Definición de Arquitectura para una Línea de Producto de Software

Alberto Romo Moreno.

Dra. Perla Velasco.

Control De Versiones

Nombre del Archivo	Versión	Fecha	Autor	Comentarios
ReporteTecnico_Definicion_de_arquitectura_de_una_SPL_v2.doc	1	28/07/09	ARM	Presentación final

Índice

Índice de Fíguras

Centro de Investigación en Matemáticas CIMAT,

Resumen

Este reporte técnico trata el estado del arte en la definición de Arquitecturas en las Lineas de producto de software, por lo que en el se tratan los diversos conceptos de Arquitectura de software encontrados en la literatura revisada; para comprender el concepto de arquitectura de software, debemos entender los conceptos relacionados con él, por lo tanto tambien encontramos los conceptos de vista, estilo y patrón arquitectónico, así como los Lenguages de Definición de Arquitectura (ADL por sus siglas en inglés), y modelo de arquitectura.

Así también se parte del concepto de Línea de producto de software (SPL por sus siglas en inglés) y los concepos relacionados con una arquitectura de lineas de productos de sofware como son: Core asset, commonalities, variability y dominio, los cuales deberán ser plasmados en la arquitectura de la línea de productos de software. Para establecer una línea de producto de software se deben realizar ciertas actividades, las cuales son organizadas por el Software Engeneering Institute en áreas de práctica; en este documento podrá encontrar una breve descripción de las actividades y áreas de práctica relacionadas con las líneas de producto de Software.

Una vez entendidos los dos conceptos que involucra Arquitectura de una línea de productos de software se plasman aspectos relacionados con la definición de la arquitectura para la SPL, es ahí donde se describe porque los conceptos de SPL deben ser incrustados en la arquitectura, para la definición de la arquitectura es necesario conocer una forma de representar la arquitectura (ADL), por lo que se listan algunos de las notaciones consideradas como lenguajes ADL's y su origen.

Palabras Clave

Arquitectura, Lenguaje de Definición de Arquitectura, ADL's, Línea de Producto de Software, SPL.

Dedicatoria.

A mi familia, la cual es la fuerza que me impulsa en las tareas que valen la pena.

Agradecimientos

A los investigadores del grupo de ingeniería de software del CIMAT, por la guía y apoyo que me ofrecieron para hacer este reporte técnico.

Capítulo 1. Arquitecturas de Software.

1.1 Concepto de Arquitectura de Software.

En la literatura revisada se puede encontrar diversas definiciones de arquitectura de software:

En el artículo “who needs an architect?” Martin Fowler presenta varias definiciones de arquitectura, entre las que se encuentran: “una palabra que usamos cuando queremos hablar de diseño pero queremos hacer que suene importante”. También presenta la siguiente frase acerca de arquitectura “en los proyectos de software más exitosos, los desarrolladores expertos que trabajaron en ellos han compartido el entendimiento del diseño del sistema. Ese entendimiento compartido es llamado arquitectura. Ese entendimiento incluye el cómo el sistema se dividió en componentes, y cómo los componentes interactán a través de las interfaces. Esos componentes están frecuentemente compuestos de componentes más pequeños, pero la arquitectura solo incluye los componentes e interfaces que son entendidos por todos los desarrolladores”

El SEI (Software Engineering Institute) en su Framework for software Product Line Practice [Northrop 2007] cita la siguiente definición dada por Bass, Clements y Kazman en su libro *Software Architecture in Practice, Second Edition*, en el año 2003,

> *“La arquitectura de un programa o sistema computacional es la estructura o estructuras del sistema, el cual comprende elementos de software, las propiedades externamente visibles de esos elementos, y las relaciones entre ellos. Las propiedades “externamente visibles” son esas supociciones que otros elementos pueden hacer de un elemento, como lo son los servicios provistos, características de desempeño, la captura de faltas, uso de recursos compartidos, etc.”*

Esta definición también aparece citada en el capítulo IX del libro “Design Software Intensive System” de [Tyako 2009]. Además aparecen las siguientes:

> *“Es una de las actividades del diseño de software la cual incluye los requerimientos funcionales y no funcionales del sistema”.*
>
> *“La organización fundamental de un sistema representado por sus **componentes**, sus **relaciones** entre ellos y el **ambiente**, y los principios que gobiernan su diseño y evolución”[IEEE 1471].*

Además [Tyako 2009] señala que el concepto de arquitecturas de software está más allá de los algoritmos y estructuras de datos y se centra en diseño y especificación del conjunto organizacional del sistema y la estructura del control global; protocolos para la comunicación, sincronización y acceso a datos; tareas

de funcionalidad para diseñar elementos; distribución física; composición de elementos de diseño; escalabilidad y desempeño y **selección entre altenativas de diseño.**

Para Medvidovic en [Medvidovic 2006] Una arquitectura de sistema de software es el conjunto de las decisiones de diseño principales acerca del sistema.

