UNA ARQUITECTURA DE COMPONENTES DE SOFTWARE PARA EL DOMINIO DE REDES DE DISTRIBUCIÓN

WILLIAM RIVAS JONÁS MONTILVA

GIDatos -IUT-ULA. Mérida -Venezuela
william_ve@yahoo.com *jonas@ing.ula.ve*

RESUMEN

En este artículo se propone una arquitectura de software basada en componentes reutilizables dirigidos al dominio de redes de distribución. Su utilidad práctica está en el proceso de diseño de aplicaciones de software que involucren el uso de redes hidráulicas, oleoductos y acueductos. Para su elaboración, es usado un método de Ingeniería de Dominios denominado Sherlock adaptado a fin de asegurar la interoperabilidad de los componentes en plataformas heterogéneas.

Paclabras clave: Componentes de software; reutilización de software; componentes de software distribuidos; Software para redes de distribución.

ABSTRACT

In this article we propose a software architecture based on reusable components. The architecture is oriented to be used in distribution networks. Its practical utility lies in the capability to support the design of software applications for hydraulics, pipelines and aqueducts domain. For its elaboration a method of Domain Engineering called Sherlock is used and was adapted in order to assure interoperability in a heterogeneous environment.

Key words: Software components, reusable software, distributed networks, software for distributed networks.

INTRODUCCIÓN

Una red de distribución es una estructura espacial formada por entidades físicas interconectadas por medio de otras entidades lineales, a través de las cuales circula materia, bienes, personas o información desde un punto de la red a otro.

Existe una gran diversidad de dominios que requieren herramientas de software para la gestión de redes de distribución. En Ingeniería, es común el uso de software basado en redes para resolver problemas comunes de modelado, simulación, y cálculos. Así, por ejemplo, en ingeniería eléctrica se usa software para el modelado de redes de distribución de energía eléctrica; en ingeniería de fluidos se emplea software para modelar y simular redes hidráulicas; en ingeniería química se emplea en las redes de oleoductos; en ingeniería civil en cálculos para la gestión de redes de transporte, etc..

Algunas de estas herramientas son software de calidad propietaria, es decir, desarrollado por empresas especializadas. Otras utilizan software hecho a la medida, el cual requiere componentes especializados y desarrollados exclusivamente para cada aplicación. Pese a las diferencias que existen entre estas herramientas de software, hay en ellas muchos elementos, requerimientos y componentes comunes, lo cual, si se desea codificar de nuevo implica un esfuerzo redundante en el desarrollo de las mismas. Esta redundancia podría minimizarse si se dispone de un conjunto de componentes de software que pueda ser reutilizado en el proceso de desarrollo de herramientas y aplicaciones. Estos componentes pueden conectarse unos con otros para construir nuevos sistemas de una manera más rápida y económica. El resultado de reutilizar estos componentes se traduce en aplicaciones de software mucho más confiables, eficientes y fáciles de mantener (Heineman y Councill, 2001). Esta manera de desarrollar software es conocida en la literatura bajo de nombre de reutilización de componentes.

El objetivo del presente trabajo es proponer un modelo o arquitectura de componentes de software reutilizables para el dominio de redes de distribución. Para ello, es considerado el método de Ingeniería de Dominios Sherlock (Predonzani et. al., 2000) adaptado a fin de asegurar la interoperabilidad de componentes de software en plataformas heterogéneas. La noción de grafo espacial, descrita en (Montilva, 2000) y utilizada en la definición de patrones de diseño para grafos espaciales (Montilva y Ramos, 2001) constituyen marcos de referencia para el diseño de los componentes.

La principal ventaja que la arquitectura presenta es la facilidad de realizar el proceso de diseño de nuevas aplicaciones en el dominio de redes de distribución; pues, sirve de marco o patrón de diseño que abstrae todos aquellos aspectos y componentes comunes a todas las aplicaciones de ese dominio.

Este artículo está estructurado de la siguiente forma: en la siguiente sección se trata brevemente la descripción del método Sherlock, señalando brevemente las adaptaciones realizadas y resultados obtenidos. La sección 3 muestra cómo reutilizar los componentes de la arquitectura en programas clientes Java. Finalmente, se presentan las conclusiones y se señala el trabajo futuro a seguir.

