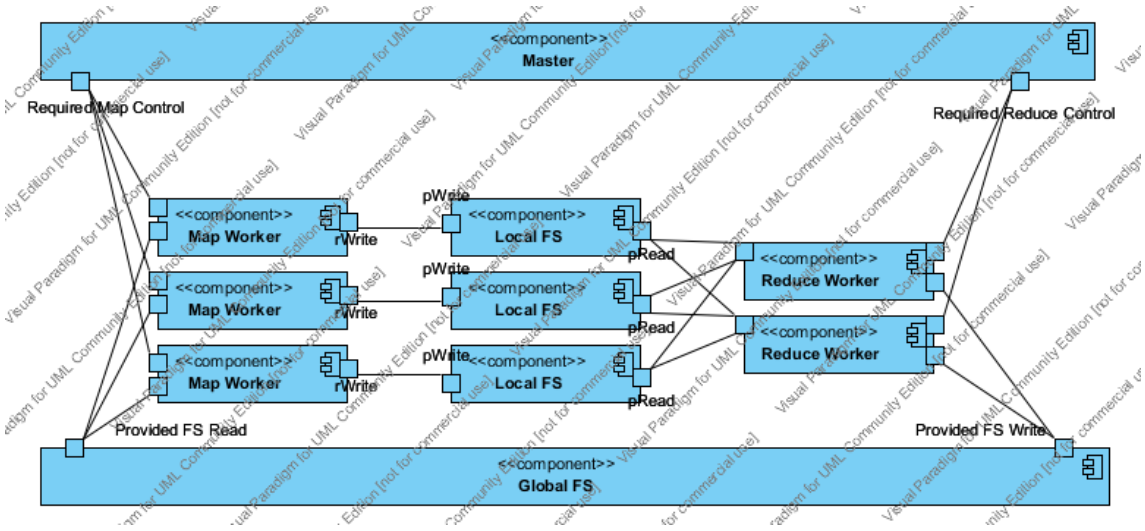
El master worker es responsable de instanciar los otros workers y asignar los splits a los map workers. También monitorea la salud de los workers y replanifica el trabajo cuando los workers fallan.

Elementos y restricciones

Un sistema map-reduce tiene un solo master worker y múltiples componentes map worker y reduce worker.

El master worker se comunica con los demás con un conector worker controller.

Los map workers pueden escribir datos a un filesystem local usando un conector de filesystem local y los reduce workers leen datos de manera similar. Ambos usan también un conector de filesystem global.



Calidades resultantes

El atributo de calidad primario que mejora este estilo es la *escalabilidad.*

Las tareas que eran imprácticas para computar con una sola computadora pueden ser divididas entre varias máquinas, mejorando el *rendimiento.*

Una vez un programa está escrito para usar este etilo, puede correr en un cluster de o uno o miles de máquinas.

También promueve la *disponibilidad*, ya que se recupera de fallas en máquinas replanificando el trabajo en otra máquina.

Notas

El rendimiento está altamente influenciado por la localidad de los datos. Los resultados intermedios necesitan ser mantenidos cerca de los componentes map y reduce worker para evitar uso de ancho de banda de red.

Usualmente es combinado con el estilo batch-secuencial, donde la salida de un job map-reduce provee la entrada para el siguiente. Cada job map-reduce es una etapa en la red batch-secuencial. La combinación de estos estilos puede transformar un problema que no era adecuado para map-reduce a uno que lo es.