Unidad II Elaboración de un Proyecto Distribuido

M.C. Juan Carlos Olivares Rojas

Temario

2.1 Diseño
2.2 Desarrollo
2.3 Documentación
2.4 GUI’s

2.1 Diseño

- El diseño de un sistema distribuido es similar al de un sistema centralizada en cierta forma.
- Se deben considerar todas aquellas consideraciones que involucran por definición los sistemas operativos.
- Por ejemplo, se pueden utilizar técnicas como los diccionarios de datos distribuidos.

Diccionario distribuidos

- Es una técnica de especificación formal que sirve para representar los requerimientos de una aplicación:

insertar = ***procedimiento***(*x: elemento*)

requiere *x Ï Miembros*

efectos *Agrega x a Miembros*

Redes de Petri

- Otra técnica de especificación formal que ayuda en el modelado de un sistema distribuido son las Petrinets.

- Una red de Petri es un grafo dirigido G=(N, V) a cuyos componentes se les llama lugares (estados) y transciones (eventos)

- Ayudan además en la sincronización de procesos entre otros.

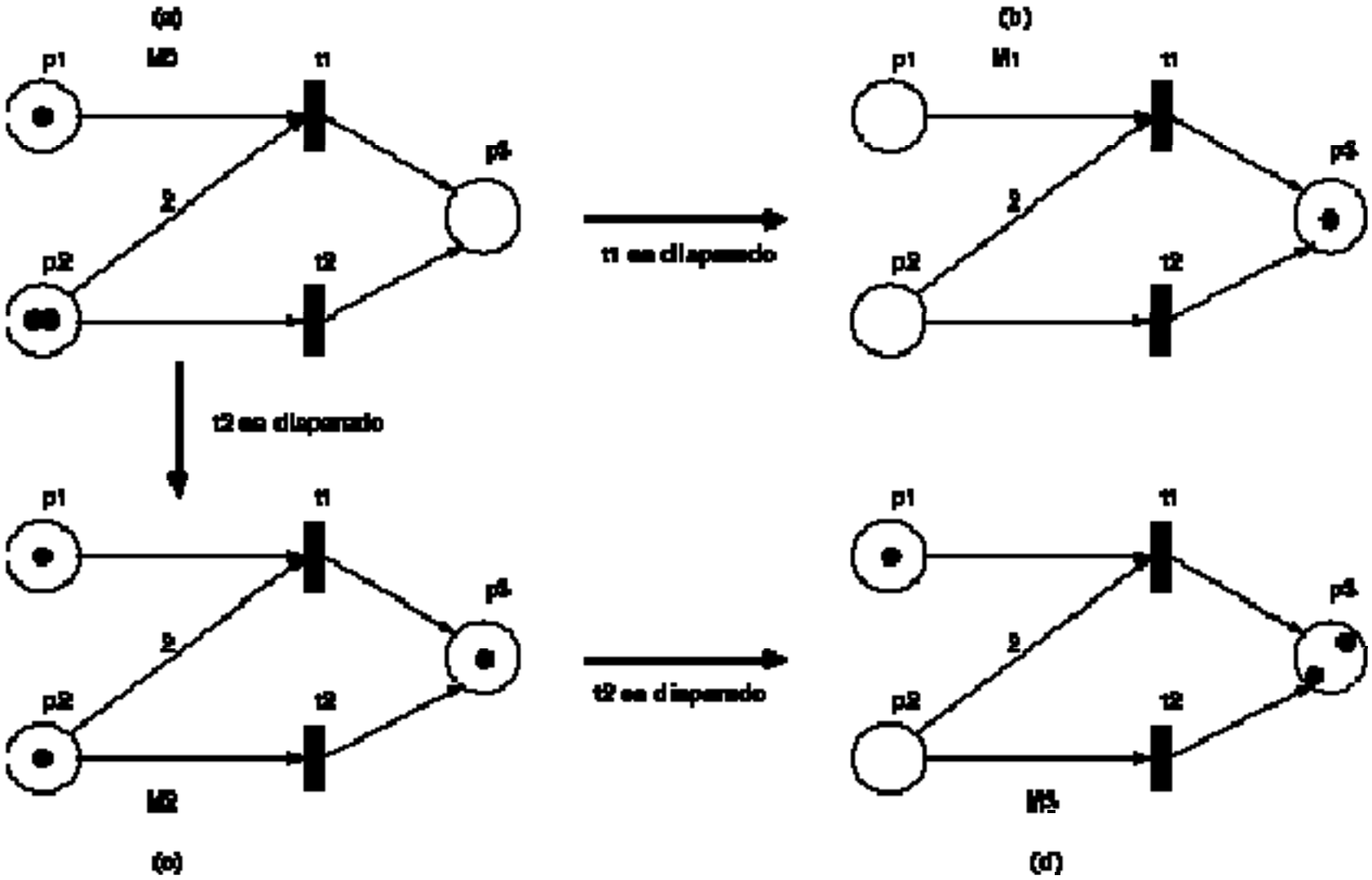
Redes de Petri

- Formalmente una red de Petri está formada por: RP = (P, T, I, O), donde I es una función de entrada y O es una función de salida.

