Centro de Investigación en Matemáticas, A.C.

Attribute Driven Design (ADD) y Software Product Line (SPL)

REPORTE TÉCNICO

Que para obtener el grado de
Maestro en Ingeniería de Software

P r e s e n t a:
Heriberto Ortega Domínguez

Director de Tesis:
Dr. Cuauhtémoc Lemus Olalde

Zacatecas, Zac. Julio de 2010

Contenido

Índice de Figuras

1 Introducción

1.1 Definición del problema y alcance del Reporte Técnico

Aunque el diseño de la arquitectura de software depende en gran medida de la experiencia e incluso la intuición del arquitecto, en años recientes se están desarrollando métodos enfocados en sistematizar este proceso de diseño.

La etapa de requerimientos se enfoca más en la funcionalidad deseada, la descripción operacional, el diseño de la arquitectura, el diseño detallado y la implementación, pero los atributos de calidad, por general, no son claramente identificados en dicha etapa, por lo que no son debidamente especificados, lo cual trae como consecuencia que no sean bien entendidos y por lo tanto son débilmente implementados.

Attribute Driven Design (ADD) es un método que mediante el análisis de los atributos de calidad definidos en la fase de requerimientos, obtiene una arquitectura inicial del sistema, identificando módulos, componentes y conectores.

Por otro lado, el paradigma de Software Product Line (SPL) se centra en desarrollar una familia de sistemas que tienen como base una funcionalidad común y que al aplicar variaciones se obtienen productos diferentes.

En el presente reporte se realizará un análisis del método ADD y su aplicación en Software Product Line, con el objetivo de buscar si se pueden obtener desde la fase de diseño arquitecturas reutilizables y modificables que den como resultado productos de software diferentes a partir de una arquitectura base.

1.2 Contenido del Reporte Técnico

En este reporte se describe el resultado del análisis realizado a un conjunto de artículos con la finalidad de conocer a detalle los pasos que conforman el método Attribute Driven Design, ilustrar su implementación y los casos registrados recientemente sobre la implementación del método bajo el enfoque de Software Product Line.

1.3 Criterios de selección de fuentes de consulta

La estrategia para la búsqueda de información fue a través de la exploración en Internet utilizando motores de búsqueda para localizar artículos sobre el tema ADD y su aplicación en SPL. En el sitio de Internet del Software Enginnering Institute (SEI), creador del método Attribute Driven Design, fue donde se localizaron la mayoría de artículos analizados.

Se seleccionaron todos los artículos que abordaban ambos temas por separado o en el mismo artículo o documento; posteriormente se hizo una depuración de documentos conforme se consultaban de acuerdo al nivel de detalle del manejo de información que presentaron.

2 El Método Attribute Driven Design (ver. 2.0)

2.1 Qué es el método ADD

Attribute Driven Design (ADD) es un método creado por el Software Engineering Institute para sistematizar la definición de la arquitectura de software. Utiliza como elementos de entrada los atributos de calidad, los requerimientos funcionales y las decisiones de diseño definidos en la etapa de requerimientos de un proyecto y priorizados por los stakeholders.

La estrategia utilizada por ADD es descomponer a un sistema en varios elementos y aplicar tácticas arquitectónicas en busca de satisfacer el cumplimiento de los atributos de calidad. En cada nivel de descomposición son validados los requerimientos de calidad y los requerimientos funcionales.

Una vez aplicado el método se obtiene un diseño o arquitectura inicial de software la cual contiene módulos, componentes y conectores que ilustran la relación entre elementos.

Los resultados que se obtienen al aplicar el método son:

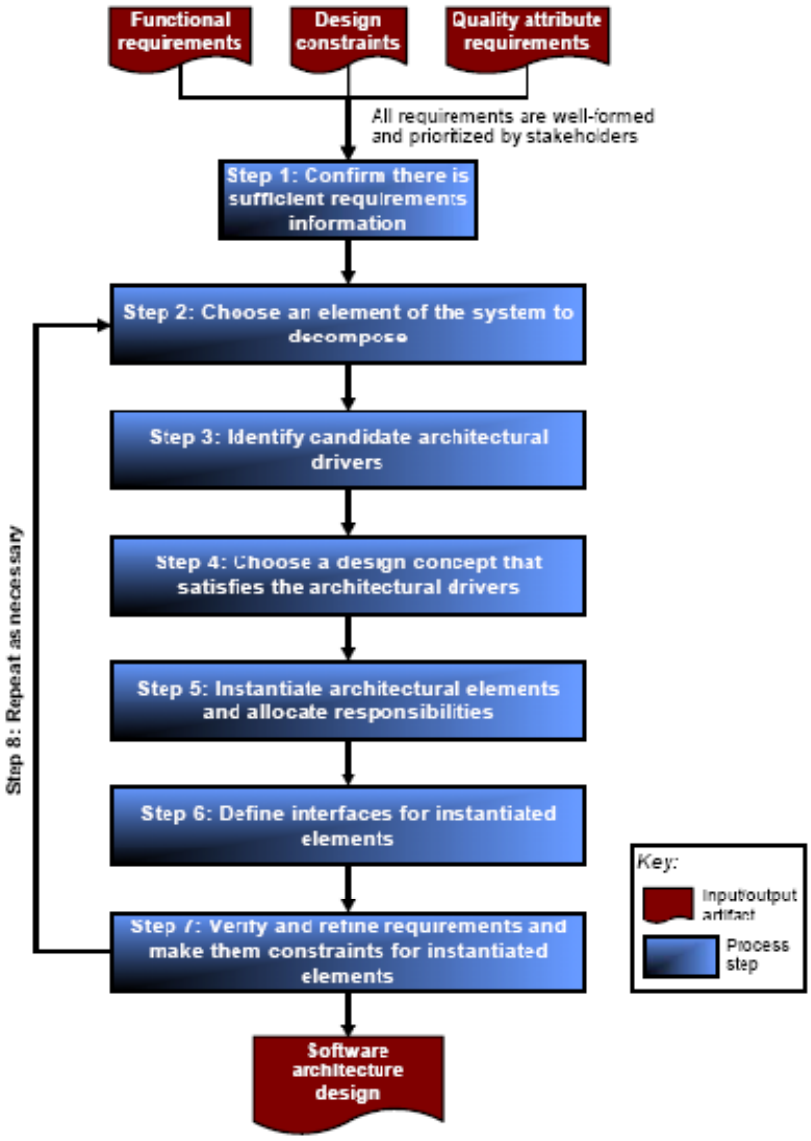
- Conjunto de decisiones de diseño estructuradas.
- Interconexión y coordinación de mecanismos.
- Aplicación de patrones y tácticas arquitectónicas para especificar partes de la arquitectura.
- Requerimientos de atributos de calidad cubiertos.
- No se obtienen interfaces detalladas.

El método se compone de los siguientes 8 pasos, como se muestra en la Figura 1.

1. Confirmar que existe suficiente información de requerimientos.
2. Elegir un elemento del sistema para descomponerlo.
3. Identificar drivers arquitectónicos desde el conjunto de escenarios de calidad y requerimientos funcionales.
4. Elegir atributos primitivos que satisfagan las características arquitectónicas.
5. Instanciar los elementos arquitectónicos y asignar responsabilidades.
6. Definir interfaces para elementos instanciados.
7. Verificar y refinar los requerimientos (casos de uso y escenarios de calidad) y aplicar las restricciones a los elementos instanciados.
8. Repetir del paso 2 al 7 para los siguientes elementos del sistema que se desee descomponer.

Antes de iniciar la implementación del método es importante que los atributos de calidad hayan sido priorizados por los stakeholders de acuerdo a las metas de negocio y la misión del cliente. Los requerimientos de calidad deben estar expresados en escenarios de calidad específicos.

Figura 1. Pasos del método ADD



Fuente: Reporte Técnico Attribute Driven Design (ADD) Versión 2.0, SEI / Carnegie Mellon

2.2 Pasos del método ADD

1. Confirmar que existe suficiente información de requerimientos.

En este paso se verifica que se cuenta con:

- Los requerimientos priorizados de acuerdo a las metas de negocio.
- Atributos de calidad expresados en escenarios de calidad (ver Figura 2. Ejemplo de escenario de calidad).

De todos los requerimientos que se tienen registrados para el sistema, sólo pocos son significativos para la arquitectura. El proceso puede iniciar cuando se tiene seguridad de que se conocen los drivers arquitectónicos de los requerimientos.

Figura 2. Ejemplo de escenario de calidad

Quality Attribute Scenario 1: Quick Recovery

Element	Statement
Stimulus	A Track Manager software or hardware component fails.
Stimulus source	A fault occurs in a Track Manager software or hardware component.
Environment	Many software clients are using this service. At the time of failure, the component may be servicing a number of clients concurrently with other queued requests.
Artifact	Track Manager
Response	All query requests made by clients before and during the failure must be honored. Update service requests can be ignored for up to two seconds without noticeable loss of accuracy.
Response measure	The secondary replica must be promoted to primary and start processing update requests within two seconds of the occurrence of a fault. Any query responses that are underway (or made near the failure time) must be responded to within three seconds of additional time (on average).

Fuente: A Practical Example of Appliying Attribute-Driven Design (ADD) Ver. 2.0 SEI – Carnegie Mellon. William G. wood

2. Elegir un elemento del sistema para descomponerlo.

Cuando inicia el proceso el elemento a descomponer es por lo general el sistema entero.

El elemento a descomponer debe satisfacer los drivers arquitectónicos. En este proceso influyen factores tales como:

- El número de dependencias que tendrá con otros elementos del sistema.
- Los riesgos o dificultades para asociar los requerimientos al elemento.
- Criterios de negocio (por ejemplo el impacto en el mercado).
- Criterios organizacionales (por ejemplo impacto del uso de recursos humanos y cómputo).

3. Identificar drivers arquitectónicos[1] desde el conjunto de escenarios de calidad y requerimientos funcionales.

Se clasifican los requerimientos previamente priorizados por los stakeholders en alto, mediano o bajo, mediante grupos pares (H,H), en donde el primer elemento representa la importancia que tiene el requerimiento para los stakeholders y el segundo elemento indica la dificultad (impacto potencial del requerimiento en la arquitectura). La Figura 3 muestra una tabla de priorización de drivers arquitectónicos.