El método y su aplicación al dominio de redes de distribución

Existen varios métodos para el desarrollo de componentes de software basados en el análisis de dominio. Entre los métodos usados por la comunidad del software se destacan los siguientes: FODA (Krut, 1993), JODA (Holibaught, 1993), y Sherlock (Predonzani et. al., 2000). El método FODA se origina en 1990 en el Instituto de Ingeniería de Software de la Universidad Carnegie-Mellon *(SEI - Software Engineering Institute),* el método enfatiza el análisis del dominio para identificar características o funcionalidades comunes y variantes entre aplicaciones dentro del dominio analizado. El método JODA nace en el grupo *"Joint Integrated Avionics Working Group"* (JIAWG) en 1992 y hace énfasis en la consulta de expertos del dominio para realizar el modelado y construcción de la arquitectura de software. La reutilización de código es también un aspecto importante en el método JODA. El método Sherlock (Predonzani, Succi, Breñaza, 2000) permite realizar el análisis e Ingeniería de dominio para la construcción de una línea de aplicaciones. El método surge ante la necesidad de caracterizar productos y considerar el análisis económico en la producción de *software*.

El método Sherlock aventaja a los anteriores debido a que incorpora el lenguaje estándar de modelado UML (Unified Modelling Language), lenguaje exitosamente empleado en el desarrollo de sistemas de software (Booch, Jacobson, Rumbaugh, 1998). Sherlock presenta como producto final un marco

de trabajo que puede emplearse en Ingeniería de Dominio para el desarrollo de aplicaciones similares.

El método Sherlock consiste en cinco fases: (I) Definición del dominio, (II) Caracterización del dominio, (III) Alcance del dominio, (IV) Modelado del dominio y (V) Desarrollo marco de trabajo.

En la definición del dominio se recolecta y se clasifica la información básica del dominio; en la caracterización del dominio se evalúan los valores internos y externos del producto a desarrollar desde el punto de vista de mercado. En la fase de alcance del dominio, se delimita el espacio de variabilidad de la aplicación. En la fase de modelado del dominio, las aplicaciones son modeladas desde la perspectiva del usuario valorando una posible estructura interna representada por medio de diagrama de clases. Finalmente, en el proceso de desarrollo del marco de la arquitectura se utiliza la Ingeniería de dominio para implementar componentes de software.

El método Sherlock está orientado a la construcción de líneas de aplicaciones para un mercado altamente competitivo. Por esta razón, el método es adaptado para que pudiera enfocarse a la construcción de componentes reutilizables en un contexto educacional: En la fase de definición de dominio, se introduce el análisis del contexto para describir en detalle el contexto de la aplicación. En la fase de caracterización se considera una lista de requisitos variantes provenientes del análisis del dominio.

Seguidamente se describen brevemente cada una de las fases del método Sherlock adaptado y resultados.

Fase I: Definición del dominio: El dominio general seleccionado para este trabajo es el de las redes de distribución. Este dominio es dividido en tres subdominios: redes de distribución de electricidad, redes de distribución de agua potable (acueductos) y redes de distribución de crudo (oleoductos). Para cada uno de estos subdominios, es construido un diccionario básico de dominio a fin de evitar confusiones en el manejo de términos. La estrategia para proveer información a las fases posteriores del método fue la búsqueda y evaluación de aquellas aplicaciones más usadas y disponibles. Las aplicaciones consideradas para el análisis de variantes fueron: Flow Pro 2.0 (ProSoft Apps), Mike Net 2000 (DHI Water & Environment), AFT Impulse (Applied Flow Technology), Sistema de Cadela para protección de redes (SID), Flow Map 2.1 (Utrecht University), además de los manuales de aplicaciones comerciales, como por ejemplo, OeX (Dynamic Solutions Ltd.).

Fase II: Caracterización del dominio: Las funcionalidades las aplicaciones seleccionadas en la Fase I son comparadas a fin de identificar características comunes y sus variantes.

El componente abstracto para las redes es denominado RedEspacial, esto por poseer características espaciales. Consta de una estructura de nodos, enlaces y funcionalidades típicas de los grafos. Además, se identifican componentes abstractos para cada uno de los subdominios tales como el Tanque, Juntura, Tubo, Bomba, etc.