- Decisiones de diseño relacionadas con la **estructura** del sistema, por ejemplo: el sistema estará compuesto por los sguientes componentes: la lógica del negocio, la interfaz de usuario y el almacenamiento de datos.

- Decisiones de diseño relacionadas con el **comportamiento** del sistema (también llamadas funcionales), por ejemplo el almacenamiento, visualización y procesamiento de datos deberá ser tratado separadamente.

- Decisiones de diseño relacionadas con la **interacción** por ejemplo, la comunicación entre los elementos del sistema se hará mediante notificación de eventos.

- Decisiones de diseño relacionadas con el **desarrollo** del sistema mismo por ejemplo el proceso que se seguirá para desarrollar y evolucionar el sistema.

- Decisiones de diseño relacionadas con el **bussines position** del sistema por ejemplo, su relación con otros productos y time to market

Las definiciones señaladas anteriormente tienen en común que todas hablan de la descomposición del sistema en sus componentes, (excepto la que se mencionó primero de Martin Fowler), la de [Northrop 2007] coincide con [IEEE 1471] y con [Medvidovic 2006] en que la interacción de los componentes forman parte de la arquitectura.

Pero Medvidovic va más allá al detallar que la arquitectura contiene las desiciones relacionadas con el desarrollo del sistema y con la posición del sistema en el negocio, esto determina lo que en la definición de la IEEE se menciona como los principios que gobiernan el diseño y la evolución del sistema.

1.2 Conceptos relacionados con arquitectura de software.

Las diferentes características de un sistema computacional se representa con diferentes vistas en la arquitectura de software. Existen diferentes conjuntos de vistas de los cuales entre los más populares se encuentran: el conjunto de vistas 4 + 1 de Krutchen, y recientemente una propuesta de tres vistas hecha por

Clements. A continuación se resume las 4 +1 vistas de Krutchen y las 3 vistas de Clements

1.2.1 4+1 vistas de Krutchen

1.2.1.1 Vista lógica.

Provee una descomposición del sistema en subsistemas y define sus interfaces y la interacción entre ellos.

1.2.1.2 Vista de procesos.

Define la concurrencia y mecanismos de sincronización entre las tareas en los nodos de hardware y sus interacciones.

1.2.1.3 Vista de deployment.

Define la organización estática del software en el ambiente de desarrollo.

1.2.1.4 vista de casos de uso.

Se definen escenarios (instancias de casos de uso) para ayudar a descubrir elementos arquitectónicos y validar la arquitectura.

1.2.1.5 La vista de implementación.

Representa los componentes usados para ensamblar y liberar el sistema físico. Se enfoca en la administración de la configuración y en liberaciones de software construido.

1.2.2 Vistas propuestas por Clements

Las vistas propuestas por Clements son: vistas de modulos, vista de componentes y conectores y vista de Localización.

1.2.2.1 Vista de modulos.

Es una vista estructural de la arquitectura comprende los modulos de código tal como clases, paquetes y subsistemas en el diseño, también captura la descomposición de modulos, herencia, asociaciones y agregaciones.

1.2.2.2 La vista de componentes y conectores

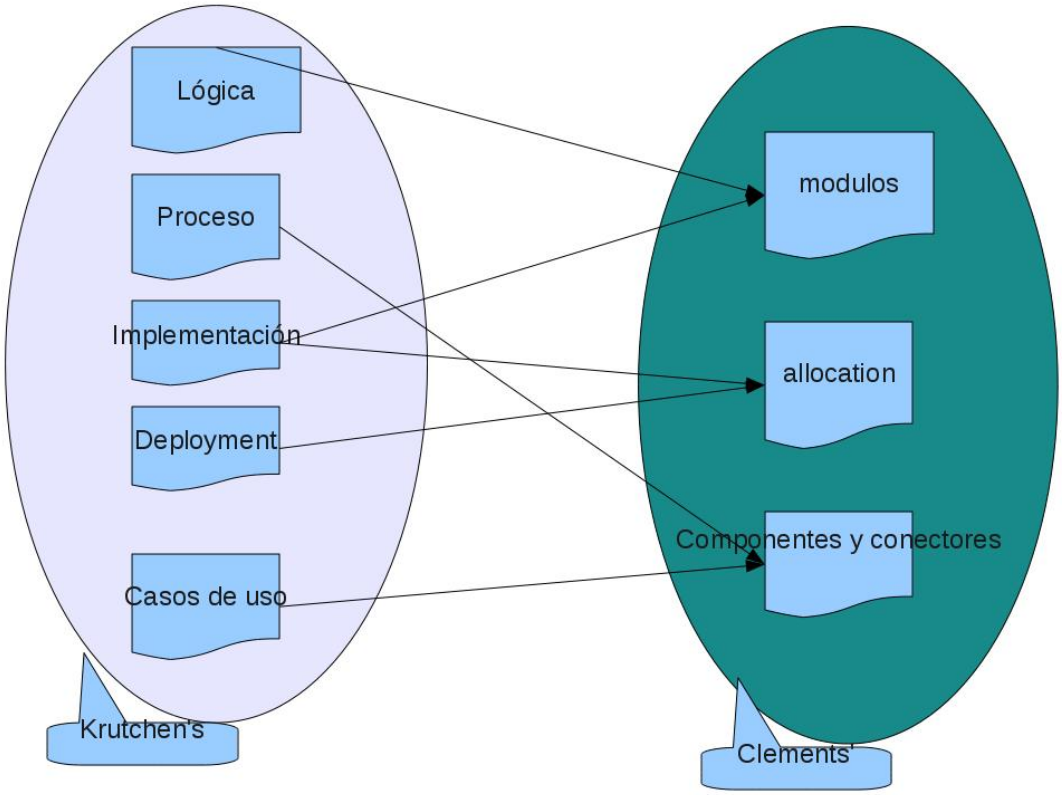
Describe los aspectos de comportamiento de la arquitectura. Los componentes son típicamente objetos, hilos o procesos, y los conectores describen como los componentes interactuan.

