Guía de notación

Seminario de Arquitectura de Software

Tabla de Contenido

§1. Introducción

Este documento ofrece una guía de notación para la descripción de la arquitectura de software de un sistema informático. Esta guía muestra cómo utilizar Unified Modeling Language (UML) para describir las vistas de la arquitectura propuestas en el framework arquitectónico 4+1 sugerido por Rational Unified Process (RUP).

Primera se describe la estructura de un documento de descripción de arquitectura de software, inspirado en el template propuesto por RUP. Luego, se presenta la guía de aplicación de la notación UML para especificar las vistas de la arquitectura, apoyada con ejemplos.

§2. Estructura del SAD

El Software Architecture Document (SAD) es el artefacto que provee la vista global de la arquitectura de un sistema de software en RUP. Es el medio de comunicación de la arquitectura a todos los interesados (stakeholders) en el proyecto y en el producto. El SAD como artefacto es un documento de texto y esta organizado en vistas, así como el modelo completo del sistema esta organizado en modelos desde distintas perspectivas. Así, el SAD incluye diagramas que ilustran parte de los modelos del sistema que son significativos para la arquitectura.

Aunque la estructura del documento depende del proceso, proyecto y producto en desarrollo, en general incluye las secciones descritas a continuación.

1. Introducción

Provee una reseña del documento completo, incluyendo las siguientes secciones:

1.1. Propósito

Describe el propósito del SAD como artefacto, en el contexto de toda la documentación del proyecto, y presenta brevemente su estructura. Identifica la audiencia esperada e indica cómo se espera que éstos lo utilicen.

1.2. Alcance

Brinda una breve descripción de a qué es que el SAD aplica, qué es afectado por él y qué lo influencia.

1.3. Definiciones, acrónimos y abreviaciones

Provee la definición de todos los términos, acrónimos y abreviaciones requeridos para interpretar correctamente el documento. Puede incluir simplemente una referencia al glosario del proyecto, si existe.

1.4. Referencias

Provee una lista completa de todos los documentos referenciados. Cada documento debe estar identificado por su título, fecha y la organización que lo publica, y deben proveerse las fuentes de dónde obtener tales referencias. Esta subsección puede ser una sección al final del documento.

1.5. Resumen

Explica la organización y el contenido del resto del documento.

2. Representación de la arquitectura

Describe cómo está representada la arquitectura del sistema. Presenta el framework arquitectónico y enumera las vistas definidas para representar la arquitectura. Indica qué tipo de vista (viewpoint) corresponde a cada una y las relaciones entre ellas.

3. Contexto y Alcance

Describe el contexto del sistema de software y su alcance.

Stakeholders

Describe las categorías de interesados en el sistema.

Objetivos y motivadores

Enumera los objetivos del sistema y aquellos elementos que motivaron su desarrollo. Provee una breve descripción del negocio en el que se inserta el sistema.

Contexto

Describe las principales áreas de funcionalidad, los principales subsistemas (si existen), los sistemas que serán reemplazados total o parcialmente y la información que será migrada al nuevo sistema.

Alcance

Describe qué responsabilidades están dentro del sistema y cuáles no lo están, ya sea porque no se dará soporte informático o porque el soporte lo ofrece otro sistema.

[4..n]. Vista i

Describe la arquitectura desde una perspectiva particular. Ver secciones §3 a §6.

§3. Vista de Casos de Uso, Escenarios y Atributos de Calidad; Vista '+1'

Esta vista presenta los requerimientos del sistema que tienen impacto en su arquitectura. Es la guía para la realización y el entendimiento de las restantes vistas. El contenido de esta vista puede organizarse como sigue.

3.1. Procesos de Negocio

Describe el/los proceso(s) de negocio en los cuales el sistema participa. Se enumeran los procesos brindando una sinopsis de los mismos, e indicando qué rol juega el sistema en ellos. Puede acompañarse la descripción por medio de una figura o mediante un diagrama de actividad de UML, Figura 1 y 2 respectivamente.