Figura 3. Priorizando drivers arquitectónicos

Architectural Driver Priorities

#	Architectural Drivers	Section Discussed In	Importance	Difficulty
1	Scenario 1 Quick Recovery	2.3	high	high
2	Scenario 2 Slow Recovery	2.3	medium	medium
3	~~Scenario 3~~ ~~Restart~~	~~2.3~~	~~low~~	~~low~~
4	Requirement 1 Track Manager Functionality	2.1	high	high
5	Design Constraint 1 Capacity Restrictions	2.2	high	high
6	Design Constraint 2 Persistent Storage Service	2.2	medium	low
7	Design Constraint 3 Two Replicas	2.2	high	high
8	ADD Step 1, #2 Deployment Characteristics	3.2	high	high
9	ADD Step 1, #3 Communication Mechanisms	3.2	high	low
10	ADD Step 1, #4 Checkpoint Timing	3.2	high	high

Fuente: A Practical Example of Appliying Attribute-Driven Design (ADD) Ver. 2.0 SEI – Carnegie Mellon. William G. wood

[1] Elementos de entrada al proceso de ADD que tienen un gran impacto en la estructura de la arquitectura.

Se eligen requerimientos de los que los pares resultantes son de alta prioridad (drivers arquitectónicos) los cuales serán utilizados en el resto de los pasos del proceso.

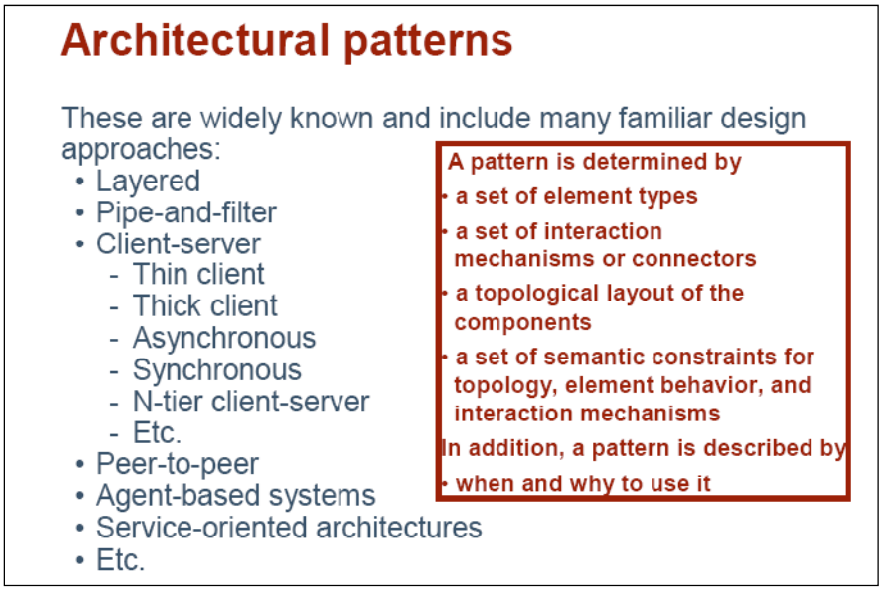
4. Elegir los atributos primitivos que satisfagan las características arquitectónicas.

En esta parte se elige el mayor número de elementos que formarán parte de la arquitectura y se identifican las relaciones que habrá entre ellos.

Al inicio de toda arquitectura, se toman en cuenta los llamados estilos o patrones arquitectónicos. Son soluciones a problemas que se han probado previamente que constituyen un conjunto de decisiones de diseño y que describen una clase de arquitectura en un enfoque general. En la Figura 4. Se listan los principales patrones arquitectónicos conocidos.

Las tácticas (decisiones de diseño con un enfoque más detallado) y los atributos de calidad son utilizados en el método ADD para determinar los tipos de elementos, las relaciones y sus interacciones para satisfacer los requerimientos de calidad.

Figura 4. Tipos de patrones arquitectónicos



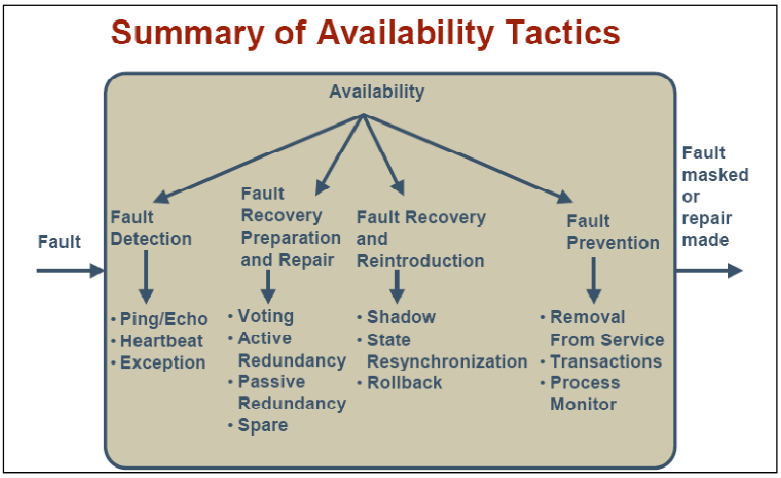
Fuente: Current Best Practices in Software Architecture
Paul Clements, Software Engineering Institute
Carnegie Mellon University

En esta etapa, el arquitecto debe seguir los siguientes 6 subpasos:

4.1 Identificar tácticas arquitectónicas que son asociadas con los drivers de diseño arquitectónico.

Por ejemplo, para el atributo de calidad availability, algunas de las tácticas arquitectónicas definidas son: prevención a fallos, detección de fallos y recuperación de fallos como se muestra en la Figura 5.

Figura 5. Tácticas del atributo de calidad Availability



Fuente: Current Best Practices in Software Architecture
Paul Clements, Software Engineering Institute
Carnegie Mellon University

4.2 Para cada táctica arquitectónica identificar una lista de patrones subordinados.

Para identificar los patrones subordinados se debe tomar en cuenta:

- El conocimiento, habilidades y experiencia a cerca de los patrones arquitectónicos previamente descritos.
- El conocimiento previo de tácticas arquitectónicas para lograr los atributos de calidad.
- Si un driver arquitectónico es de mayor dificultad que un atributo de calidad, quizá se le deban aplicar múltiples tácticas.
- Otras fuentes tal como libros, artículos, material de conferencias o motores de búsqueda.

Los patrones subordinados son aquellas opciones conocidas que pueden conseguir que una táctica arquitectónica sea satisfecha, en la Figura 6 se muestran patrones subordinados para la táctica de recuperación a fallos.

Por ejemplo, para la táctica de prevención a fallos, uno de los patrones subordinados identificados es process monitor (monitoreo de componentes críticos: remover el componente de un servicio y reinstanciar un nuevo proceso en su lugar).

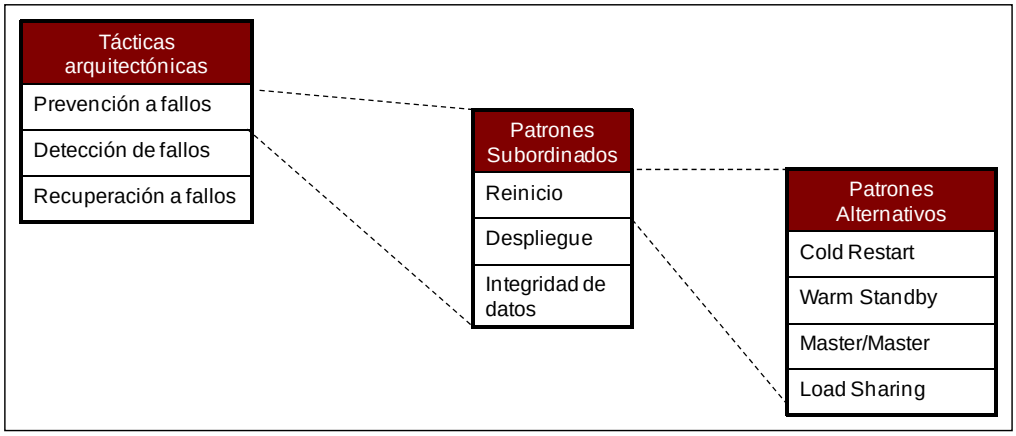
Para cada patrón subordinado de la lista, se debe:

a. Identificar cada parámetro diferenciador del patrón que ayude a elegir entre el patrón y la táctica en la lista.

Por ejemplo, en cualquier patrón de reinicio, la cantidad de tiempo que se toma para reiniciar es un parámetro diferenciador. Para patrones usados para lograr modificability (por ejemplo layering) un parámetro diferenciador es el número de dependencias que existen entre los elementos en el patrón.

b. Estimar los valores de los parámetros diferenciadores.

Figura 6. Identificando tácticas, patrones subordinados y alternativos



4.3 Seleccionar patrones alternativos que sean más apropiados para satisfacer los drivers arquitectónicos.

Realizar esta selección de manera razonable. Decidir cuáles patrones son apropiados.

a. Crear una matriz (Figura 7) con los nombres de los patrones en los títulos de las columnas y los drivers arquitectónicos listados en el lado izquierdo. Usar la matriz para analizar las ventajas y desventajas (pros y contras) de aplicar cada patrón para cada driver arquitectónico.

Ejemplo de matriz de selección de patrones alternativos, en este caso, para el patrón subordinado de Reinicio que pertenece a la táctica Prevención a fallos.

Figura 7. Matriz de selección de patrones alternativos

Driver Arquitectónico	Patrones alternativos							
	Cold restart		Warm standby		Master/Master		Load sharing	
	pros	contras	pros	contras	pros	contras	pros	contras
Escenario: Rápida recuperación a fallos La actualización de peticiones de servicio se puede ignorar máximo 2 seg. Las peticiones de servicios de consulta deben ser respondidas en 3 segundos más		Pérdida del servicio Tiempo muerto > 2 minutos	Tiempo muerto > 0.03 seg. Fácil de aplicar	No es probable que se pierda el servicio	Tiempo muerto > 50 ms No se pierde el servicio.	Difícil de aplicar	Tiempo muerto > 50 ms No se pierde el servicio	Difícil de aplicar

Considerar lo siguiente al hacer la selección:

- ¿Qué beneficios se esperan cuando se usa cada patrón?
- ¿Cómo hacer la mejor combinación de patrones entre sí?
- ¿Son mutuamente exclusivos todos los patrones?

b. Elegir los patrones que logren satisfacer juntos los drivers arquitectónicos.

De acuerdo al ejemplo de la matriz de selección de patrones alternativos, el driver arquitectónico requiere un tiempo de reinicio para recuperarse a un fallo máximo de 2 segundos, se puede apreciar que cold restart no es apropiado. Por otro lado master/master y load sharing, se desecharon

debido a que su aplicación es más compleja. Asimismo warm standby cumple fácilmente con el requerimiento, por lo que es el patrón elegido.

Lo anterior implica que se tendrá un componente primario que recibirá las peticiones y efectuará las respuestas. Adicionalmente, se contará con un componente secundario que será cargado en otro procesador y realizará las mismas acciones.