- Dentro de cada lugar, se encuentran ciertas marcas o token que se desplazan a otros lugares cuando se activa una transición

- Existen variantes como las Redes de Petri coloreadas.

Redes de Petri



Diseño

- Se pueden utilizar otros lenguajes de modelado como UML para especificar

- Los diagramas de UML más utilizados en el modelado de sistemas distribuidos son: diagramas de caso de uso, de secuencia, de estado, de actividades, de componente y de despliegue.

Diseño de Protocolos

- La parte más importante del diseño de un sistema distribuido es el protocolo de comunicación a nivel de aplicación, ya que aquí se gestionan los mecanismos de solicitud-respuesta.

- Un protocolo es un conjunto de reglas que se deben seguir para tener una comunicación exitosa.

Diseño de protocolos

- Los diseñadores pueden crear su protocolo el cual debe de estar abierto para que se puedan implementar clientes y servidores.

- Por ejemplo el servicio POP3 tiene definido los siguientes verbos o acciones: USER, PASS, STAT, DELE, RETR

Diseño de protocolos

- Ejemplo: Un banco necesita brindar a sus clientes servicios de consulta de cuentas a sus clientes. El banco diseño su propio protocolo el cual se muestra a continuación:

- CS*<num_cta> Consulta de Saldo
- S*<cant_sal> Cantidad saldo
- CC*<usuario> Consulta de cuentas
- IC*cta_1*….*cta_n

Modelado de concurrencia

- Se puede aplicar teoría de colas para ver el comportamiento que tiene un(os) servidor(es) con varios clientes.

- Se pueden realizar simulaciones para determinar todos los posibles puntos de falla y ver el funcionamiento general del sistema.

- Otro método son las pruebas de stress.

2.2 Desarrollo

- Para el desarrollo de una aplicación distribuida, se deben tomar en cuenta algunos detalles que a continuación se presentan:

- Mantenimiento de los datos más usados en un almacén rápido (caché).
- Mantenimiento de los datos cerca de donde se requieren.

Desarrollo

- Realizar duplicación de datos lo más que se pueda.
- Eliminar cuellos de botella.
- Emplear paralelismo
- Comprimir datos.
- Minimizar la latencia.
- Poner el mayor procesamiento posible dentro del cliente.

Desarrollo

- Poner todas las actividades de cómputo intensivo (audio, video, interfaces gráficas, etc.) en el cliente.
- Administrar todos los recursos compartidos en el servidor.
- Evitar la centralización de servicios.
- Usar procesos en capas para lograr mayor escalabilidad.
- Manejar esquemas Pull en lugar de Push

Desarrollo

- Minimizar la transferencia de datos entre clientes y servidores.
- La caché debe cambiar lentamente o bien mantener datos estáticos.
- Se debe tener puntos de revisión dentro de la transferencia de grandes volúmenes de información.
- Se debe administrar de manera centralizada con propagación distribuida

Desarrollo

- Se debe tener una administración remota y monitoreo.
- Se deben construir perímetros de seguridad
- Se deben utilizar modelos de cola para evitar los puntos críticos.
- Se debe diseñar con compatibilidad para atrás.

Metodología de Desarrollo

- Consta de cuatro fases:

- Localización de funciones
- Distribución de recursos
- Mapeo a la realidad
- Diseño completo

Localización de funciones

- Identificar funciones
- Maximizar el procesamiento en los clientes
- Identificar servicios virtuales

Distribución de recursos

- Crear capas
- Predecir cargas de trabajo
- Modelar la arquitectura (interacciones, confiabilidad, rendimiento)
- Analizar los resultados del modelo

Mapeo a la realidad

- Configurar productos y tecnologías
- Medir la interoperabilidad del producto
- Relocalizar componentes del sistema

Diseño completo

- Base de datos
- GUI
- Interfaces entre clientes y servidores
- Interfaces externas
- Recuperación de errores
- Administración
- Seguridad
- Lógica de la aplicación

2.3 Documentación

- La documentación de un sistema distribuido se realiza de manera muy similar a un sistema centralizado, sólo se debe tener en cuenta que existen dos componentes o pequeñas aplicaciones dentro de cada sistema (el cliente y el servidor).

- Se debe detallar el protocolo de aplicación en el manual técnico.