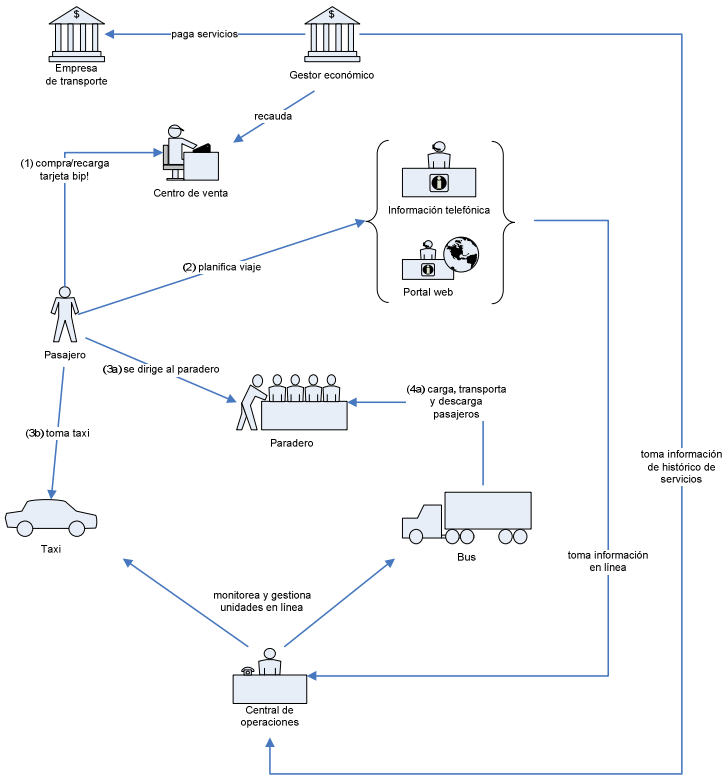

Figura 1. Ejemplo de figura de descripción de procesos de negocios (informal)

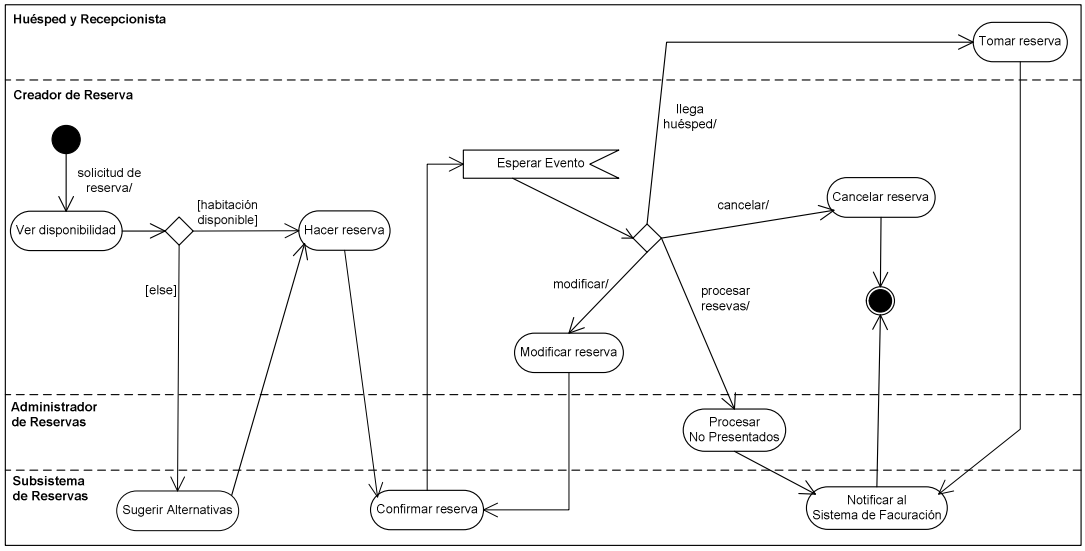

Figura 2. Ejemplo de diagrama de actividad de UML.

La Figura 2 presenta cuatro andariveles, cada cual agrupa las actividades que hace un determinado actor. El diagrama consiste de un grafo dirigido cuyos nodos corresponden al estado inicial y final, actividades realizadas por los actores, condicionales y espera de eventos, y cuyas aristas indican las posibles secuencias en que los nodos pueden ocurrir.

3.2. Contexto del Sistema

Describe en mayor detalle el contexto del sistema identificando los actores participantes y las principales áreas de funcionalidad sobre las que actúan.

Para cada actor se brinda una descripción de su rol (responsabilidades) y se indica si corresponden a actores humanos o a sistemas externos. Pueden identificarse actores abstractos y pueden relacionarse mediante especialización-generalización. Las áreas de funcionalidad se describen mediante un nombre y se indica los principales casos de uso o escenarios que involucra.

Esta subsección puede acompañarse de un diagrama de contexto del sistema; la Figura 3 presenta un ejemplo. En ella, se denota el contexto del sistema (recuadro), en el cual dentro se coloca el nombre y los paquetes correspondientes a las áreas de funcionalidad. Pueden presentarse dependencias entre los paquetes si las áreas de funcionalidad están vinculadas de alguna manera (extensión, inclusión, especialización). Fuera del recuadro (sistema) se ubica a los actores. Generalmente, se utiliza la notación canónica (rectangular) para representar a los actores consistentes en sistemas externos, y el estereotipo usual (muñequito) para denotar actores humanos. Si el sistema externo consiste en un repositorio de datos puede utilizarse un cilindro para denotarlo. Pueden incluirse además las relaciones entre los actores. Se utiliza una asociación entre los actores y los paquetes para denotar la participación de un actor (el uso) en un área funcional.

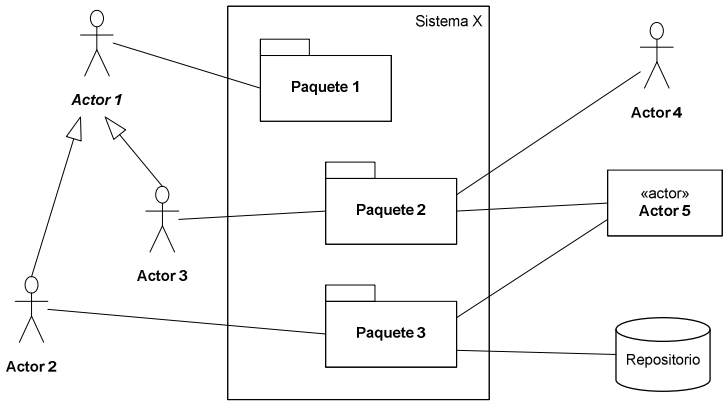

Figura 3. Ejemplo de diagrama de contexto del sistema.