4.4 Considerar la identificación de patrones que se tenga hasta el momento y decidir cómo se relacionan con los otros.

La combinación de patrones seleccionados dará como resultado un nuevo patrón.

a. Decidir cómo son relacionados los tipos de elementos desde varios patrones.
b. Decidir qué tipos de elementos desde varios patrones son relacionados.
c. Tomar en cuenta la funcionalidad y uso como un indicador para cada combinación de patrones.
d. Identificar nuevos tipos de elementos que resultan de la combinación de patrones.
e. Revisar la lista de decisiones de diseño y confirmar que se han hecho todas las decisiones relevantes.

4.5 Describir los patrones seleccionados para iniciar la documentación de las diferentes vistas arquitectónicas.

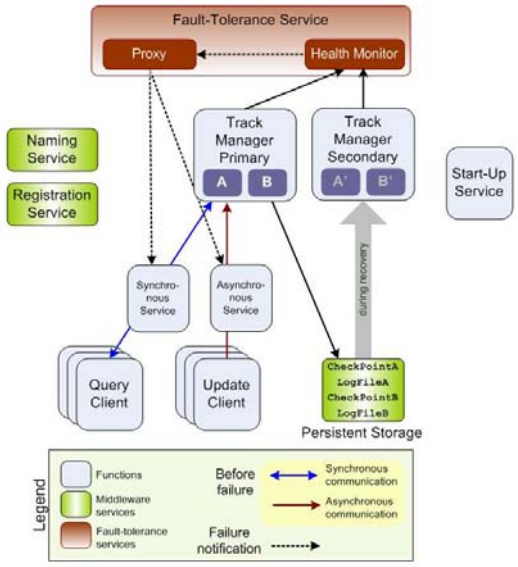
No se necesita crear completamente las vistas arquitectónicas documentadas hasta este punto. Documentar cualquier información que de seguro se necesitará sobre la arquitectura (incluyendo lo que se sabe sobre las propiedades de los diferentes tipos de elementos). Idealmente se deben usar formatos de vistas para capturar esta información.

Una de las vistas que se puede crear es la vista funcional (ver Figura 8) de los elementos y las relaciones entre ellos, que hasta el momento se tienen identificados. Otra vista que en este punto se puede capturar es un diagrama de secuencia.

Figura 8. Vista funcional de elementos de software en la arquitectura y su relación, de acuerdo a los elementos identificados

System Elements and the ADD Iteration in Which They're Developed

#	This Element	Is Developed in This ADD Iteration
1	Track Manager	Requirement
2	Query clients	Requirement
3	Update clients	Requirement
4	Persistent storage	Requirement
5	Track Manager A	1
6	Track Manager B	1
7	Synchronous communications	1
8	Asynchronous communications	1
9	Naming service	1
10	Registration service	1
11	Health monitor	2
12	Proxy server	2
13	CheckpointFileA	2
14	CheckpointFileB	2
15	LogFileA	2
16	LogFileB	2



Fuente: A Practical Example of Appliying Attribute-Driven Design (ADD) Ver. 2.0 SEI – Carnegie Mellon. William G. wood

4.6 Evaluar y resolver inconsistencias en el diseño conceptual.

En esta parte el arquitecto puede construir modelos para describir el comportamiento del sistema.

Las tareas iniciales de la evaluación de inconsistencias son:

a. Evaluar el diseño contra los drivers arquitectónicos. Si es necesario, usar patrones, experimentos, simulaciones, análisis formal y métodos de evaluación de arquitecturas.
b. Determinar si hay algún driver arquitectónico que no fue considerado.
c. Evaluar patrones alternativos o aplicar tácticas adicionales, si el diseño no satisface los drivers arquitectónicos.
d. Evaluar el diseño del elemento actual contra el diseño de otros elementos en la arquitectura y resolver cualquier inconsistencia. Por ejemplo, mientras se diseña cualquier elemento, quizá se descubran ciertas propiedades que deberían ser difundidas para otros elementos en la arquitectura.

Las decisiones de diseño tomadas durante el paso 4 son:

a. Decidir sobre un conjunto de conceptos que incluyan el mayor tipo de elementos que aparecerán en la arquitectura y el tipo de relaciones entre ellos.
b. Tener identificada alguna de la funcionalidad asociada con los diferentes tipos de elementos.
c. Conocer cómo y cuándo los tipos de elementos de software se mapean el uno al otro (estática o dinámicamente).
d. Pensar en la comunicación entre los diferentes tipos de elementos (elementos internos de software y entidades externas).

Considerar además:

- Qué tipo de elementos necesitan comunicarse con otro
- Qué clase de mecanismos y protocolos serán usados por la comunicación entre los elementos de software y entidades externas (síncrona, asíncrona, híbrida acoplada, o llamada remota contra llamada local)
- Las propiedades de los mecanismos que serán usadas para la comunicación entre los elementos de software y las entidades externas (síncrona, asíncrona, híbrida acoplada, capacidad de cola y confiabilidad)
- Atributos de calidad de requerimientos asociados con los mecanismos de comunicación
- Los patrones de datos sobre las dependencias de comunicación
- Los tipos de elementos computacionales que soportan las diferentes categorías de uso del sistema
- Cómo los componentes heredados y componentes que están fuera (COTS) serán integrados en el diseño.

Hasta aquí se tiene razonado sobre los elementos de software y recursos del sistema, pero quizás se tienen decisiones diferidas a cerca de ellos. Para eso se debe considerar:

- Qué recursos son requeridos por los elementos de software.
- Qué recursos necesitan ser administrados.
- Los límites de los recursos.
- Cómo serán administradas los recursos.
- Qué estrategias de calendarización serán empleadas.
- Qué elementos son de estado.
- Los mejores modos de operación.

Hasta aquí se tienen pensadas las dependencias entre los diferentes tipos de elementos internos de software pero quizás se tienen también decisiones diferidas a cerca de éstas. Para ello se debe considerar:

- Qué dependencias de ejecución existen entre elementos.
- Cómo y dónde son resueltas las dependencias de ejecución entre elementos.
- La activación y desactivación de dependencias entre elementos de software.

Se debería incluso considerar lo siguiente:

- Los mecanismos de abstracción utilizados.
- Qué elementos del sistema se conocen a la fecha.
- Qué modelos proceso/hilos será empleados.
- Cómo serán dirigidos los requerimientos de atributos de calidad.

5. Instanciar los elementos arquitectónicos y asignar responsabilidades.

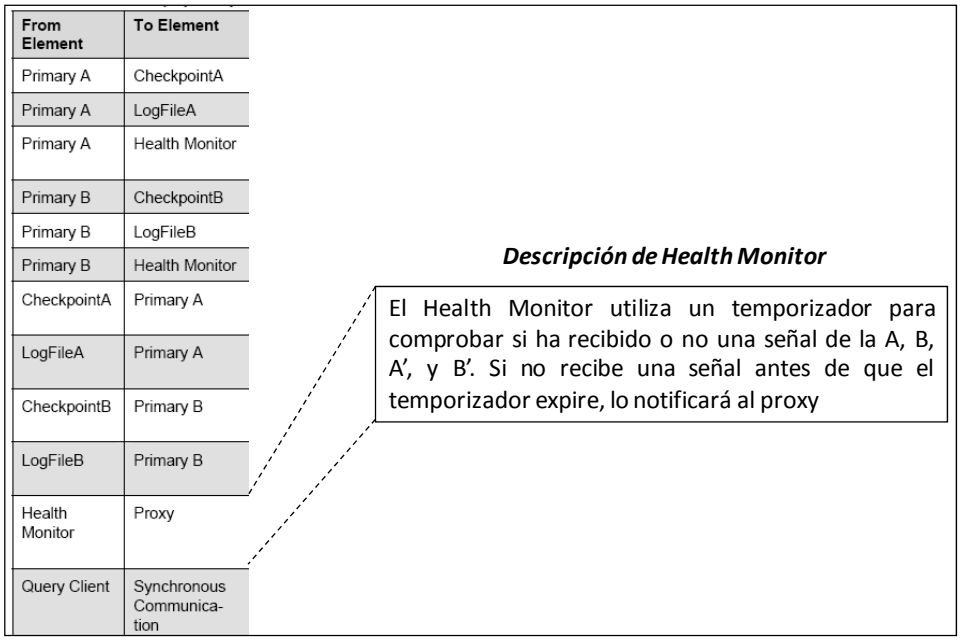
En esta etapa se muestra como los elementos de diseño serán instanciados para usar los drivers arquitectónicos funcionales. Los requerimientos funcionales ayudan a determinar grupos o instancias de elementos específicos de diseño.

Los drivers funcionales son derivados de los requerimientos funcionales abstractos (por ejemplo las características) o de requerimientos funcionales concretos (casos de uso o lista de responsabilidades).

Actividades a seguir en este paso:

- Asignar los requerimientos funcionales del elemento padre a los elementos hijo.
- Definir responsabilidades de los elementos hijo (ver Figura 9, ejemplo de responsabilidades del proxy con health monitor).
- Descubrir el intercambio necesario de información entre elementos, creando una relación productor/consumidor.
- Especificar interacciones entre elementos (llamadas, suscripciones a y notificaciones).
- Representar la arquitectura con vistas.

Figura 9. Ejemplo de responsabilidades del proxy con health monitor



From Element	To Element
Primary A	CheckpointA
Primary A	LogFileA
Primary A	Health Monitor
Primary B	CheckpointB
Primary B	LogFileB
Primary B	Health Monitor
CheckpointA	Primary A
LogFileA	Primary A
CheckpointB	Primary B
LogFileB	Primary B
Health Monitor	Proxy
Query Client	Synchronous Communica-tion

Fuente: A Practical Example of Appliying Attribute-Driven Design (ADD) Ver. 2.0 SEI – Carnegie Mellon. William G. wood

En este paso se le asignan responsabilidades[2] a los elementos instanciados. Al finalizar este paso, cada elemento funcional asociado a un elemento padre debe ser representado por medio de responsabilidades dentro de los elementos hijo.

Se puede considerar realizar casos de uso que ayuden a identificar nuevas responsabilidades o nuevos tipos de elementos.

Otra tarea que se realiza en este paso es crear instancias de elementos para cubrir responsabilidades de atributos de calidad asignadas a un elemento o para lograr otro atributo de calidad de requerimientos.

Para analizar y documentar las decisiones de diseño que se toman en este paso se elaboran las siguientes vistas:

- Vistas de módulos
- Vistas de deployment
- Vistas de componentes y conectores.

