LA ARQUITECTURA SOFTWARE. EL MODELO 4+1

Algunas notas breves sobre la arquitectura software y su modelización en 4+1...

La arquitectura software trata el diseño e implementación de la estructura de alto nivel del software. Es el resultado de ensamblar un cierto número de elementos arquitectónicos para satisfacer la funcionalidad y ejecución de los requisitos del sistema; así como los requisitos no funcionales del mismo: fiabilidad, escalabilidad, portabilidad, disponibilidad, etc. Perry y Wolf (1992) describen una arquitectura software como:

Arquitectura Software = {Elementos, Formas, Fundamento/Restricciones}

Es muy complejo capturar la arquitectura software en un sólo modelo (o diagrama). Para manejar esta complejidad se representan diferentes aspectos y características de la arquitectura en múltiples vistas. Una vista es "una presentación de un modelo, la cual es una descripción completa de un sistema desde una particular perspectiva" (Kruchten, 1995). El modelo más aceptado a la hora de establecer las vistas necesarias para describir una arquitectura software es el modelo 4+1 (Kruchten, 1995).

Este modelo define 4 vistas principlaes:

- Vista Lógica (Logical View), modelo de objetos, clases, entidad – relación, etc.
- Vista de Proceso (Process View), modelo de concurrencia y sincronización.
- Vista de Desarrollo (Development View), organización estática del software en su entorno de desarrollo (librerías, componentes, .ear, .jar, etc.).
- Vista Física (Physical View), modelo de correspondencia software - hardware (aspectos de distribución en máquinas, por ejemplo)

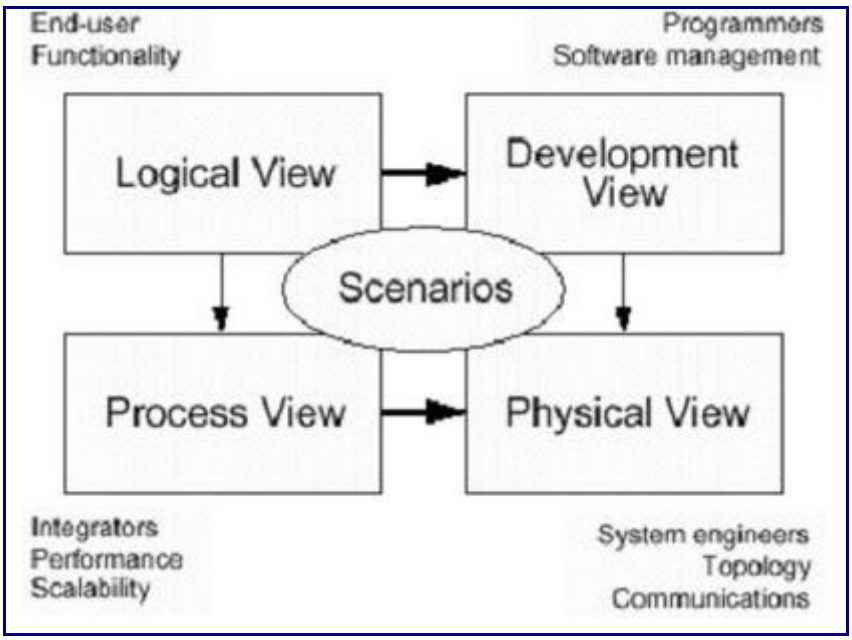

Figura 1. Modelo de vistas 4+1

Y una vista más, la "+1", que se muestra y traza en cada una de las anteriores y que está formada por las necesidades funcionales que cubre el sistema, y que en ocasiones identificamos como vista de "casos de uso". De donde deducimos que según este modelo, la arquitectura es en realidad evolucionada desde escenarios

El modleo 4+1 aplica la ecuación de Perry y Wolf (1992) de forma independiente para cada vista, por ejemplo, cada vista puede definir un conjunto de elementos para su uso (componentes, contenedores y conectores).

Cada vista es descrita usando su particular y más adecuada notación (por ejemplo, existen diagramas UML que se adapatan más a una vista que otra). Para cada vista los arquitectos pueden escoger cierto esquilo arquitectónico (patrón arquitectónico), permitiendo la coexistencia de múltiples estilos en un sistema.

Y por último, también comentar que el modelo de vistas “4+1” es “genérico”: otras notaciones y herramientas a parte de UML pueden usarse, y cualquier método de diseño, especialmente para las descomposiciones lógicas y de proceso.

1. Arquitectura Lógica (Logical Architecture)

Soporta el análisis y la especificación de los requisitos funcionales: lo que el sistema debería proporcionar en términos de servicios a sus usuarios. El sistema se descompone en un conjunto de abstracciones clave tomadas mayormente del dominio del problema, en forma de objetos o clases. En esta vista se usan comúnmente los diagramas de clases, los de interacción y objetos.

- Notación: La notación más usada es UML, y dentro de esta diagramas de clases y paquetes.
- Estilo: El estilo más usado para la vista lógica es el Orientado a Objetos.