3.3. Requerimientos funcionales

La descripción de los requerimientos funcionales es textual, organizada generalmente por área de funcionalidad (paquete del contexto del sistema).

Una forma de mostrar estos requerimientos es mediante casos de uso. Se incluye la versión resumida de cada caso de uso significativo. Esta versión incluye el nombre, los actores involucrados indicando el actor que inicia (dispara) el caso de uso en subrayado, y la sinopsis del mismo. Para aquellos casos de uso cuyos escenarios son de importancia para la arquitectura, puede utilizarle la versión expandida de los casos de uso. Esta versión incluye además el curso típico de eventos y las extensiones. La Figura 4 presenta un ejemplo de caso de uso expandido. Las acciones se numeran, y cada una tiene la forma 'X hace Y' siendo X un actor o el sistema, y Y la tarea realizada. Puede incluirse otro caso de uso utilizando la forma 'Incluir Z' siendo Z otro caso de uso.

La sección de extensiones puede incluir un subconjunto de todas las extensiones del caso de uso. El escenario correspondiente al curso típico de eventos suele incluirse siempre. Además, acompañando la descripción del caso de uso, puede indicarse los atributos de calidad asociados (y específicos) al caso de uso.

Nombre	Paquete 1 :: CasoDeUso 1
Actores	Actor 4
Sinopsis	El CasoDeUso 1 comienza cuando Actor 4 solicita tal tarea. El sistema realiza tal y tal actividad.
Curso Típico de Eventos	
	1. Actor 4 solicita tarea. 2. Sistema confirma disponibilidad. 3. Incluir CasoDeUso 2. 4. Actor 4 hace tal cosa. 5. Sistema hace tal otra.
Extensiones	
	2a. No hay disponibilidad: 1. Sistema deja registro de la ausencia de disponibilidad. 2. Fallo. 2b. No hay disponibilidad pero sí en otro lugar: 1. Sistema ofrece alternativas. 2. Actor 4 elije alternativa. 3. Resumir en 3.

Figura 4. Ejemplo de caso de uso expandido.

Junto a la descripción textual de los casos de uso puede incluirse un diagrama de casos de uso en el que se muestre sus relaciones con otros elementos del modelo. En el ejemplo de la Figura 5 Actor 1 utiliza entre 40 y 50 veces por día CasoDeUso1. Este caso de uso es extendido por CasoDeUso2 cuando Actor 1 requiere la función X. CasoDeUso2 incluye CasoDeUso3 (abstracto), siendo CasoDeUso4 una especialización concreta de este. A su vez, CasoDeUso2 se comunica con el sistema externo Actor 2.

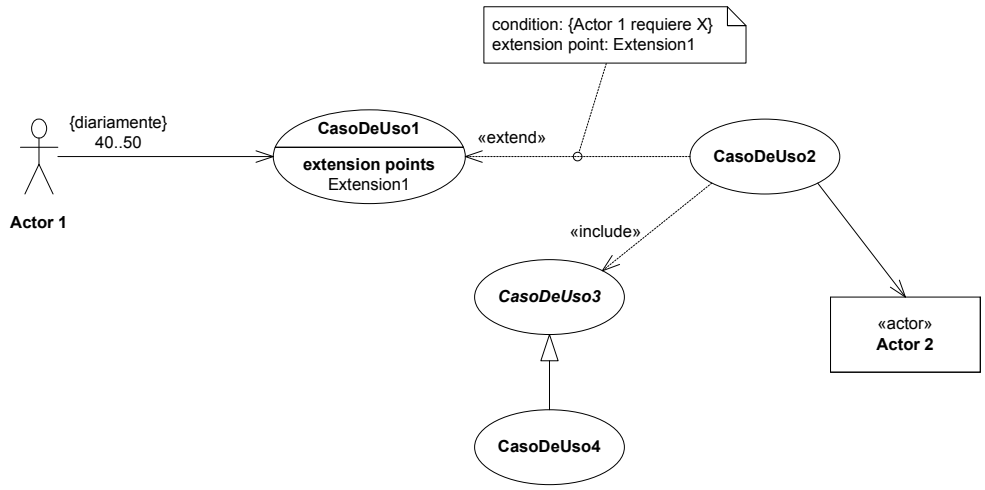

Figura 5. Ejemplo de diagrama de casos de uso.

Un enfoque alternativo al uso de casos de uso para la descripción de los requerimientos funcionales es el uso de escenarios funcionales. Éstos consisten en descripciones textuales de escenarios de uso del sistema, estructuradas de la siguiente manera:

Sinopsis: Breve descripción de lo que el escenario pretende ilustrar.