1.2.2.3 La vista de localización.

Muestra cómo los procesos en la arquitectura son mapeados en el hardware y cómo se comunican. También captura la vista del código fuente bajo

administración de la configuración y quién es el responsable para los módulos de código.

Se encuentra similitud entre los dos conjuntos de vista, la Fígura 1 se muestra el mapeo entre las dos versiones de conjuntos.



Fígura 1: Relación entre los conjuntos de vistas de Krutchens y Clements

1.2.3 Estilo Arquitectónico (architectural Style).

David Garlan en [Garlan 1995] define un estilo arquitectónico como un patrón recurrente de la organización del sistema que provee un framework abstracto para una familia de aplicaciones. Los estilos arquitectónicos se pueden aplicar en la simplificación del análisis de propiedades críticas de un sistema.

1.2.4 Patrón Arquitectónico.

En [Tyako 2009] se definen los patrones arquitectónicos como los pares compuestos de problemas-soluciones documentados que esta comprobado que la experiencia de diseño puede ser reusada, no solo por expertos si no tambien por gente menos experimentada.

Los patrones arquitectónicos identifican y especifican abstracciones arriba del nivel de clases simples e instancias o de componentes.

Los patrones arquitectónicos proveen un vocabulario común a los arquitectos que mejora su habilidad para comunicar y comprender el sistema. Proveen también medios de documentar la arquitectura del software. Además ayudan en la construcción del software con propiedades definidas.

Un patrón es especificado describiendo sus componentes, sus responsabilidades, sus relaciones y sus colaboraciones.

La documentación de un patrón incluye: nombre, contexto, problema, fuerzas, soluciones, estratégias, beneficios e incovenientes.

1.2.5 Lenguajes de Descripsión de la arquitectura por sus siglas en ingles ADL's (Arquitectural Description Lenguajes).

Los lenguajes de descripción de arquitecturas, “son usados para formalizar la representación de las arquitecturas de software, más especificamente los componentes y sus interacciones. Las herramientas que acompañan a los ADL's ayudan a analizar la arquitectura usando técnicas de simulación y verificación” [Tiako 2009].

Otra definición de Lenguaje de Descripción de Arquitectura **(ADL)** aparece en [Medvidovic 2006] como “una notación en la cual se pueden expresar los modelos de arquitectura”.

Las dos definiciones anteriores difieren en el grado de formalización que cada autor le brinda al lenguaje, mientras Tiako indica que estos lenguajes son usados para formalizar la representación de las arquitecturas, Medvidovic considera que toda notación que sea capaz de representar las desiciones de diseño consideradas principales por los stakeholders, es un ADL.

1.2.6 Modelo de arquitectura.

Es el documeto o artefacto que captura todas o algunas desiciones de diseño que componen la arquitectura del sistema. Estos también son llamados **descripción de la arquitectura.**

El Modelado de arquitectura también es el esfuerzo de capturar y documentar las decisiones de diseño que conforman la arquitectura del sistema.

Capítulo 2. Líneas de producto de Software.

2.1 Definición de una Línea de Producto.

Una línea de producto es un conjunto de sistemas que comparten un conjunto de características para satisfacer las necesidades específicas de un segmento de mercado particular. Las líneas de producto no son nuevas en manofactura, la industria automotríz, electrónica y de aviación tienen bastante tiempo explotando las líneas de producto, de tal forma que por ejemplo en la industria de aviación se puede llegar a usar cerca del 60% de los componentes de dos aviones totalmente distintos.