2. Arquitectura de Procesos (Process Architecture)

Se tratan algunos requisitos no funcionales. Ejecución, disponibilidad, tolerancia a fallos, integridad, etc. Esta vista también especifica que hilo de control ejecuta cada operación identificada en cada clase identificada en la vista lógica. La vista se centra por tanto en la concurrencia y distribución de procesos.

- ❒ Notación: La notación más usada es UML, y dentro de esta diagramas estados, actividad y similares.
- ❒ Estilo: pueden encajar varios estilos. Por ejemplo, tomando la taxonomía de Garlan y Shaw (1993), pueden usarse tuberías y filtros (pipes and filtres) o Cliente – Servidor (con variantes de múltiples clientes – simple servidor y múltiples clientes – múltiples servidores). Para sistemas más complejos puede usarse un estilo similar a la ISIS system's process groups, como describe Kenneth Birman (1994) .

3. Arquitectura de Desarrollo (Development Architecture)

La vista de desarrollo o despliegue se enfoca en la organización de los módulos software en el entorno de desarrollo. El software es empaquetado en pequeños trozos (librerías de programa, subsistemas, componentes, etc.), los subsistemas se organizan en capas jerárquicas, y cada capa proporciona una interfaz bien definida a sus capas superiores

La vista de desarrollo toma por tanto requisitos internos relacionados con facilidad de desarrollo, gestión del software (reparto de requisitos, trabajo del equipo), evaluación de costes, planificación, monitorización del progreso del proyecto, reutilización, portabilidad, seguridad y restricciones impuestas por las herramientas o por el lenguaje de programación

Esta organización del software se suele apoyar en diagramas de módulos o de subsistemas que muestran las relaciones de exportación (export) e importación (import).

Y destacar que podrá describirse la vista de desarrollo por completo solamente después de haber identificado todos los elementos software.

- Notación: La notación más usada es UML, y dentro de esta diagramas de componentes y paquetes.
- Estilo: se recomienda definir de cuatro a seis capas de subsistemas en la vista de desarrollo. Una regla de diseño es que un subsistema puede solamente depender de subsistemas en la misma capa o en las menores. Esto minimiza las dependencias entre módulos a favor de una más simple estrategia capa – capa.

4. Arquitectura Física (Physical Architecture)

La vista física se centra en los requisitos no funcionales, tales como la disponibilidad del sistema, la fiabilidad (tolerancia a fallos), ejecución y escalabilidad. Y también presenta cómo los procesos, objetos, etc., corresponden a nodos de proceso:

- Componentes: nodos de proceso.
- Conectores: LAN, WAN, bus, etc.
- Contenedores: subsistemas físico

Varias configuraciones físicas pueden usarse. La correspondencia de el software a los nodos debe ser altamente flexible y tener el mínimo impacto en el código fuente.

5. Escenarios (Scenarios)

La vista de escenarios corresponde con instancias de casos de uso que unifican todas las vistas. Así, desde casos de uso se debiera poder hacer una trazabilidad a todos los componentes del sis tema software, viendo, por ejemplo, que máquinas, o clases, o componentes, o .jar, o procesos, son los responsables de que el sistema cubra una cierta funcionalidad.

6. Relación entre las vistas

Como sucede en otras muchas ocasiones, si bien el modelo no es una metodología si que "sugiere" un método de trabajo. Parece lógico que la vista de escenarios o casos de uso sea la de arranque, y que de ahí se pase a la vista lógica. Desde la vista lógica afrontaremos la de desarrollo y procesos, una vez que tenemos por ejemplo las clases definiremos maneras de agruparlas y modos de ejecución. Para con todo concluir en la vista física. Todo ello, obviamente, con sus correspondientes y típicas iteraciones.

7. Arquitectura y UML

Se ha ido exponiendo a lo largo de la explicación de cada una de las vistas su translación a un lenguaje de modelado concreto como UML. Hya que tener en cuenta que UML nace casi a la vez que el modelo 4+1, por lo que en un origen no existía una clara relación entre ambos, lo que amenudo produce confusión al diseñador que en la actualidad quiere modelar una arquitectura con ambas herramientas. A modo de resumen la translación se presenta en la siguiente tabla:

Vista	UML
Escenarios	Casos de Uso
Lógica	Clases, de Estados y Colaboración
Desarrollo	Componentes
Física	Despliegue
Procesos	Actividad, Estados, Secuencia

8. Referencias

D. Garlan and M. Shaw, "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering, Vol. 1, World Scientific Publishing Co., Singapore, 1993.

Kruchten P. Architectural Blueprints—The “4+1” View Model of Software Architecture. IEEE Software, November 1995, 12 (6), pp.42-50.

Perry D. E., Wolf A. L., “Foundations for the Study of Software Architecture,” ACM Software Engineering Notes, 17, 4, October 1992, 40-52.

K.P. Birman and R. Van Renesse, Reliable Distributed Computing with the Isis Toolkit, IEEE CS Press, Los Alamitos, Calif. 1994.