Estado del sistema: Estado del sistema antes de que ocurra el escenario. Usualmente, consiste en la descripción de la información significativa que ya está almacenada en el sistema para que el escenario tenga sentido.

Entorno del sistema: Cualquier observación significativa sobre el entorno en el cual el sistema está ejecutando, tales como la disponibilidad de sistemas externos, comportamiento particular de la infraestructura, restricciones temporales, etc.

Estímulo externo: Una definición de qué causa que ocurra el escenario, tal como la llegada de datos a una interfaz, entrada del usuario, el paso de cierto tiempo, o cualquier otro evento significativo.

Respuesta requerida: Explicación, desde la perspectiva de un observador externo, de cómo el sistema debería responder a este escenario, es decir, qué tareas debe hacer y qué se espera que entregue como salida.

La Figura 6 presenta un ejemplo de escenario funcional.

Sinopsis: Edición de archivos fuente.

Estado del sistema: El proyecto de desarrollo ha sido creado y los archivos están almacenados en la carpeta del proyecto.

Entorno del sistema: Hay un editor de texto disponible.

Estímulo externo: El usuario abre un archivo fuente, lo edita y lo almacena.

Respuesta requerida: El sistema almacena la nueva versión del archivo fuente, chequea que el contenido siga las reglas de buena formación del lenguaje de programación. Si está mal formado, notifica al usuario. Si está bien formado, construye/actualiza las estructuras auxiliares (abstractas) correspondientes a la representación lógica del contenido del archivo.

Figura 6. Ejemplo de escenario funcional de un IDE.

3.4. Atributos de calidad

Los atributos de calidad del sistema se documentan mediante escenarios de calidad. Estos consisten en descripciones textuales de escenarios en que ocurre un cambio en el entorno del sistema, estructuradas de la siguiente manera:

Sinopsis: Breve descripción de lo que el escenario pretende ilustrar.

Entorno del sistema: Cualquier observación significativa sobre el entorno en el cual el sistema está ejecutando, tales como la disponibilidad de sistemas externos, comportamiento particular de la infraestructura, restricciones temporales, etc. En algunos casos, puede capturar hechos sobre el estado del sistema si el comportamiento especificado del escenario recae sobre el estado.

Cambios en el entorno: Una explicación de qué ha cambiado en el entorno del sistema que causa la ocurrencia del escenario. Puede ser cambios en la infraestructura o fallas, cambios en el comportamiento interno del sistema, ataques de seguridad, modificaciones requeridas, o cualquier otro cambio en el entorno que requieren que el sistema posea una propiedad particular para manejarla.

Comportamiento esperado: Una definición de cómo el sistema debe comportarse en respuesta al cambio en el entorno.

La Figura 7 presenta un ejemplo de escenario funcional.

Sinopsis: Cómo el sistema se comporta cuando ocurre una falla al escribir los resúmenes estadísticos en la base de datos.

Entorno del sistema: El entorno esta trabajando correctamente.

Cambios en el entorno: Al escribir los resúmenes estadísticos en la base de datos, el sistema recibe una excepción indicando que la escritura fallo (e.g. la base de datos está llena).

Comportamiento esperado: El sistema debe detener el proceso de las estadísticas inmediatamente y suspender cualquier trabajo en curso. El sistema debe agregar al log un mensaje de error fatal en la consola de monitoreo operacional y debe terminarse.

Figura 7. Ejemplo de escenario de calidad de un sistema de información.

§4. Vista Lógica

La Vista Lógica consiste en la descripción lógica de la estructura y el comportamiento del sistema; la estructura se describe mediante el tipo de vista de *Módulos* mientras que el comportamiento se describe mediante el tipo de vista de *Componentes y Conectores*. Así, la sección para la Vista Lógica del SAD se organiza en dos partes, cada una atacando uno de estos dos aspectos.

4.1. Módulos

La estructura lógica del sistema se describe mediante diferentes niveles de refinamiento. El nivel superior corresponde al sistema en forma completa y usualmente se omite. El nivel inferior corresponde a los elementos de las construcciones lógicas básicas de un sistema; e.g. en un sistema orientado a objetos este último nivel consiste de clases e interfaces y sus relaciones. La Vista Lógica no necesariamente corresponde exclusivamente a los primeros niveles de refinamiento. Los presenta, pero puede incluir además aquellos elementos de los niveles de refinamiento más bajos que se utilizan para la realización de los casos de uso de la Vista de Casos de Uso.

Para especificar la Vista Lógica según este enfoque, primero deben presentarse los distintos niveles de refinamiento, indicando el cometido de cada uno de ellos. Cada nivel refina al nivel inmediatamente anterior en el sentido de que su organización interna está más refinada (incluye mayor detalle) que la organización interna del nivel anterior. Generalmente, cada elemento de un nivel es refinado por una organización de elementos en el nivel siguiente. Esta organización mediante refinamientos puede entenderse como la aplicación del estilo *Descomposición*.