[2] La responsabilidad de un elemento es la funcionalidad, dato o información que debe proporcionar.

Las vistas de módulo son usadas para documentar las propiedades que no son de tiempo de ejecución de un sistema; por ejemplo, modificability.

Las vistas de asignación son usadas para analizar las relaciones entre los elementos de software y los que no lo son. Por ejemplo, cómo los elementos de software serán asignados a los elementos de hardware.

Las vistas de componentes y conectores son usadas para documentar comportamientos de tiempo de ejecución y propiedades de un sistema. Por ejemplo, cómo interactuarán los elementos con otros en tiempo de ejecución para encontrarse con varios requerimientos y qué características de desempeño deberían mostrar esos elementos.

Las decisiones de diseño que se analizan en general para elaborar las vistas mencionadas anteriormente son las siguientes:

- Cómo serán instanciados cada tipo de elementos y qué propiedades individuales y relaciones estructurales tendrán.
- Qué elementos computacionales serán usados para soportar las diferentes categorías que usa el sistema.
- Qué elementos soportarán los mayores modos de operación.
- Cómo han sido satisfechos los atributos de calidad de los requerimientos dentro de la infraestructura y aplicaciones.
- Cómo es dividida y asignada la funcionalidad a elementos de software, incluyendo cómo es asignada la funcionalidad cruzando la infraestructura y aplicaciones.
- Cómo son mapeados los elementos de software con otros.
- Comunicación entre los diferentes elementos (elementos internos de software y entidades externas).
- Elementos internos de software y recursos del sistema.
- Dependencias entre los elementos internos de software.
- Cómo son usados los mecanismos de abstracción.
- Cuántos elementos del sistema se conocen a la fecha.
- Qué modelo será empleado (proceso / hilos).
- Cómo serán direccionados los atributos de calidad de los requerimientos.

6. Definir interfaces para elementos instanciados.

En este paso se definen los servicios y propiedades requeridas y otorgadas por los elementos de software en el diseño.

Una interfaz incluye alguno de los siguientes elementos:

- Sintaxis de operación
- Semántica de operación (describe pre y post condiciones, restricciones)
- Información intercambiada (eventos señalados, datos globales)
- Atributos de calidad de los requerimientos de elementos individuales u operaciones
- Manejo de errores

En este paso se realizan los siguientes tres subpasos:

- Utilizar los requerimientos funcionales que involucran a los elementos instanciados en el paso 5.

- Observar cualquier operación que es producida por algún elemento y consumida por otro. Considerar las interfaces desde la perspectiva de diferentes vistas. Por ejemplo, una vista de módulo permitirá analizar el flujo de información.

- Registrar lo encontrado en la documentación de interfaz para cada elemento.

Algunas de las decisiones de diseño que se hacen en este paso involucrarán varios puntos como los siguientes:

- Las interfaces externas para el sistema.
- Las interfaces entre particiones de alto nivel del sistema.
- Las interfaces entre aplicaciones dentro de particiones de alto nivel del sistema.
- Las interfaces para la infraestructura.

7. Verificar y refinar los requerimientos y aplicar las restricciones a los elementos instanciados.

En este paso se realizan los siguiente subpasos:

- Verificar que todos los requerimientos funcionales, requerimientos de atributos de calidad y decisiones de diseño asignadas al elemento padre se han asignado a uno o más elementos hijo en la descomposición.

- Trasladar cualquier responsabilidad que fue asignada al elemento hijo dentro de los requerimientos funcionales para los elementos individuales.

- Refinar los requerimientos de atributos de calidad para un elemento individual hijo tanto como sea necesario. Los escenarios de calidad son refinados.

- Si un escenario de calidad no es satisfecho con la actual descomposición, se debe valorar la importancia de dicho escenario para reconsiderar su descomposición.

En este paso no se realizan decisiones de diseño.

8. Repetir del paso 2 al 7 para los siguientes elementos del sistema en que se desee descomponer.

Una vez que se han completado los pasos del 1 al 7, se tiene una descomposición de elementos padre en los elementos hijo. Cada elemento hijo es una colección de responsabilidades, y tiene una descripción de interfaz, requerimientos funcionales, requerimientos de atributos de calidad y decisiones de diseño. Ahora se puede regresar al proceso de descomposición en el paso 2 donde se selecciona al siguiente elemento a descomponer.

El proceso del método ADD termina cuando ya no existen elementos del sistema a descomponer en el paso 2.

2.3 Aplicaciones del Método ADD

De acuerdo al reporte técnico "Un ejemplo práctico aplicando Attribute Driven Design ADD"[3], en ocasiones los pasos no son ejecutados de manera secuencial como se describió en la sección 2.2.

En dicho reporte se describe que al realizar una primera iteración con los pasos del 1 al 7, se obtiene una primera lista de elementos identificados que compondrán el sistema y se realiza una vista para representarlos.

Si con esta arquitectura inicial no se cubren por completo los requerimientos funcionales, las decisiones de diseño o los atributos de calidad de los requerimientos, se realiza una segunda iteración, en la cual ya no será necesario ejecutar el primer paso, ya que desde la primera iteración se conocen los datos de entrada que se están utilizando en el método, por lo que el proceso inicia en el paso 2 del método ADD.

[3] A Practical Example of Applying Attribute-Driven Design (ADD), Version 2.0
Technical Report, CMU/SEI 2007-TR-005
ESC-TR-2007-005

3 Estado del Arte

3.2 Arquitectura de software en Software Product Line (SPL)

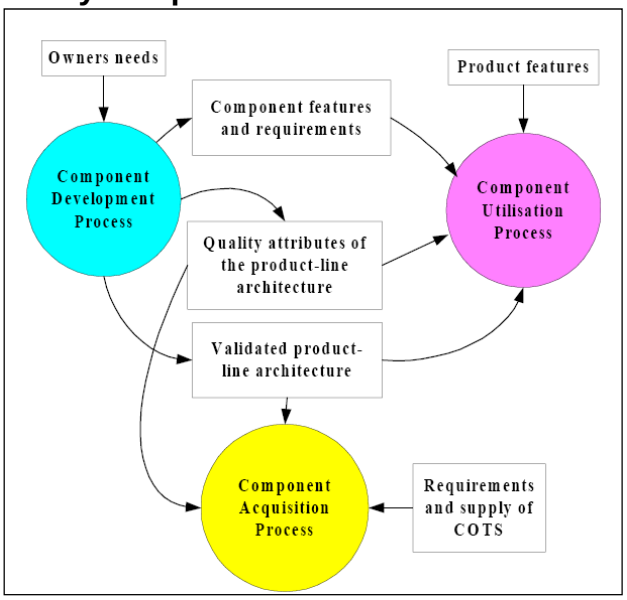
Un problema que ha enfrentado el análisis de SPL es cómo aprovechar los conceptos arquitectónicos para cumplir con los atributos de calidad de una forma sistemática. En la Figura 10 se ilustra la relación que existe entre la arquitectura de línea de productos, los atributos de calidad y los componentes de software.

En SPL los arquitectos de software construyen una arquitectura flexible a los cambios que se puedan presentar, para lo cual crean los mecanismos necesarios para soportar dichos cambios.

La diferencia de la arquitectura de SPL y la de un producto único es que además de las prácticas esenciales tales como modelado de la arquitectura, análisis e implementación de acuerdo a la especificación, incluye las áreas prácticas para obtener una arquitectura de software genérica y adaptable, así como el diseño y desarrollo de componentes reutilizables, además de ligar la arquitectura con los requerimientos comunes y variables.

La arquitectura de un software Product Line es determinante en la construcción de una gran variedad de versiones de productos basados en la línea de productos (ver Figura 11 Definición de Product Line). Lo anterior explícitamente incluye el concepto de variación.

Figura 10. Arquitectura Product Line, atributos de calidad y componentes de software

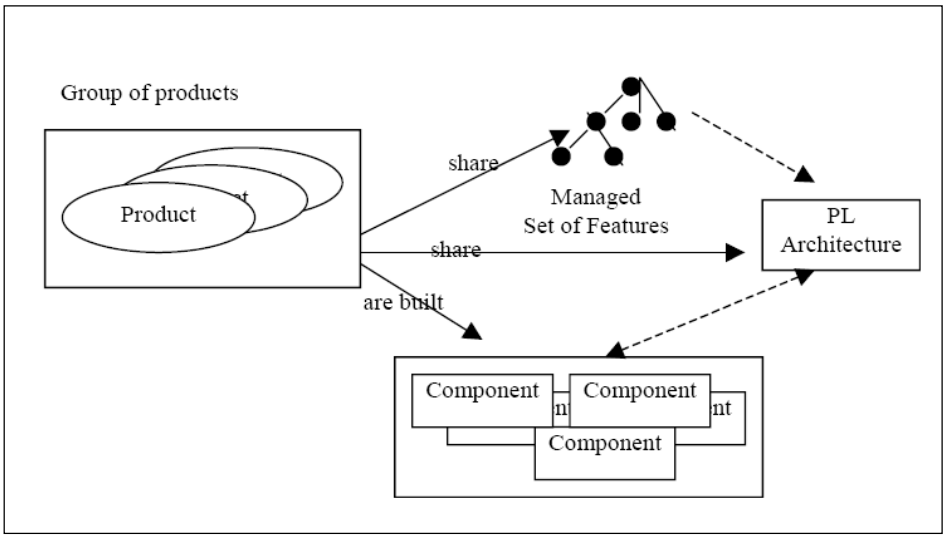


Fuente: A Strategy for analysing product line software architectures
Liliana Dobrica & Eila Niemelä, Technical Reserch Centre of Finland

Las variaciones se pueden localizar en las restricciones de un producto en particular, estas variaciones manifiestan diferentes atributos de calidad.

Cada producto en particular, tiene su propia arquitectura, la cual será una instancia de la arquitectura de la línea de productos, tomando en cuenta los mecanismos de variación.

Figura 11. Definición de Product Line



Fuente: A Strategy for analysing product line software architectures
Liliana Dobrica & Eila Niemelä, Technical Reserch Centre of Finland

En lo que se refiere al diseño de la arquitectura de software para línea de productos de software, el SEI contempla tres enfoques:

- Scoping (determinar el alcance).
- Levantamiento de requerimientos para línea de productos y para productos individuales.
- Diseño de arquitectura, para línea de productos y para productos individuales.

Las diferentes vistas de la arquitectura de línea de productos deben mostrar las diferentes variaciones que son posibles implementar describiendo los mecanismos de variación, los cuales deben explicar cómo serán ejercitados para crear un producto específico.

La variación puede darse en una función en particular, en datos específicos, en el control de flujo, en la tecnología, en las metas de calidad, en el ambiente donde operará el sistema, ente otros.