Para cada componente se definieron los tipos de usuarios y los diagramas de flujo de usuarios, quedando estos claramente especificados mediante el lenguaje UML.

Fase III: Alcance del Dominio: En esta Fase se dispone el espacio de variabilidad de los componentes. El espacio de variabilidad está definido por los diferentes conceptos o características que pueden ser seleccionados por un usuario potencial para el descarte o desarrollo de una nueva aplicación. Los puntos de variación establecen los aspectos que diferencian las aplicaciones de dominio. Los puntos de variación pueden extenderse a medida que se adquiere más conocimiento del dominio. En el caso del subdominio de redes hidráulicas, los puntos de variación son:

- PV1 Modelado de red
- PV2 Cálculo
- PV3 Reporte

Cada uno de estos puntos de variación posee puntos internos los cuales pueden establecer ligeras diferencias entre las aplicaciones. En particular para el primer punto de variación se dispone:

- VP11 Modelado Estructural/Estático
- VP12 Modelado Transitorio
- VP13 Modelo sin pérdida
    - VP131 Cálculo potencia de bombas
    - VP132 Cálculo de flujo
    - VP133 Cálculo de presión

- VP14 Modelo con pérdida
  - VP141 Cálculo potencia de bombas
  - VP142 Cálculo de flujo
  - VP143 Cálculo de presión
- VP14 Temporizado
  - VP141 Tiempo real

A nivel de implementación, los puntos de variación son reflejados en las interfaces de los mismos componentes. Las variantes son implementadas mediante técnicas usadas en la programación orientadas a objetos tales como parametrización, sobrecarga, herencia, interfaces y agregación.

Modelado del Dominio y Arquitectura

Los aspectos comunes entre las aplicaciones, identificados en la Fase II, pueden convertirse en componentes reutilizables. Conceptualmente tales aspectos son agrupados en diferentes niveles: a) Nivel de Interfaz b) Nivel de clase c) Nivel de paquete. A nivel de interfaz, la agrupación es formada por el conjunto de operaciones comunes entre los elementos identificados y que además son accesibles desde las aplicaciones cliente. La agrupación a nivel de clase es formada precisamente por las características comunes y operaciones pertinentes a esas clases. La agrupación a nivel de paquete es formada por interfases y clases que conceptualmente están muy relacionadas pero existe diferencias claramente definidas.

Como resultado de un proceso de abstracción entre los tres subdominios considerados se tiene un modelo de componente genérico para crear y manipular redes. Este es el componente RedEspacial que se muestra en la Figura 1 descrito en alto nivel de abstracción mediante el diagrama de clases UML.

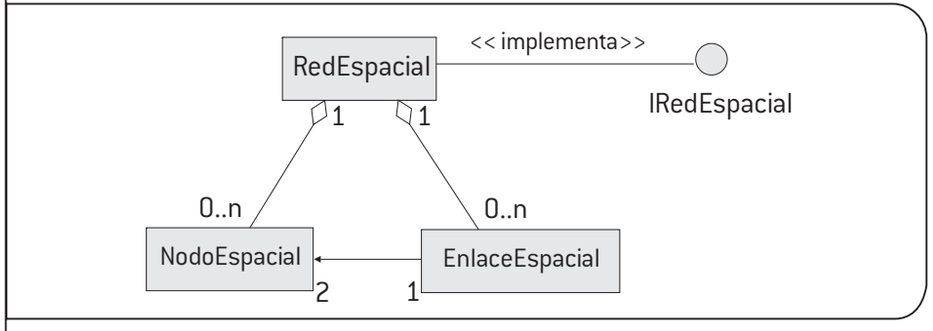

Figura 1. Componente RedEspacial **Fuente:** William Rivas –Jonás Montilva

El componente RedEspacial mediante su interfaz IRedEspacial presenta operaciones comunes usadas en los grafos, tales como flujo máximo, camino mínimo, árbol de expansión mínimo, búsqueda primero en profundidad y el reporte de conectividad del grafo. Estas operaciones son comunes en estructuras de grafos y son ampliamente usadas en Ingeniería (Montilva, 2000).