La Figura 8 presenta un ejemplo de cuatro niveles de refinamiento. La Figura 9 presenta un ejemplo de los elementos componentes de cada nivel de refinamiento aplicando directamente el estilo *Descomposición*. Para un sistema grande, presentar el árbol completo en un diagrama no es sencillo, por lo que organizar la Vista Lógica utilizando una sección por cada nivel de refinamiento, y una subsección por cada elemento a ser refinado, resulta más legible.

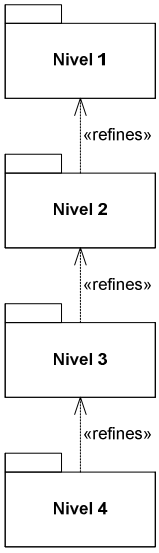

Figura 8. Esquema de niveles de refinamiento.

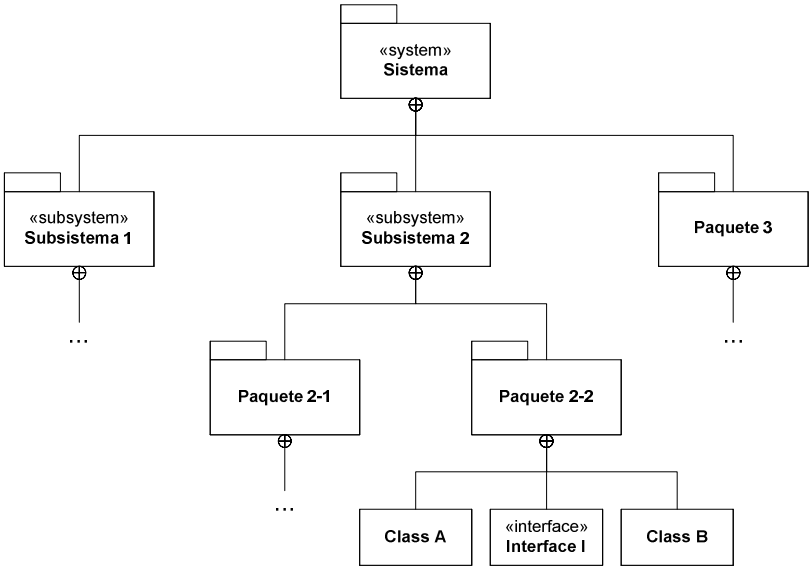

Figura 9. Árbol de descomposición.

4.1.1. Organización interna de los módulos

Para presentar la organización interna (la descomposición de un elemento en una organización de elementos) se utiliza un diagrama de estructura estática de UML. El diagrama se titula con el tipo y nombre del elemento cuyo estructura interna es presentada, y dentro incluye los elementos que lo componen. Asimismo, se presentan las relaciones de Uso y Generalización correspondientes. La Figura 10 presenta un ejemplo de cómo se presenta la organización del elemento *Subsistema 2* de la Figura 9.

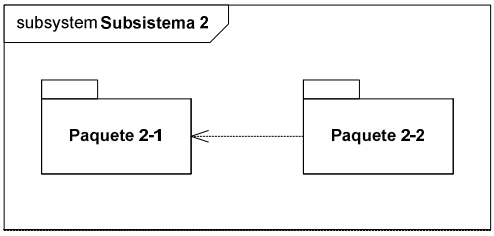

Figura 10. Ejemplo de diagrama de estructura estática para el módulo *Subsistema 2*.

La Figura 11 presenta la organización interna del módulo *Paquete 2-2*. Notar que en la Figura 10 se indicó una relación de uso desde *Paquete 2-2* a *Paquete 2-1*. Esta relación indica que al menos un elemento de *Paquete 2-2* usa al menos un elemento de *Paquete 2-1*. Por dicho motivo, en la Figura 11 se incluyen los elementos de los otros módulos de los cuales dependen los elementos del módulo siendo descrito. A aquellos elementos externos se les indica de qué módulo provienen y puede utilizarse otro color para denotarlos. Las relaciones de los elementos externos no se presentan en un diagrama, solo aquellas relativas a los elementos internos.

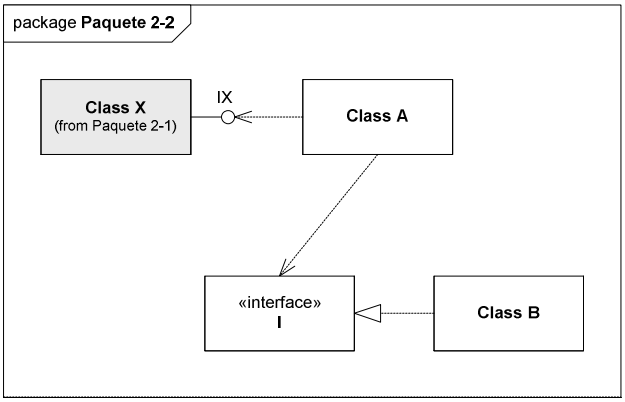

Figura 11. Ejemplo de diagrama de estructura estática para el módulo *Paquete 2-2*.

Cuando se presenta la organización interna de un módulo, debe incluirse un catálogo de los módulos internos, indicando su nombre y su cometido en la arquitectura.