Para relacionar la variación que existe en la arquitectura con las características de los requerimientos y poder documentarla, se utiliza el concepto de puntos de variación. Los puntos de variación contienen la información necesaria para construir la arquitectura requerida por la especificación del producto.

3.1 Tendencias de la arquitectura en Software Product Line

Enfoques recientes indican que se están realizando investigaciones con el objetivo de lograr que la arquitectura y sus componentes sean desarrollados para cumplir con todos los requerimientos que se pueden obtener de todos los miembros de la línea de productos, incluso antes de desarrollar el primer producto.

Aunado a esto, las tendencias de investigación, muestran que el driver de SPL apunta a enfocarse en posibles cambios en los requerimientos de miembros ya existentes en la línea de productos y además de agregar nuevos productos.

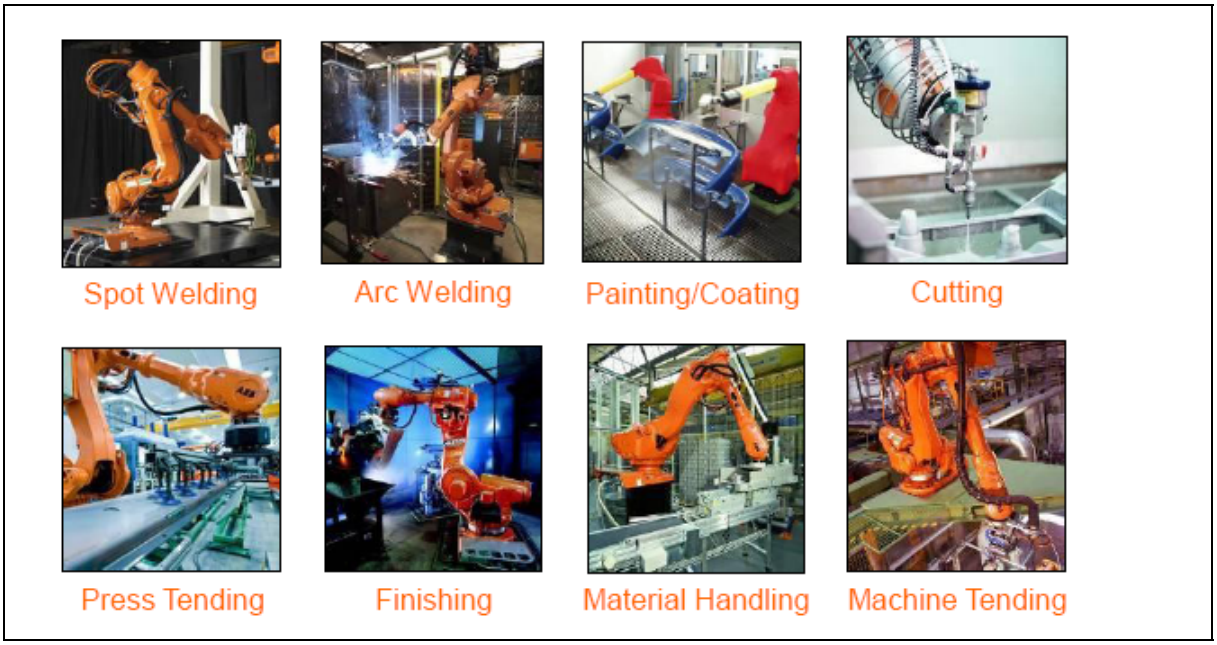
3.2 Aplicaciones de ADD en Software Product Line

3.2.1 Caso de Estudio, Bottom-up Software Product Line Design

ABB Robotics Software

Este sistema maneja aplicaciones de software para robots en una planta constructora de automóviles, dichas aplicaciones forman parte de más de 10 diferentes dominios (ver Figura 12), tales como soldadura, pintado, recubrimiento, manejo de material, prensado, cortado y terminado entre otros.

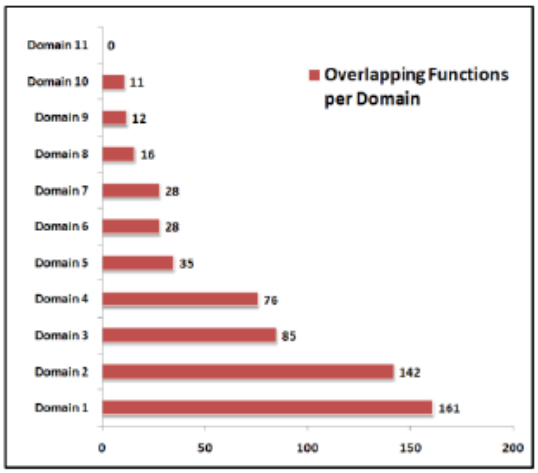
Figura 12. Dominios del sistema ABB Robotics Software



Fuente: Button-up software product line design
ABB Corporate Research Germany

Las aplicaciones tienen un tamaño en LOC's que van desde 1 a 2500 KLOC's. Existe una gran cantidad (ver Figura 13) de funciones con características similares en cada uno de los dominios del sistema, por ejemplo: monitoreo de operación, manejo de alarma y eventos, planeación de producción.

Figura 13. Funciones con características similares por dominio



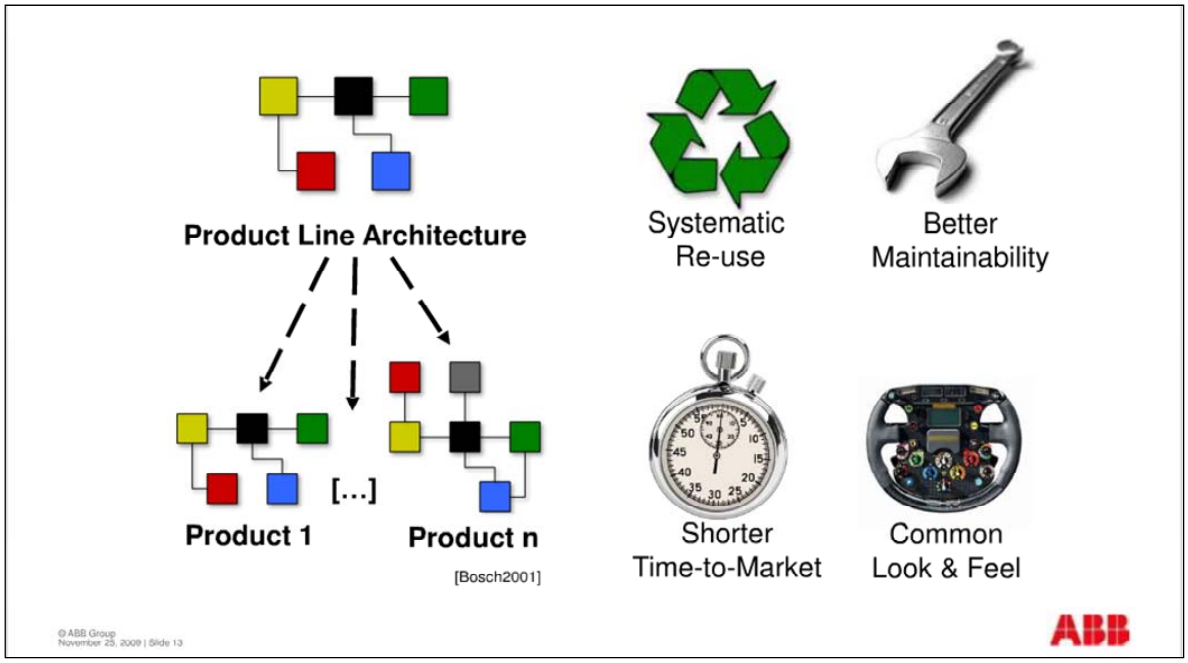
Fuente: Button-up software product line design
ABB Corporate Research Germany

El caso de estudio plantea obtener una Arquitectura de Línea de Productos (PLA) a partir del sistema ya en operación denominado ABB Robotics Software, con el objetivo de que dicha arquitectura maneje el reúso de manera sistemática y que a su vez permita reducir los costos de mantenimiento.

Las restricciones planteadas por la empresa (ilustradas en la Figura 14) para que se diseñe la arquitectura de línea de productos son:

- Que se mantenga la funcionalidad actual del sistema.
- Que se permita agregar más usuarios (distribuidos).
- Que tenga un look and feel común.
- Que sea posible realizar la transición a SPL de manera iterativa.

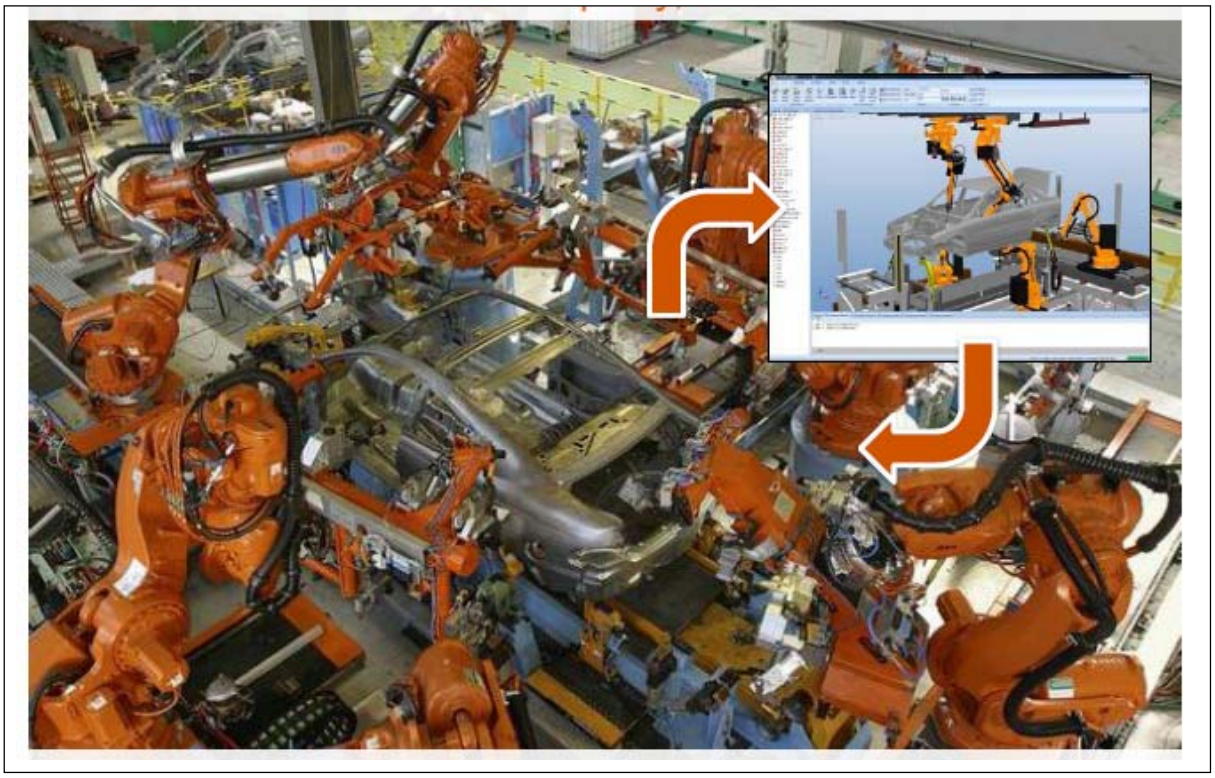
Figura 14. Diseño Product Line, Requerimientos y Metas



Fuente: Button-up software product line design
ABB Corporate Research Germany

3.2.1.1 Diseño de la arquitectura en SPL

Figura 15. Vista de la línea de producción y módulo de sistema



Fuente: Button-up software product line design
ABB Corporate Research Germany

Las primeras actividades que se realizaron para iniciar con el diseño de la arquitectura fueron:

- Analizar la documentación de la arquitectura actual.
- Realizar entrevistas exhaustivas con los stakeholders.
- Realizar inspecciones manuales y automatizadas de código.