Una línea de producto de software, según el Framework for software Product Line Practice , versión 5.0, es un conjunto de sistemas de software producidos de forma intensiva, que comparten un conjunto común de características administradas para satisfacer las necesidades específicas de un segmento de mercado particular y que son desarrollados desde un conjunto común de core assets (activos nucleo, es la traducción más cercana, sin embargo de aquí en adelante el término core asset se dejará en inglés) de una forma preestablecida.

La actividad principal en la línea de producto de software es la *integración* más que la *programación desde cero*, cuando se llega a tener un conjunto amplio de core assets y estos se pueden insertar en los nuevos productos, entonces la Línea de productos de software llega a ser más económica. Para cada línea de productos de software, existe una guia predefinida o plan que especifica el enfoque exacto de la construcción del producto.

El conjunto común de assets y el plan de como son usados para construir productos no se realiza sin planear, requieren previsión organizacional, inversión planeación y dirección. Para producir los assets y el plan se requiere pensamiento estratégico que busque más allá de un simple producto. El uso disciplinado de los assets comunes no llega por si solo, la administración debe dirigir, dar seguimiento y forzar el uso de los assets. Las líneas de productos de Software son más que prácticas de negocio, son prácticas técnicas.

Las líneas de productos de software da elenfoque económico, lo que significa que la organización toma ventaja económica del hecho que muchos de tus productos son muy similares, porque fueron planeados de esa manera.

2.2 Conceptos relacionados con líneas de producto de software.

2.2.1 Core assets.

Son los artefactos y recursos reusables que forman la base para la línea de productos de software. Estos incluyen a: la arquitectura, componentes de software reusables, documentación, especificaciones modelos de dominio, statements de reqerimientos, modelos de desempeño, calendarios,

presupuestos, planes de pruebas, planes de trabajo, y descripción de procesos. La arquitectura es de los core assets clave [Northrop 2007].

2.2.2 Desarrollo.

Es un término genérico usado para describir cómo los core assets llegan a la organización, ya sea por construcción dentro de la organización (desde cero o modificando un componente) por medio de compra ó comisionandolo (haciendo un contrato con alguien más para que el lo haga por la organización).

2.2.3 Dominio.

Es un cuerpo especializado de conocimiento, un área de expertiz o una colección de funcionalidad relacionada.

2.2.4 La práctica de la línea de producto de software.

Es el uso sistemático de los core assets para ensamblar, instanciar o generar múltiples prodctos que constituyen una línea de productos de software.

2.2.5 Familia de productos.

Es el conjunto de productos al cual llamamos línea de productos.

2.3 Actividades escenciales de la línea de productos de software.

En escencia una línea de productos de software envuelve las siguientes actividades de **desarrollo** de Core assets, y desarrollo de productos usando los core assets, estas actividades se realizan bajo el eje de administración organizacional. Como se muestra en la Fígura 2, el desarrollo de los core assets y de nuevos productos ocurren en un orden cíclico que puede ser: un nuevo producto es construido a partir de core assets, o los core assets es extraido de los productos existentes.