4.2. Componentes

El comportamiento lógico del sistema se describe en términos de componentes interconectados que realizan la funcionalidad esperada del sistema. Asimismo, la organización de estos componentes determina las propiedades de calidad que presentará el sistema. Para construir esta parte de la Vista Lógica se utiliza el viewtype Componentes y Conectores aplicando el/los estilo(s) necesarios para satisfacer los requerimientos del sistema.

Esta sección del SAD se organiza en subsecciones, cada una atacando un aspecto significativo del sistema; a cada subsección se le llama view-packet. Un view-packet consiste en la aplicación de uno o más estilos de C&C (generalmente uno), en que se resuelve un aspecto significativo del sistema. Notar que un mismo componente puede aparecer (y generalmente lo hace) en más de un view-packet. Cada view-packet esta conformado por las partes listadas a continuación.

Primero, debe incluir un diagrama de componentes que muestre la organización estática de los componentes y sus conecciones. La Figura 12 presenta un ejemplo de dos componentes según el estilo Cliente-Servidor. La Figura 13 muestra un ejemplo de aplicación del estilo Pipes&Filter. A un componente puede adjuntársele un conjunto de puertos y cada puerto puede tener una o más interfaces o sockets; una interfaz refiere a servicios provistos y un socket a servicios requeridos. Los conectores pueden denotarse por la conexión de un socket a una interfaz (ver Figura 12), o a la conexión de un componente a una interfaz, o la conexión de dos componentes (ver Figura 13). Pueden usarse estereotipos para indicar el tipo de conector.

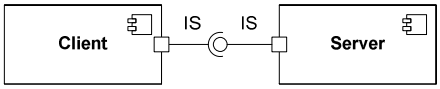

Figura 12. Ejemplo de diagrama de componentes aplicando Client-Server.

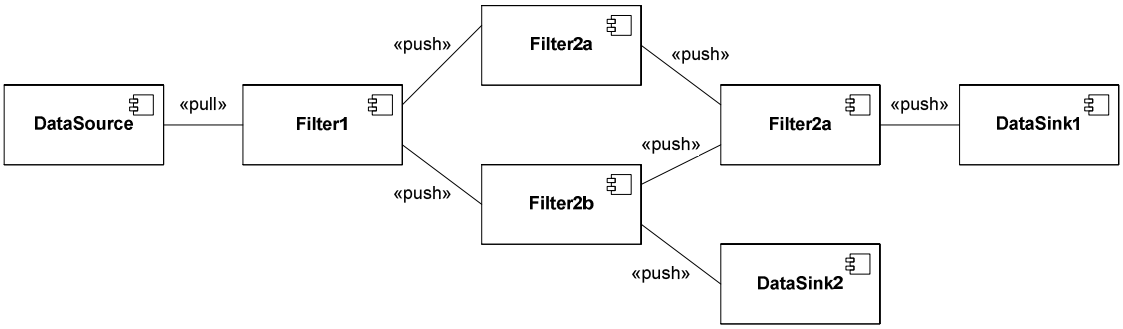

Figura 13. Ejemplo de diagrama de componentes aplicando Pipes&Filtres.

Segundo, un diagrama de secuencia en que se presente la interacción entre los componentes; la Figura 14 presenta un ejemplo. Si la interacción es conocida (dado que el patrón en que se basa el view-packet fue aplicado sin modificaciones) este diagrama puede omitirse. Es importante notar que esta interacción se muestra a alto nivel y no debe abusar en detalles.

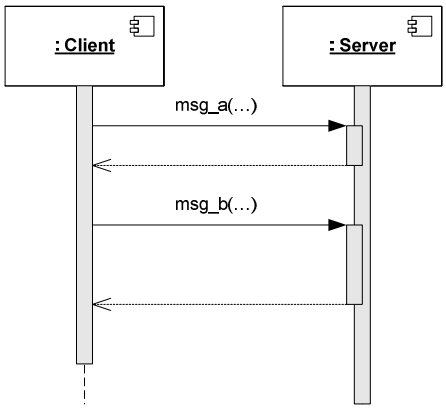

Figura 14. Ejemplo de diagrama de secuencia presentando la interacción entre componentes.

Tercero, el view-packet debe incluir un catálogo de los elementos incluidos en él, entendiendo por elementos tanto a los componentes como a los conectores. Si no hay información relevante respecto a un conector particular, el mismo puede no llevar nombre en los diagramas y omitirse del catálogo. Cuarto, se debe indicar el rationale del view-packet, es decir, los fundamentos que llevaron a tomar la decisión de organizar el view-packet tal como se está presentando. Por último, el view-packet debe describir cómo se relaciona con otros view-packets.

Refinamiento

Existe una notación especial cuando un view-packet refiere a la organización interna de un componente (probablemente ya mostrado en otro view-packet). Esto sucede generalmente cuando los view-packet presentan la organización de elementos en distintos niveles de refinamiento. En estos casos, la organización estructural del componente se presenta dentro del propio componente, indicando cómo los servicios ofrecidos y consumidos están vinculados con los servicios de los componentes internos. La Figura 15 presenta un ejemplo.