2.4 GUI's

2.4.1 GTK
2.4.2 QT
2.4.3 Motif
2.4.4 Python
2.4.5 Tk/tcl

Interfaces Gráficas de Usuario

- La llegada de la arquitectura cliente/servidor ha traído como consecuencia que la lógica de presentación haya mejorado mucho, hablando incluso de clientes ricos.

- Algunas interfaces gráficas manejan una arquitectura cliente/servidor, la más famosa es X11.

X11

- Es un administrador de primitivas gráficas compuesta por un proceso servidor encargado de dibujar los objetos en pantalla y un proceso cliente denominado terminal X. También cuenta con procesos de teclado y ratón.

- Esta arquitectura permite manejar interfaces gráficas de manera remota en clientes con pocas capacidades.

2.4.1 GTK

- Es un grupo de librerías encargadas de la gestión de ventanas.
- Está asociado con el proyecto GNOME.
- Es la abreviatura de Gimp ToolKit.
- Es software libre, disponible bajo licencia LGPL.

GTK

- Se puede programar interfaces usando lenguajes como C, C++, Java, Perl, C#, entre otros.

- Se basa fundamentalmente en la librería Glib. La versión más actual es GTK+2.

- Existen herramientas como glade, gambas que ayudan a este tipo de desarrollo.

2.4.2 QT

- Es una biblioteca multiplataforma para el desarrollo de interfaces gráficas de usuario.
- Es desarrollado por la compañía Trolltech. Es utilizada en el gestor de ventanas KDE.
- Se programa nativamente en C++, pero existen versiones para lenguajes como Perl, Java, C, Ruby, entre otros.

QT

- Es una herramienta de código abierto pero no libre.
- Actualmente QT, maneja actualmente un esquema de licencias doble GPL para software libre y licencia de pago para aplicaciones cerradas.
- Se ha portado con éxito para entornos de cómputo móvil: Qtopia.

2.4.3 Motif

- Fue una de las primeras bibliotecas gráficas para el desarrollo de entornos gráficos en ambientes Unix.

- Es una librería descontinuada por el uso de otras más robustas como GTK y QT.

- Muchos sistemas Unix siguen utilizando este tipo de interfaces

2.4.4 Python

- Es un lenguaje de scripting al estilo de Perl, Ruby, etc.
- Cuenta con un amplio soporte para interfaces gráficas.
- Es un proyecto de software libre, de los inicadores de este movimiento.

2.4.5 Tk/tcl

- Tcl/Tk es la abreviación de: Tool Command Language/ ToolKit

- Es un lenguaje de scripting utilizado en el diseño de prototipos rápidos con interfaz gráfica de usuario.

- Ha perdido terreno por el uso de interfaces gráficas como GTK y QT.

Tendencias en el Desarrollo de Interfaces de Usuario

- Las nuevas tendencias en el desarrollo de interfaces radican en la Web, con el surgimiento de la Web 2.0 y los RIC (Rich Internet Clients)

- Otras tendencias radican en el uso de desarrollar interfaces en base a lenguajes XML como XUL, XAML, entre otros

Referencias

- R. Pressman, “Ingeniería del Software”, 5ª. Edición, McGraw-Hiil, España, 2002.

- R. Johnsonbaug, Matemáticas Discretas, 4a. Edición, Prentice Hall, México, 1999, ISBN: 970-17-0253-0.

Referencias

- R. Orfali, et al., “Cliente/servidor y objetos. Guía de supervivencia”, tercera edición, Oxford University Press, México, 2002, ISBN: 970-613-597-9.

- W. Inmor, “Developing Client/Sever Applications”, Wiley, Estados Unidos, 2003, ISBN: 0-471-56906-2.

Referencias

- D. Dewire, “Client/Server Computing”, McGraw-Hill, Estados Unidos, 1993, ISBN: 0-07-016732-X.

- W. Marion, “Client/Server Strategies”, McGraw-Hill, Estados Unidos, 1994, ISBN: 0-07-040539-5.

Referencias

- P. Renaud, “Introduction to Client/Server Systems”, Wiley, Estados Unidos, 1993, ISBN: 0-471-57773-1.

- P. Kimmel, “Manual de UML. Guía de aprendizaje”, McGraw-Hill, México, 2006, ISBN: 0-07-226182-X.

Referencias

- J. Senn, “Análisis y Diseño de Sistemas de Información”, 2da. Edición, McGraw-Hill, México, 1992, ISBN: 968-422-991-7.

- A. Tanenbaum, et al., “Sistemas Operativos. Diseño e implementación”, 2da. Edición, Prentice Hall, México, 1998, ISBN: 970-17-0165-8.

¿Preguntas?