Enseguida se definió el enfoque del diseño de la arquitectura.

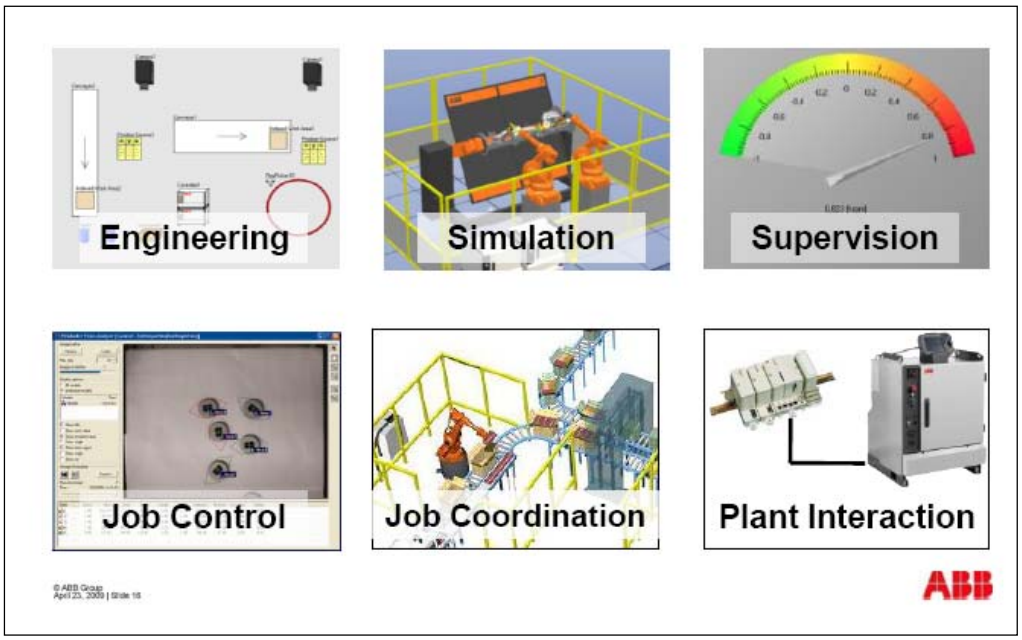
a. Análisis de requerimientos

- Tipos de drivers arquitectónicos:
  - Composability (reutilización y calidad)
  - Maintainability (menor costo en situaciones distribuidas)
  - Performance.

b. Ingeniería de dominio:

- Unificación de conceptos subyacentes.
- Identificar funcionalidad común (ver Figura 16).
- Definición de componentes reutilizables.

Figura 16. Ingeniería de Dominio, Funcionalidades comunes

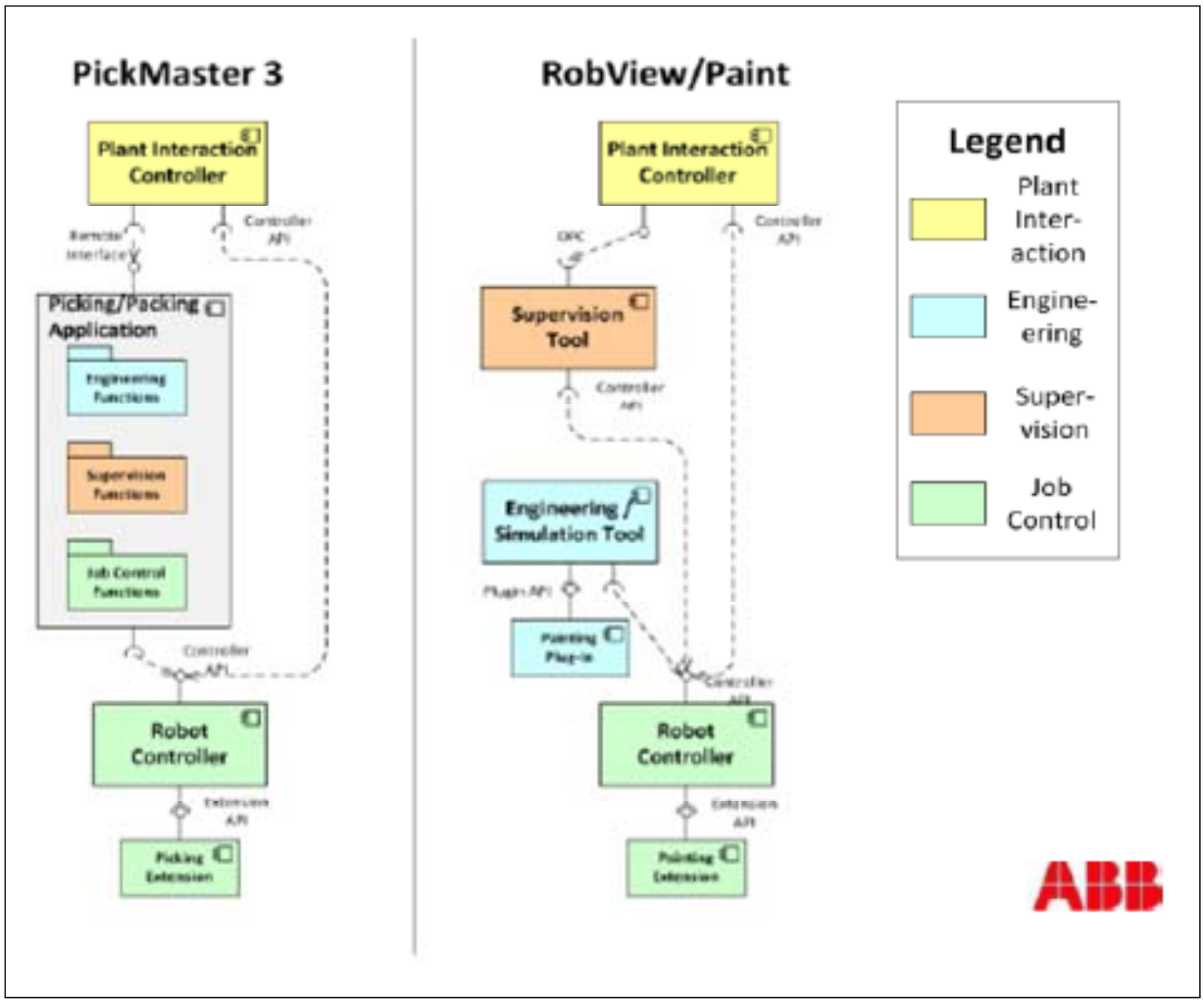


Fuente: Button-up software product line design
ABB Corporate Research Germany

c. Aplicando el método ADD

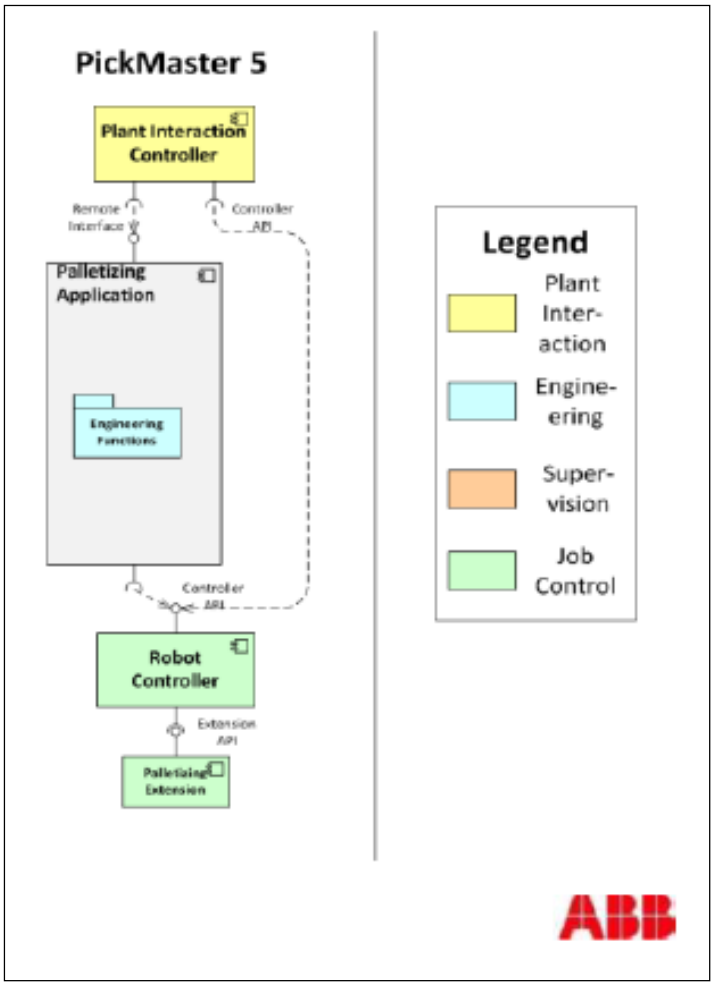
- En esta etapa se mapeó la funcionalidad de los componentes con aplicaciones existentes para satisfacer los drivers arquitectónicos y se realizó un mapeo de la funcionalidad de componentes que formaban parte de la arquitectura anterior (Figura 17 y Figura 18).