La organización lógica de estos elementos y sus relaciones constituye la arquitectura de dominio. En la Figura 2 se muestra la organización lógica de paquetes de la arquitectura propuesta. El paquete de interfaz GUI agrupa elementos reutilizables que permiten ofrecer un frente visible al usuario donde es posible definir y desplegar gráficamente la conectividad de la red. El paquete de servicios agrupa las clases e interfases que facilitan la instanciación de estructuras de datos para representar las redes y asociar elementos comunes a ellas. Dentro del paquete de servicios existen tres paquetes más, cada uno asociado a un subdominio. En el paquete Espacial se tiene componentes para la representación espacial en gráficos tal como representación XY. Finalmente, el paquete administrador de base de datos contiene componentes reutilizables que permiten recuperar datos espaciales desde una base de datos remota, cuyo acceso se realiza mediante EJB (Enterprise Java Bean) de tipo sesión y el protocolo JDBC (Java Data Base Connectivity).

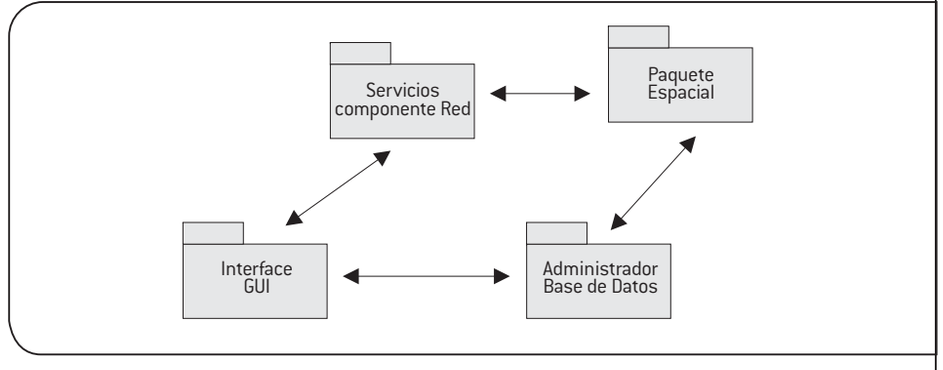

Figura 2. Paquetes en la arquitectura de componentes. **Fuente:** William Rivas –Jonás Montilva

Como ejemplo, en la Tabla 1 se muestra algunos de los componentes de los paquetes servicios de red hidráulica e interfaz gráfica de usuario.

Paquete	Clases	Componente	Interfaz	Operaciones de Interfaz
Servicios Red. Hidráulica	Tanque	Tanque	ITanque	capacidad[ ] nivel[ ] ...
	Bomba	Bomba	IBomba	potencia[ ] eficienciaHP[ ] ...
	Válvula	Válvula	IValvula	abrir[ ] cerrar[ ] ...
	Juntura	Juntura	IJuntura	diámetro[ ] ...
	Tubo	Tubo	ITubo	longitud[ ] ...
InterfaceGUI	VisorXY Punto E	VisorXY	IVisorXY	desplegar[ ] aumentar[ ] ...

Tabla 1. Algunos elementos de la arquitectura de componentes

Implementación o desarrollo del marco

Para la implementación del marco se considera las técnicas para el desarrollo de componentes de la especificación básica de Enterprise Java Bean implementada en el servidor J2EE (Java 2 Enterprise Edition). Para la implementación de cada componente de software de la arquitectura, se sigue como lo describe (Gail y Anderson, 2002):

a) Creación de interfaz Home
b) Creación de Interfaz Remota
c) Creación de clases que implementa la interfaz remota y clases auxiliares
d) Creación archivo descriptor ejb.xml para cada uno de los componentes
e) Despliegue y prueba de los componentes

Uso de componentes de la arquitectura

Al momento de desarrollar una aplicación es fácil reutilizar los componentes de la arquitectura. Para esto se debe obtener una referencia a la interfaz remota de los componentes y luego invocamos las operaciones deseadas.

Para ilustrar cómo se usa la arquitectura, considérese una red de transporte abstracta mostrada en la Figura 3. Aunque este un ejemplo trivial sirve con propósito ilustrativo de cómo emplear el *framework* desarrollado.