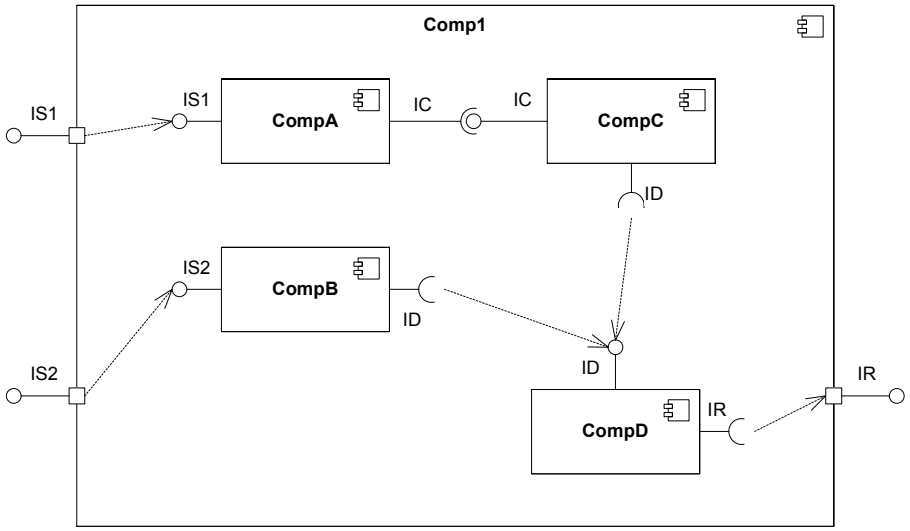

Figura 15. Ejemplo de composición interna de un componente.

§5. Vista de Procesos

La Vista de Procesos del documento de descripción de arquitectura presenta la estructura completa del sistema en términos de procesos y aquellos threads más relevantes desde el punto de vista de la arquitectura. Esta vista se organiza de la siguiente manera.

Primero, la vista indica mediante un diagrama de clases, los distintos procesos y threads que componen al sistema en tiempo de ejecución. Procesos y threads se denotan mediante clases, utilizando los estereotipos *«process»* y *«thread»* respectivamente. La Figura 16 presenta un ejemplo. En la figura, Proceso1 tiene dos threads, Thread11 y Thread12, creados según la dependencia con estereotipo *«create»*. A su vez, tanto Proceso1 como Thread12 realizan comunicación (inter-process communication) con Proceso2.

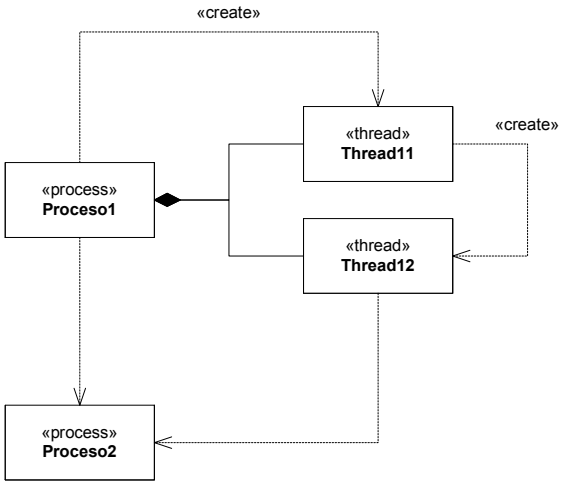

Figura 16. Ejemplo de diagrama presentando procesos y threads.

Segundo, la vista debe incluir un catálogo de procesos y threads, indicando cuál es su objetivo en el sistema. A su vez, se debe indicar qué componentes de la Vista Lógica se espera que corran en cada proceso.

§6. Vista de Implementación

La Vista de Implementación describe la organización estática de las unidades de implementación y de deployment del sistema. Las unidades de implementación consisten en el conjunto de elementos fuente del sistema mientras que las unidades de deployment corresponden a los elementos que serán instalados en los nodos de procesos. El Modelo de Implementación del sistema generalmente incluye ambos aspectos, pero desde el punto de vista de la arquitectura, el primero generalmente no es relevante. El segundo aspecto, en cambio, si es relevante.

Para presentar las unidades de deployment del sistema de utiliza una vista basada en el tipo de vista *Módulos*, siendo los elementos las unidades de deployment (archivos exe, dll, jar, war, ear, zip, wav, jpg, html, jsp, asp, etc.) y las dependencias entre ellos la relación de uso entre los módulos. Si la cantidad de unidades de deployment del sistema es grande pueden utilizarse paquetes para organizar la presentación. Estos paquetes no corresponden a packages o namespaces (ya que éstos son aspectos lógicos) y tampoco a subdirectorios en el sistema de archivos (aunque puede decidirse que sí corresponda para este último caso).

La Figura 12 presenta un ejemplo de diagrama de estructura estática de UML para la presentación de la organización de las unidades de deployment del sistema. Nótese que los módulos llevan el estereotipo correspondiente a *Artefacto* para indicar que son unidades de deployment.