Figura 17. Mapeo de funcionalidad de componentes, arquitectura anterior (parte 1)



Fuente: Button-up software product line design
ABB Corporate Research Germany

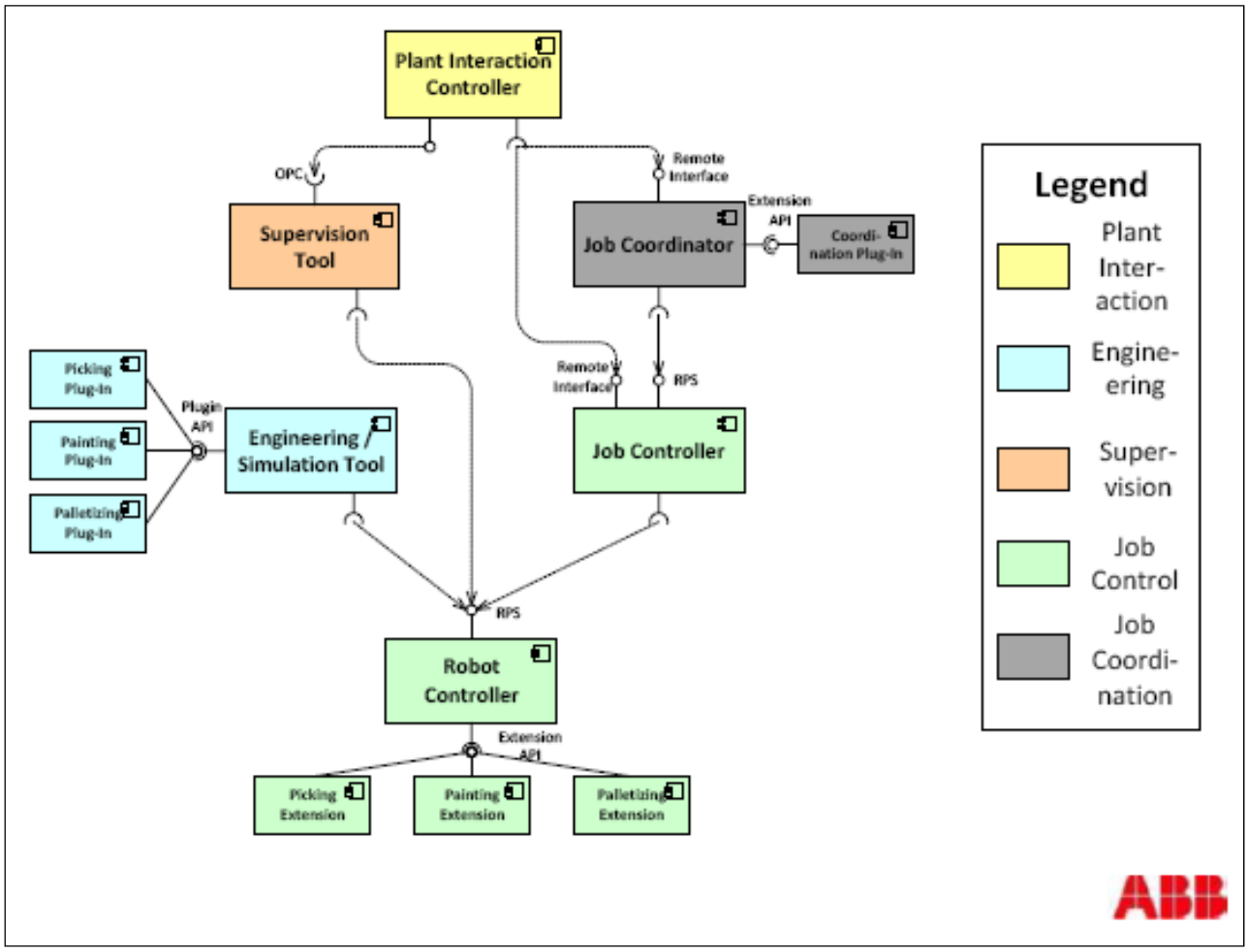
Figura 18. Mapeo de funcionalidad de componentes, arquitectura anterior (parte 2)



Fuente: Button-up software product line design
ABB Corporate Research Germany

- Se realizaron de dos o tres iteraciones de acuerdo a la complejidad y prioridad de los elementos a descomponer.

Figura 19. Nuevo diseño LPA (vista de componentes y conectores)

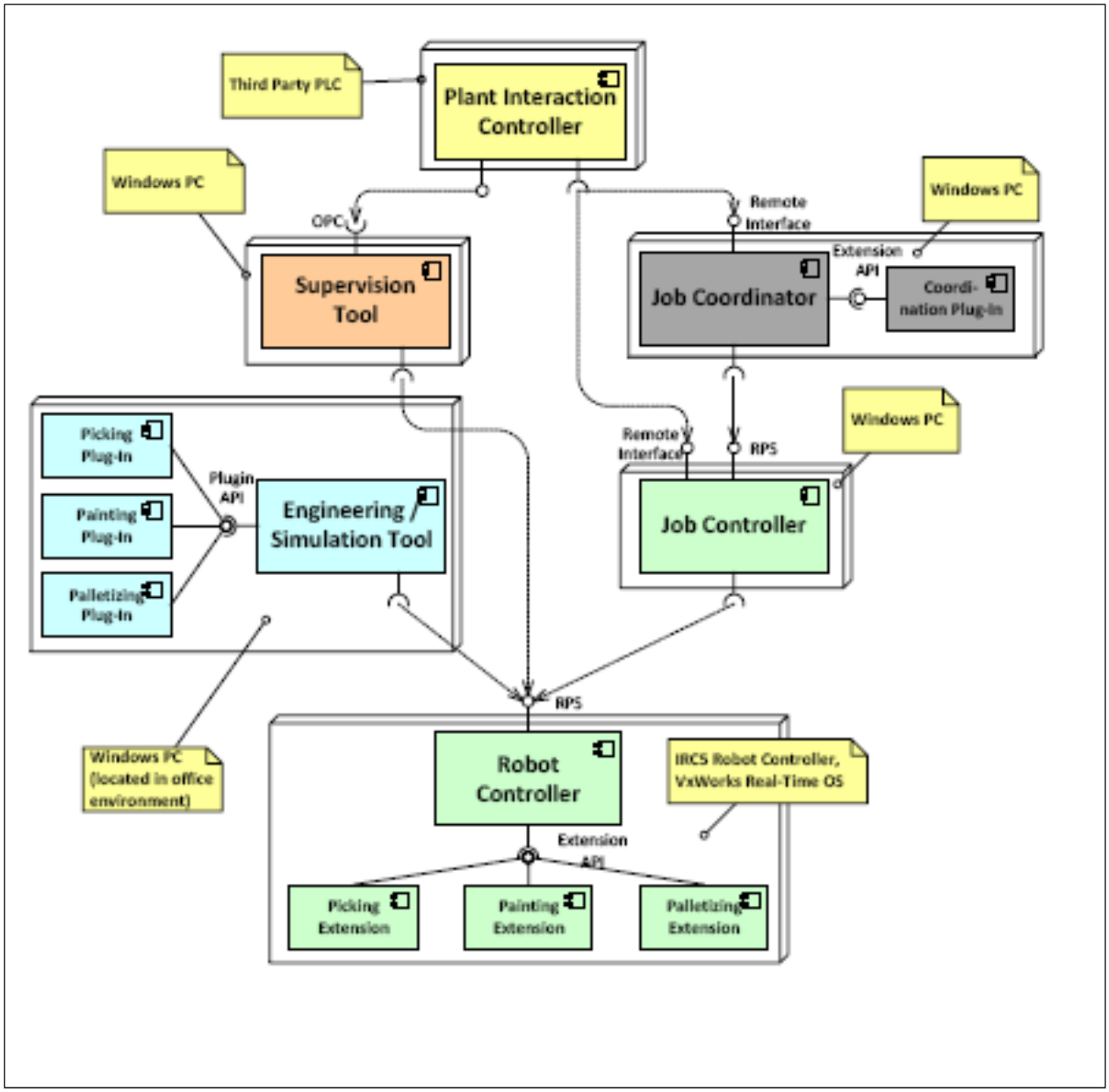


Fuente: Button-up software product line design
ABB Corporate Research Germany

Una de las partes importantes de esta etapa fue mantener una firme coordinación con los arquitectos líder de los productos individuales para con ello agilizar el proceso de diseño y asegurar una correcta adaptación a una arquitectura de línea de productos.

La Figura 19 muestra la vista de componentes y conectores, mientras que la Figura 20 muestra una vista de de la arquitectura resultante después de aplicar el método ADD.

Figura 20. Diseño Product Line, Vista Deployment (despliegue)



Fuente: Button-up software product line design
ABB Corporate Research Germany

3.2.1.2 Resultados de aplicar ADD al ABB Robotics Software.

- Usar el método iterativo ADD fue esencial para establecer la arquitectura de línea de productos.
- Cuando se tienen múltiples stakeholders, el propósito de la arquitectura necesita más argumentación por parte de estos y un enfoque más iterativo.
- Si el enlace es con el arquitecto de mayor experiencia y jerarquía se puede agilizar el proceso de diseño y adopción a una arquitectura de línea de productos.

3.2.1.3 Trabajos futuros a realizar en el ABB Robotics Software.

- Iniciar la planeación para migrar las actividades y realizar el diseño detallado.
- Ejecutar la transformación de la arquitectura de los productos independientes utilizando ADD.

3.2.2 Otras aplicaciones del método ADD en Software Product Line.

En el reporte técnico Introducing Product Lines in Small Embedded Systems, de Roberto Bosh Corporation, se describe la aplicación del método ADD mediante la técnica bottom-up al sistema Sunroof que migra al paradigma de SPL con el objetivo de reducir los costos de software.

El Sunroof es un sistema automotriz que se encarga entre otras tareas de reducir el ruido a una cierta velocidad, un arranque y frenado suave y además usa interfaces locales y remotas para obtener información de las condiciones del clima.

El sistema usa 32KROM y de 1KRAM, lo cual es obviamente una restricción de memoria. Si se aplicaran cambios menores a procesador o memoria implicarían un gran costo, por lo que se pensó en buscar la solución en el software y en otras herramientas, que hagan que la empresa sea más competitiva en el mercado.