El problema es obtener otra red en la cual se transporta el máximo flujo de material desde el nodo origen A al nodo destino D. El primer número sobre los enlaces es el flujo actual y el segundo número es la capacidad máxima del enlace. El problema es resuelto por el clásico algoritmo Ford-Fulkerson (Cormen, Rivest y Stein, 2001), por tanto, es suficiente trabajar con el objeto que representa la red inicial, e iterar sobre ella según el algoritmo de Ford-Fulkerson. Esto equivale a enviar un mensaje al objeto RedEspacial para calcular la red de máximo flujo lo cual retorna un objeto del mismo tipo

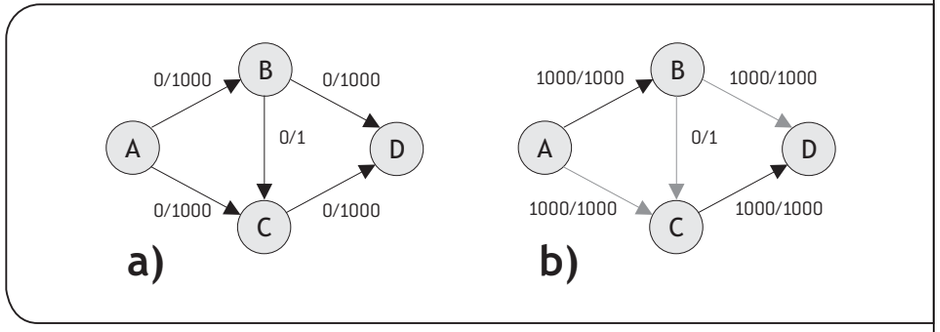

Figura 3. Red de transporte simple a) Red inicial b) Red de flujo máximo
Fuente: William Rivas –Jonás Montilva

Debido a que la arquitectura consta de componentes distribuidos, es necesario emplear mecanismos para localizar los componentes. Este mecanismo es provisto por la plataforma J2EE y es llamado JNDI *(Java Naming Directory Interface)*. Para ello se emplea un objeto de contexto implementado en el paquete *javax.naming* invocando el método *initialContext()* en la aplicación cliente. Sobre este objeto creado se invoca el método *lookup()* pasando como parámetro el nombre del componente que se desea reutilizar. Luego se obtiene una referencia a la interfaz *Home* con el método *narrow()* de la clase PortableRemoteObject.

Finalmente, empleamos la referencia de la interfaz Home invocando el método *create* heredado de EJBHome para obtener una referencia a la interfaz remota del componente. Luego usamos la interfaz remota para acceder a todas las funcionalidades del componente como si fuese un objeto local.

A continuación se muestra el código correspondiente para emplear el componente RedEspacial RedEspacial a fin de resolver el problema anterior.

```
…
try {
…
//Adquiere información de contexto ambiente J2EE
contexto  =  new InitialContext();
//Busca una referencia de contexto al componente RedEspacial y
obtiene //referencia de su interfaz Home°
Object objRed  =  contexto.lookup (“RedEspacial”);
//Obtiene referencia a interfaz Home del componente RedEspacial
RedEspacialHome h_red =
(RedEspacialHome)PortableRemoteObject.narrow(objRed
,RedEspacialHome.class);
//Igual para NodoEspacial y EnlaceEspacial
Object objNodo = contexto.lookup (“NododEspacial”);
NodoEspacialHome h_nodo =
(NodoEspacialHome)PortableRemoteObject.narrow( objNodo,
NodoEspacialHome.class);
Object objEnl = contexto.lookup (“EnlaceEspacial”);
EnlaceEspacialHome h_enl = (EnlaceEspacialHome)
PortableRemoteObject.narrow(objEnl, EnlaceEspacialHome.class);
//Obtiene referencia a interfaz remota del componente
RedEspacial
IRedEspacial red = h_red.create(“Red Simple 01”);
//Obtiene referencia a interfaz remota del componente
NodoEspacial
INodoEspacial nodoA = h_nodo.create(“A”);
INodoEspacial nodoB = h_nodo.create(“B”);
INodoEspacial nodoC = h_nodo.create(“C”);
INodoEspacial nodoD = h_nodo.create(“D”);
//Realiza conectividad de la red a la vez obteniendo referencia
a //interfaz remota del componente EnlaceEspacial
IEnlaceEspacial e1 = h_enl.create(nodoA,nodoB,1000);
IEnlaceEspacial e2 = h_enl.create(nodoA,nodoC,1000);
IEnlaceEspacial e3 = h_enl.create(nodoB,nodoC,1);
IEnlaceEspacial e4 = h_enl.create(nodoB,nodoD,1000);
IEnlaceEspacial e35= h_enl.create(nodoC,nodoD,1000);
//Invoca operaciones definidas en las interfases remotas, agrega
nodos y //enlaces a la red. nodos y enlaces son tipo Vector
…
red.agrega(nodos,enlaces);
//Crea otra red
IRedEspacial redmaxFlujo = h_red.create(“Red simple Max Flujo de
la 01”);
//Invoca el método FordFulkerson sobre la red original
retornando la red de //máximo flujo
…
redmaxFlujo = red.FordFulkerson();
…
} catch (NamingException e){
…
}
…
```