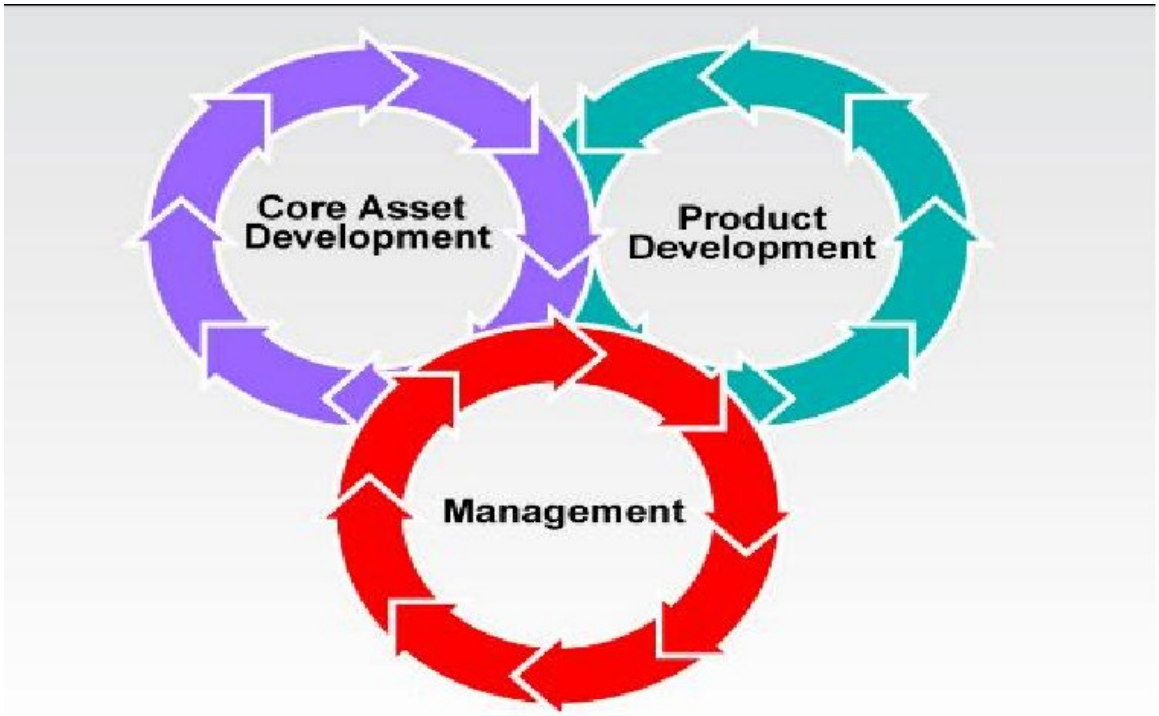


Figura 2: Actividades en una SPL [Northrop 2007] .

2.4 Áreas de práctica de las líneas de producto de software.

Para llevar a cabo las actividades escenciales de la línea de productos de software se deben dominar las áreas de prácticas relevantes a cada una y aplicarlas. Por dominar se entiende que es una abilidad de alcanzar el éxito de forma repetible, no solo una vez.

Un área de práctica es un cuerpo de trabajo o colección de actividades que una organización debe dominar para llevar a cabo las actividades de la línea de productos de software.

Las áreas de práctica de una Linea de productos de software se dividen en tres categorias:

Áreas de práctica de ingeniería de software. Son las prácticas necesarias para aplicar la tecnología apropiada para crear y evolucionar los productos y core assets. Las áreas de práctica que pertenecen a esta categoría son: Definición de la arquitectura, Evaluación de la arquitectura, Desarrollo de componentes, mineria de assets existentes, ingeniería de requerimientos, integración de sistemas de software, pruebas, entendimiento relevante del dominio y uso de software disponible externamente.

Áreas de práctica de administración técnica. Son las prácticas necesarias para administrar la creación y evolución de los productos y core assets.

Áreas de práctica de administración organizacional. Son las prácticas necesarias para orquestar el esfuerzo de la línea de productos.

Por razones de tiempo y espacio no se dará un mayor detalle de las áreas de práctica de administración técnica y administración organizacional; y solo se acotará el tema de Ingeniería de software a la definición de la arquitectura.

Capítulo 3. Definición de arquitecturas para líneas de productos de software.

La definición de la arquitectura es un área de práctica de las líneas de producto de software dentro de la categoría de áreas de práctica de ingeniería de software. Esta área de práctica describe las actividades que se deben realizar para definir una arquitectura de software.

La arquitectura es clave del éxito de cualquier proyecto de software. Es el primer artefacto del diseño que comienza a colocar los requerimientos en un espacio de la solución. La arquitectura correcta es absolutamente esencial para la buena navegación. Una arquitectura incorrecta es una receta para el desastre.

Para que una arquitectura pueda tener éxito, sus limitaciones se deben saber y deben ser articuladas. Otros drivers que un arquitecto experimentado sabe que debe tomar encuenta incluyen:

- los atributos de calidad que se requieren para cada producto a ser construido desde la arquitectura.

- Las metas del negocio que la organización desarrolladora tiene para el sistema. Éstos quizás incluyen las ambiciones para usar la arquitectura como la base para otros sistemas (o aún otras línea de productos de software). O quizás la organización desea desarrollar cierta competencia en un área. Consecuentemente, la arquitectura será influida fuertemente por esos deseos.

- Las mejores fuentes para componentes. Los componentes que son necesarios para el nuevo sistema ¿pueden ser implementados in house? ¿pueden ser comprados o licenciados de un mercado de componentes? Ó ¿pueden obtenerse a través de un contrato con terceros? O ¿pueden ser obtenidos del legado propio de la organización?

3.1 Aspectos peculiares que corresponden a una línea de productos de software.

La arquitectura de una línea de productos de software debe tomar en cuenta los reqerimientos la línea de producto, esto es toda arquitectura de Linea de producto de software debe tener identificados los mecanismos para soportar un conjunto de **variaciones explicitas.** Una de las tareas mas importantes del arquitecto de la línea de productos de software es identificar y seleccionar tales variaciones. Las variaciones escogidas deben soportar :

- las variacioes reflejadas en los productos. Los constraints de productos proveen información acerca de las variaciones imaginadas en los

productos de la línea, tales variaciones deben ser soportadas por la arquitectura. Estas variaciones freecuentemente se manifiestan como atributos de calidad.

- Otros aspectos que deben soportar las variaciones son la estrategia de producción y las constraints de producción. Los mecanismos de variación deben soportar la forma como la organización planea construir los productos.

- También las variaciones deben soportar la integración eficiente. Una línea de productos con un gran número de productos y actualizaciones requiere de un proceso de integración fácil para cada producto.