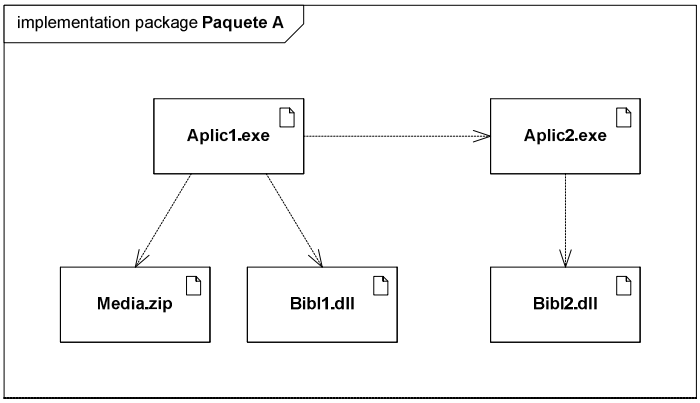

Figura 12. Ejemplo de diagrama de estructura estática de UML presentando las unidades de deployment.

Al igual que en la Vista Lógica, es importante documentar el catálogo de elementos que componen la vista, indicando su nombre y cometido, así como el conjunto de módulos de la Vista Lógica que incluye cada unidad de deployment.

§7. Vista de Deployment

Esta vista presenta una o más configuraciones físicas sobre las cuales se pondrá en producción el sistema. Comúnmente abarca la parte de la infraestructura informática de la organización que adoptará el sistema.

Una configuración puede mostrarse en dos niveles diferentes: de tipos y de instancias. En el nivel de tipos se muestran los tipos de nodos de proceso y dispositivos existentes, así como sus conexiones; la Figura 8 presenta un ejemplo. En el nivel de instancias se muestran los nodos de proceso y dispositivos existentes y sus conexiones; la Figura 9 muestra un ejemplo. Este nivel muestra la organización real de la infraestructura tecnológica. El nivel de instancias debe corresponderse con el nivel de tipos, es decir, cada instancia mostrada debe corresponder a un tipo del otro nivel. Dado que el nivel de instancias en general esta muy poblado y el nivel de tipos permite comprender cabalmente la infraestructura, así como no limita la infraestructura a una instancia particular, el nivel de instancias suele omitirse.

En la vista de deployment suele incluirse información sobre la configuración de cada nodo o dipositivo. Asimismo, suele indicarse qué unidades de deployment (de la vista de implementación) serán instaladas en cada uno. Además, suele indicarse además qué procesos (de la vista de procesos) se espera que corra en cada uno de los nodos. Las Figuras 8 y 9 presentan un diagrama de deployment en el que se indica las unidades de deployment a instalar en los nodos.

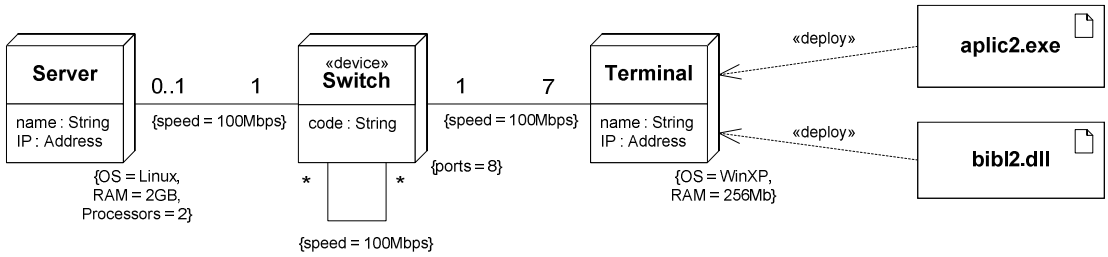

Figura 8. Ejemplo de diagrama de deployment a nivel de tipos.

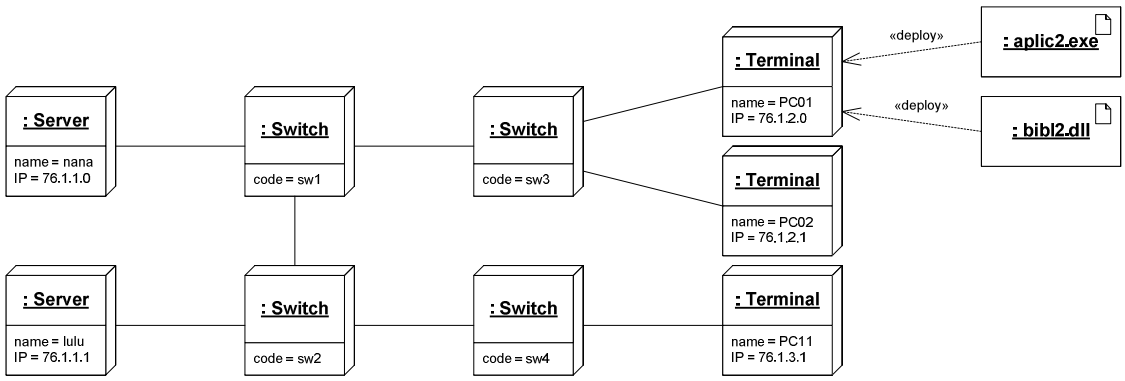

Figura 9. Ejemplo de diagrama de deployment a nivel de instancias.