Con el código anterior se confirma la facilidad de desarrollar aplicaciones empleando componentes desarrollados en la arquitectura.

CONCLUSIONES

La arquitectura de software descrita en este trabajo, representa una ayuda valiosa para el proceso de diseño e implementación de software dirigido al dominio de redes de distribución. Ella ayuda a satisfacer requerimientos comunes en subdominios de las redes, a la vez venciendo los problemas de portabilidad entre plataformas. Esta arquitectura puede ser adaptada fácilmente por diseñadores y programadores que intenten resolver problemas en los dominios indicados sin comenzar desde cero.

El método empleado en su elaboración resulta de gran utilidad para el desarrollo de arquitecturas reutilizables y a la vez es útil emplear la arquitectura para el desarrollo de familias de aplicaciones.

Se pretende, como trabajo futuro, la evaluación de la arquitectura, su extensión y su implementación para proporcionar una base de componentes de software reutilizables, distribuidos y con facilidades para la visualización de los datos espaciales en mapas y bajo una arquitectura de servicios SOA (Service Oriented Architecture).

Agradecimientos

Los autores agradecen el financiamiento proporcionado por el Consejo Nacional de Investigaciones Científicas CONICIT (bajo el proyecto No. G-97000824).

REFERENCIAS BIBLIOGRÁFICAS

- Booch, G., Jacobson, I., and Rumbaugh, J. (1998). **The Unified Modeling Language User Guide.** Addison-Wesley, Reading, MA.
- Cormen, T., Leiserson, C., Rivest, R. y Stein, C. (2001). **Introduction to Algorithms,** Second Edition. MIT Press and McGraw-Hill,. ISBN 0-262-03293-7. Section 26.2: The Ford-Fulkerson method, pp.651–664.
- Gail, Anderson. Paul, Anderson. (2002). **Enterprise JavaBeans Component Architecture: Designing and Coding Enterprise Applications.** Prentice Hall PTR. ISBN 0-13-035571-2. March 11.
- Heineman, T. y Councill, W. (2001). **Component.Based Software Engineering: Putting the Pieces Together.** Addison-Wesley.
- Holibaught, (1993). **Object–Oriented Domain Analysis Method (JODA),** Joint Integrated Avionics Working Group, Technical Report No. CMU/SEI-98-SR-3.
- Krut,R. (1993). **Integrating 001 Tool Support into the Feature-Oriented Domain. Software Engineering Institute,** Carnegie Mellon University, Technical Report No. CMU/SEI-93-TR11.
- Montilva, J. (2000). **Un modelo de grafos espaciales para la gestión de redes de servicios en sistemas de información geográfica.** Rev. Téc. Ing. Univ. Zulia, Vol. 23, No. 2, Agosto, p.87 – 97.
- Montilva, J. y Ramos, Y. (2001). **Patrones de diseño para el modelado de redes en sistemas de información geográfica.** Revista Colombiana de Computación. Vol. 1, No. 1, Diciembre, p. 91-104.
- Predonzani, Paolo. Succi, Giancarlo. Breñaza, Tullio. (2000). **“Strategic Software Production with Domain Oriented”,** Artech House Computing Library.