Los productos en una línea de productos de software existen simultaneamente y pueden variar entre ellos en terminos de su comportamiento, atributos de calidad, plataforma y de otras formas. Cada producto tiene su propia arquitectura, la cual es una instancia de la arquitectura de la línea de productos alcanzada por la ejercitación de los mecanismos de variabilidad. De aquí que una organización comprometida con el desarrollo de productos en una línea de productos deba manejar varias arquitecturas.

3.2 Lenguajes de descripción de arquitecturas (ADL's) que soportan la descripción de una línea de productos de software.

Evidentemente no todos los lenguajes o notaciones para documentar arquitecturas son ideales para documentar la arquitectura de una línea de productos de software; Medvidovic en [Medvidovic 2006] provee de una reseña de los ADL's de segunda generación y desde el punto de vista de su origen permiten ver que no todos sirben para la descripción de la arquitectura de una linea de productos. Desde el punto de vista de Medvidovic los primeros ADL's surgieron a la luz principalmente de la tecnología, es decir para modelar la aruitectura de acuerdo a cierta tecnologí escogida principalmente por el nivel de expertiz que se tenía de ella. Estos ADL's modelaban principalmente conceptos genéricos, descripciones de lenguaje, herramientas e infraestructura enfocada en concerns recurrentes en el desarrollo de sistemas de software.

Posteriormente comenzaron a surgir otros ADL's tambien de primera generación a la luz de los aspectos importantes (concerns) relacionados no solo con la tecnología, si no también con el dominio de la aplicación y con el negocio en el que está inmerso el desarrollo de software.

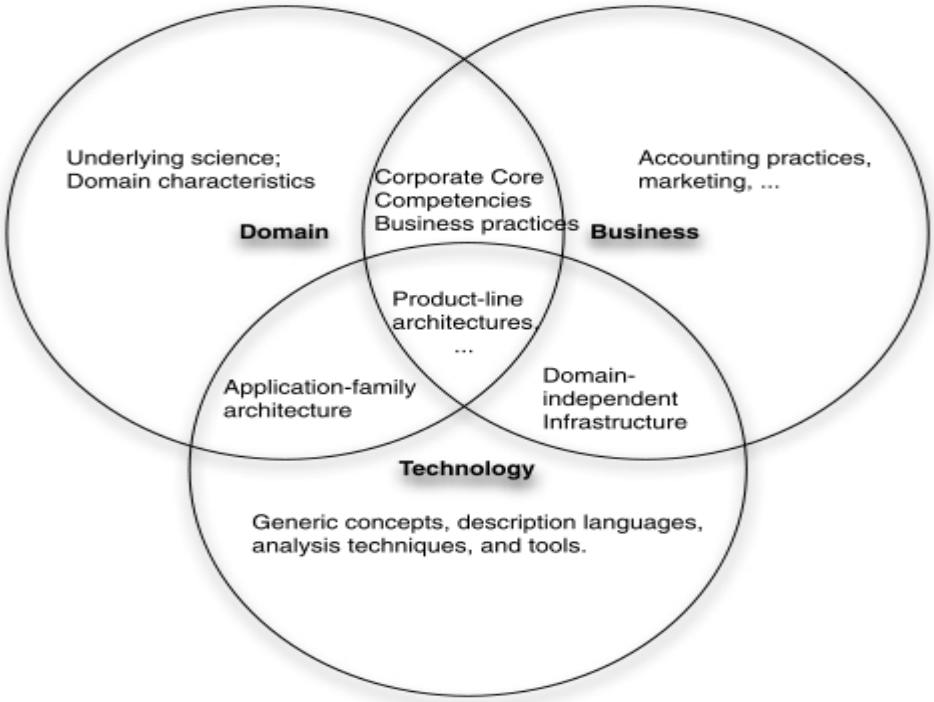
Dominio. Comprende los concerns considerando la naturaleza fundamental, principios y ciencias de un dominio así como también características que se cree son irelevantes a las perspectivas tecnología o negocios.

Negocios. Contiene las facetas del negocio independientes del dominio de la organización de desarrollo, independientes de expertiz e independiente de tecnología en la medida que este se relaciona en desarrollo de productos. Incluye prácticas de contabilidad financiera, recursos humanos, etc.

Los ADL's no representan solo un tipo de concerns, sobre todo los de segunda generación buscan representar a más de un tipo, formando así las siguientes interrelaciones.

Intercección Tecnología dominio. Incluye los concerns tecnológicos que son específicos para un dominio particular. Esto es donde el área tecnológica ataca ese problema que ocurre cuando se construyen sistemas de software en general, las áreas del dominio más la de la tecnología atacan ese problema técnico que ocurre cuando se construye un sistema de software en un dominio específico. La arquitectura de familias de aplicaciones coinciden con esta subárea como lenguaje de modelado de dominio específico para capturar esas familias de arquitecturas. Por ejemplo el lenguaje Weaves es un ejemplo de formalismo; otros ejemplos son MetaH (el dominio es control) ADAGE (Su dominio es guía para aviación, navegación y control).

Intersección Tecnología Negocios. Liga los concerns de negocio tal como costos, relaciones y procesos a la construcción técnica de sistemas de software independientemente de cualquier dominio particular ejemplos de herramientas que caen en esta área son enfoques de administración de procesos y proyectos



Fígura 3: Origenes de los ADLs y sus Intersecciones [Medvidovic 2006]

de software que relacionan pasos de procesos a elementos de software específicos, sistemas de administración de la configuración que dan seguimiento a las relaciones de varios elementos de software a uno, y las herramientas y notaciones de modelado de costos centrado en arquitectura, como estas herramientas son de dominio independiente se dice que tienen una caracterísitica “one-size-fits-all”.

Intersección Dominio y Negocio incluye las competencias nucleo de una organización: ese conocimiento de un dominio combinado con estrategias de negocio y prácticas el cuál hace posible que la organización tenga éxito en ese dominio. Esta área es tipificada como un foco sobre el modelado del dominio, requerimientos del cliente y procesos para desarrollar una aplicación mientras se difieree en la decisión de detalles tecnológicos específicos que se resolverán en posteriores fases de desarrollo.

Intersección tecnología-dominio-negocios. El conocimiento de un dominio de aplicación combinado con una estrategia de negocio para el dominio y soportado por una tecnología posibilita la representación y razonamiento acerca de una **familia de aplicaciones** comprendiendo una **business product line** (línea de productos de negocio).

Entonces ya que tocamos que los ADL's que surgen a partir de la intersección Tecnología-dominio-negocios son aquellos que son capaces de representar el razonamiento de una familia de aplicaciones me permito introducir los lenguajes que caen dentro de esa categoría: KOALA Y XADL.

3.2.1 El lenguaje KOALA.

Es un lenguaje de descripción de arquitectura derivado de DARWIN ADL, desarrollado por PHILIPS; contiene aspectos de estilos arquitectónicos, sin embargo desde que prescribe patrones y semantica específicos que son aplicados a los **constructs** descritos en el ADL KOALA, Koala es efectivamente una notación estructural. Maneja los conceptos heredados desde Darwin de: componentes, interfaces (requeridas y provistas) composiciones gerarquicas (componentes con su propia estructura interna) y links para conectar la interfaces, además de estos constructs básicos, Koala tiene constructs Especiales para soportar la **variabilidad** de las **líneas de producto**.

La principal inovación de Koala respectoa a Darwin incluye:

Tipos de Interface basados en IDL: un tipo de interface en Koala es un denominado conjunto de firmas de funciones, similar a los encontrados en **C**.

Diversidad de interfaces: una de las filosofías de koala es que los parametros de configuración de un componente no deberían ser almacenados en el componente; en su lugar los parámetros de configuración, incluyendo selección de alternativas deberían ser accesados por el componente desde una fuente externa cuando sean necesarios, lo que permite a la aplicación ser configurada centralmente, desde un simple componente o conjunto de componentes cuyo

proposito es proveer datos de configuración para la aplicación. La interfaces diversas son interfaces especiales requeridas que se adjuntan a los componentes y son usadas por cada componente para obtener parámetros de configuración.

Switches: un switch es una construcción arquitectónica nueva que representa un **punto de variación**. Permite a una interfaz requerida ser conectada a múltiples y diferentes interfaces provistas. Cuando se resuelve la variación, solo una de las conecciones será verdaderamente presentada. El par de interfaz requerida-provista conectada dependiendo de la condición de configuración. Un switch es conectado a una **diversidad de interfaces** para obtener sus parámetros de configuración, tal como un componente sería conectado, dependiendo de los valores regresados por la diversidad de interfaces, el switch ruteará llamadas a una de las interfaces requeridas conectadas a el. Si esto significa que habrá componetes desconectados, esto es, componentes que jamas serán invocados, entonces koala no instanciará tales componentes para ahorrar recursos.

Interfaces opcionales: varios componentes podrán proveer servicios similares, pero no idénticos.

Koala parece estar bajo la luz de los tres lamppost es decir, parece atacar concerns tecnológicos, de dominio y de negocios. La configuración explicita y mapeo directo a la implementación fija las descripciones koala en **tecnología**. Las características provistas por koala son utilizadas para el modelado de sistemas en un dominio particular. Aunque koala fue desarrollado para el dominio de sistemas incrustrados en electrónicos, nada previene su uso en otras líneas de productos de sistemas incrustrados.

Los aspectos de línea de prductos de koala son dirigidos primeramente por concerns de negocios. Koala permite especificar ciertas desiciones de negocio (la relación entre productos) directamente en el lenguaje. Otros concern por ejemplo reducción de time-to-market y reducciónde costos a través del incremento del reuso. Influencian la selección particular de capacidades del ADL Koala, aunque no es necesarimente expresadas en los modelos koala.

3.2.2 El lenguaje xADL.

Apunta a direccionar los issues no cubiertos por UML proviendo una plataforma sobre la cual las características de modelado pueden ser definidas modularmente y usadas a través de proyectos. Nuevas características pueden ser creadas y agregadas al lenguaje como entidades de primera clase. XADL hereda lecciones desde ADL's de primera generación así como conceptos de otros de segunda generación como las líneas de productos de Koala. Su **mayor contribución** es su soporte para extención de lenguaje. En una manera xADL puede ser visto como una fabrica de ADL: los usuarios lo pueden usar para desarrollar rápidamente un ADL hecho a la medida a las necesidades del dominio y metas del negocio.

xADL puede ser apliado con constructs imprevistos y estructuras en formas arbitrarias, esto induce el reto que no existe un lenguaje con sintaxis y semántica estable. Deben ser desarrollados herramientas como editores, analizadores parsers y otras para cubrir una notación cuya sintaxis pueda cambiar entre proyectos. Estos retos son atacados con un conjunto de herramientas, cada una de las cuales tiene un soporte específico para tratar con nuevos esquemas. Este soporte esta dentro de la forma de adopción automática para nuevos esquemas, en el caso de herramientas dirigidas a sintaxis, o guias o APIs que permitan a los desarroladores conectar su propio soporte para nuevos esquemas.

3.2.2.1 Aspectos que incluye el xADL para dar soporte a nuevos esquemas.

El data binding library: es una libreria de clases Java que corresponde a elementos XML y atributos definidos en esquemas xADL estos elementos proveen una interface a través de la cual las herramientas pueden convertir, leer, modificar y serializar (escribir a disco) documentos xADL. Para soportar la extensibilidad de xADL la libreria misma es modularizada como el lenguaje xADL: cada esquema xADL es mapeado a un paquete de clases Java. Entonces agregar nuevos paquetes extienden la libreria para soportar nuevos esquemas.

Apigen: para evitar la escritura manual del data binding library, cada vez que fuese agregado un esquema, se tiene Apigen que es un generador de data binding library.

ArchEdit: es un editor para documentos xADL basado en GUI dirijido a sintaxis. El cual permite a los usuarios navegar por el documento xADL navegar a través de un árbol que refleja la estructura gerárquica del documento xADL XML. Las opciones disponibles al usuario (como que tipo de elementos pueden ser agregados) son determinadas automáticamente dependiendo del conjunto de esquemas xADL en uso. Cuando se agregan nuevos esquemas, ArchEdit adapta automáticamente el soporte a ellos.

XADL 2.0 se dirige directamente a aspectos de tecnología, negocio y dominio.

Las características centradas en tecnología incluyen la habilidad para modelar estructura arquitectónica, tipos e instancias.

Los esquemas relacionados a negocios incluyen esquemas de xADL de líneas de producto, lo cual permite a xADL dar seguimiento a la evolución de productos sobre el tiempo y a sus relaciones con otros productos.

XADL tiene pocos esquemas de dominio específico tal como modelado de comportamiento de sistemas basados en mensajes asincronos: pero los usuarios pueden agregar sus propios esquemas de dominio especifico.

Capítulo 4. Conclusiones.

El principal atributo de calidad con el que debe cumplir una arquitectura de linea de producto es la variabilidad.

Los lenguajes de descripción de arquitecturas nos dan la pauta para representar las variabilidades de una línea de productos de software, lo que no he identificado es como seleccionar las variabilidades lo cual como se indica en [Northrop 2007] es una de las actividades principales del arquitecto. El método de diseño dirigido por atributos (ADD) puede ayudar a identificar los atributos de calidad de la arquitectura (en general de cualquier arquitectura) habrá que estudiarlo de cerca para ver si sirbe para la identificación de la variabilidad.

Bibliografía

- [Northrop 2007] Northrop Linda M., Clements Paul C. A Framework for Software Product Line Practice, Version 5.0, July 2007. http://www.sei.cmu.edu/productlines/framework.html
- [Tiako 2009]Tiako Pierre F. Designing Software-Intensive Systems: Methods and Principles. Chapter IX - A Survey of Software Architecture Approaches. 2009.
- [IEEE 1471] IEEE Standard 1471 -IEEE 1471, 2000; Maier, Emery, & Hilliard, 2001
- [Medvidovic 2006] Medvidovic Nenad, Dashofy Eric M. , Taylor Richard N. Moving architectural description from under the technology lamppost. Available online 10 October 2006.
- [Fowler]Fowler Martin. Who Needs an Architect?
- [Galan 1995] Garlan David, Allen Robert, Ockerbloom John. Architectural Mismatch or Why it's hard to build systems out of existing parts. 1995. http://www.cs.cmu.edu/~able/publications/archmismatch-icse17/