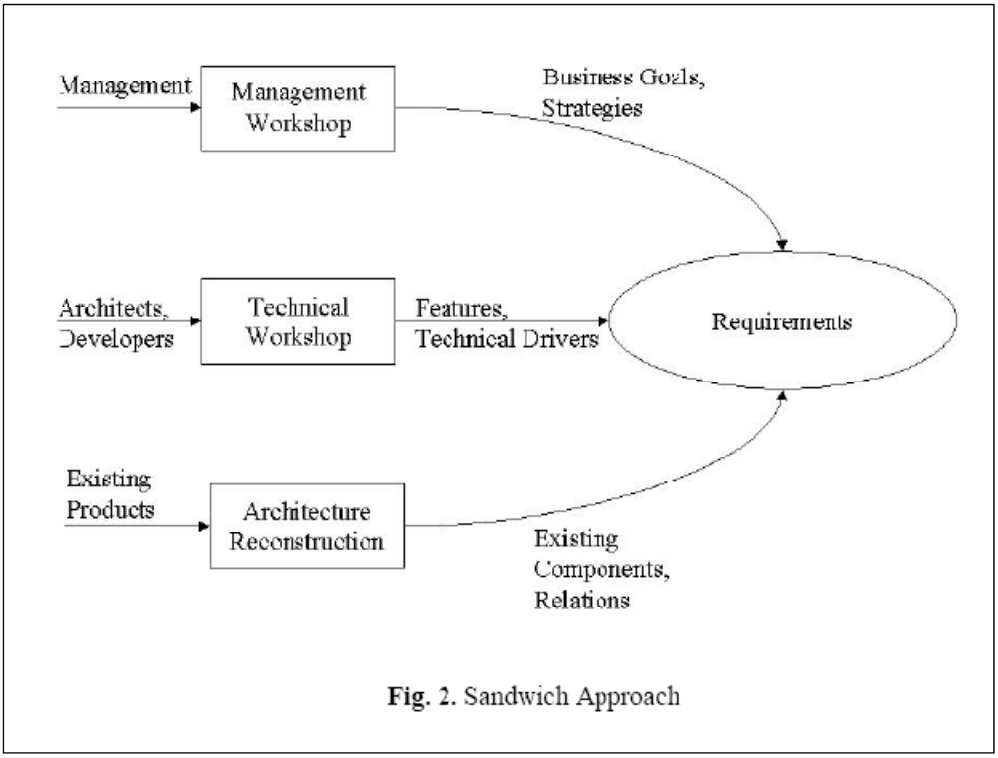
Los estudios que realizaron arrojaron las necesidades de realizar dos cambios: Aceptar una nueva tecnología en la organización y diseñar la arquitectura con enfoque de línea de productos.

El proyecto consistió de varias actividades (ver Figura 21) dentro de las cuales destacan:

- Investigación para analizar los productos existentes y licitación de requerimientos para SPL del sistema Sunroof mediante la técnica top-down.

- Reconstrucción de la arquitectura del sistema sunroof, mediante la técnica bottom-up

Figura 21. Enfoque de licitación de requerimientos y reconstrucción de arquitectura



Fuente: Introducing Product Lines in Small Embedded Systems, Roberto Bosh Corporation

La implementación de SPL en el sistema Sunroof arrojó como resultados un 20% de incremento en la eficiencia por año en el esfuerzo de la ingeniería de software con un incremento en la calidad y un 70% de reúso dentro de la familia de productos.

4 Conclusiones

ADD disminuye el riesgo de la dependencia que se ha tenido sobre la creatividad o experiencia que pueda tener el arquitecto para construir un diseño que satisfaga los requerimientos. Sin embargo no pretende reemplazar la experiencia de los arquitectos, al contrario, busca aprovecharla para obtener diseños arquitectónicos de calidad.

El diseño de arquitecturas de software es la base para crear sistemas seguros y confiables, por lo que se considera como una parte esencial del proceso de desarrollo de un proyecto de software.

Los atributos de calidad de un sistema muchas de las veces no son tomados en cuenta a profundidad desde el diseño de la arquitectura. Si la arquitectura, no cubre desde un principio dichos atributos, no se puede esperar lograrlos después durante el desarrollo del ciclo de vida.

Aunque los requerimientos funcionales son la esencia de lo que el sistema realizará, los requerimientos de calidad son los que dan forma a la arquitectura del sistema, por esta razón ADD asume que son los atributos de calidad los que deben conducir el diseño conceptual de la arquitectura de software.

ADD sólo toma en cuenta los drivers arquitectónicos de calidad como base del diseño, no toma en cuenta todos los requerimientos. Sin embargo los requerimientos menos importantes no se dejan del todo fuera del diseño, sino que son cubiertos dentro de las restricciones de los requerimientos más importantes durante la aplicación del método.

Con el método ADD no se obtiene la arquitectura definitiva del sistema sino una arquitectura conceptual que se utiliza de base para completar la fase de diseño.

En Software Product Line las actividades de levantamiento de requerimientos están incluidas en el Product Line Analysis, mientras que las actividades de diseño arquitectónico tienden a utilizar cada vez más el Método Attribute Driven Design (ADD).

La definición de la arquitectura, forma parte de las prácticas de Product Line en donde se considera como una parte crítica para obtener un producto de calidad.

5 Glosario

Arquitectura de Software
Es la estructura o estructuras del propio sistema, que comprende los elementos, las propiedades visibles de esos elementos, y las relaciones entre ellos. Las propiedades visibles de los elementos es lo que se asume pueden hacer, tal como proveer un servicio, características de desempeño, manejo por default, uso y compartición de recursos, entre otros.

Atributo de calidad
Es una propiedad de un producto de trabajo del cual su calidad será juzgada por los stakeholders. Los requerimientos de atributos de calidad tales como performance, segurity, modificability y usability tienen un significado que influye sobre la arquitectura de software de un sistema.

Atributos primitivos
Son patrones arquitectónicos que son primordiales para el logro de los atributos de calidad. Colección de componentes y conectores que colaboran para lograr alguna meta de atributo de calidad.

Commonality
Aplicaciones en SPL que tienen exactamente la misma forma.

Componente
Es el principal elemento computacional y de datos que se ejecuta en el sistema.

Composability
En un sistema, es un principio de diseño que se ocupa de las interrelaciones entre componentes. Un sistema con alta composability contiene componentes que pueden ser utilizados en distintas combinaciones para satisfacer requerimientos específicos de un usuario.

Conector
En tiempo de ejecución es el camino de interacción entre dos o más componentes.

Core assets
Son los artefactos reusables y recursos que forma la base para Software Product Line. Los Core Assets incluyen a menudo, pero no es una limitación, la arquitectura, los componentes de software reusables, modelos de dominio, declaración de requerimientos, documentación, especificación, modelos de performance, planeación, presupuestos, planes de pruebas, casos de prueba, planes de trabajo y descripción de procesos. La arquitectura es lo más importante entre la colección de Core Assets.

Dominio
Un dominio es un cuerpo especializado de conocimiento, un área de expertise, o una colección de funcionalidad relacionada.

Drivers Arquitectónicos
Es la combinación de requerimientos funcionales, atributos de calidad y requerimientos de negocio que son muy significativos para la arquitectura o para un elemento en particular que se esté considerando de la misma.

Escenarios
Describen la interacción de un sistema con respecto a algún atributo de calidad. Normalmente el enfoque es hacia tres tipos de escenarios: casos de uso (usos anticipados del sistema), cambios anticipados del sistema y exploratorio (anticipar estresar al sistema).

Estilos Arquitectónicos
Es una especialización de elementos y su tipo de relación, junto con un conjunto de restricciones sobre cómo puede ser usados. Ver patrones arquitectónicos.

Interface de componentes
Las interfaces se componen de las firmas funcionales que se encuentran en las cabeceras de los archivos. Las firmas nombran los programas y especifican los parámetros y sus tipos, pero no indican la semántica de las operaciones que se realizan. En caso de usar componentes ya existentes se debe asegurar que sus interfaces están debidamente especificadas.

Patrones Arquitectónicos
Es un conjunto de decisiones de diseño que describe clases de arquitectura de software. Contiene la descripción de elementos y su tipo de relación, junto con un conjunto de restricciones sobre cómo pueden ser usados. Ver estilos arquitectónicos.

Restricción
Es un requerimiento para el cual las decisiones de diseño son pre especificadas.

Software Product Line
Es un conjunto de sistema de software que comparte un conjunto común de características que satisfacen las necesidades específicas de un particular segmento de mercado o misión y que son desarrolladas desde un conjunto común de core assets reusables en una forma prescrita. Implementa el uso sistemático de core assets para ensamblar, instanciar o generar múltiples productos.

Stakeholder
Es la persona quien tiene un interés e injerencia directa en la arquitectura de un sistema.

Táctica
Es una decisión de diseño que está denotada por medio de estímulo/respuesta de un atributo de calidad. La táctica dice lo que se quiere hacer para afectar la métrica de la respuesta de un atributo de calidad. Las tácticas son independientes de un sistema específico.

Variability
Concepto utilizado cuando varias aplicaciones son derivadas de una línea de productos.

Vista
Es una representación de un conjunto de elementos y la relación entre ellos. Una vista muestra una perspectiva particular del sistema la cual puede ser una parte o un sistema completo.

6 Referencias

Attribute – Driven Design (ADD) Versión 2.0
SEI – Carnegie Mellon
November, 2006
Technical Report
CMU/SEI-2006-TR-023
ESC-TR-2006-023

Button-up Software Product Line design
Ronald Weiss, Jens Doppelhamer, Heiko Koziolek
Industrial Software System
ABB Corporate Research Germany
April, 2009

A Practical Example of Appliying Attribute-Driven Design (ADD) Ver. 2.0
SEI – Carnegie Mellon. William G. wood
February, 2007
Technical Report
CMU/SEI-2007-TR-005
ESC-TR-2007-0005

A Framework for Software Product Line Practice, Version 5.0
SEI, Carnegie Mellon
Linda M. Northrop, Paul C. Clements
with Felix Bachmann, John Bergey, Gary Chastek, Sholom Cohen, Patrick Donohoe, Lawrence Jones, Robert Krut, Reed Little, John McGregor, Liam O'Brien
July, 2007
Product Line Practice Initiative

Quality Attribute Design Primitives and the Attribute Driven Design Method
Len Bass, Mark Klein, and Felix Bachmann
Software Engineering Institute
Carnegie Mellon University

Managing Variability in Software Architectures
Len Bass, Felix Bachmann
Carnegie Bosch Institute
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pa 15213, USA

Software Architecture and Product Quality
SEI – Carnegie Mellon
Presentation for the Boston SPIN
May, 2003
Linda Northrop
Director, Product Line System

A strategy for analysing Product Line software architectures
Liliana Dobrica & Eila Niemelä
Techinical research centre of Finland
ESPOO 2000

Current Best Practices in Software Architecture
Paul Clements
Software Engineering Institute
Carnegie Mellon University

Introducing Product Lines in Small Embedded Systems
Christoph Stoermer, Markus Roeddiger
Robert Bosch Corporation

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS, A.C.

BIBLIOTECA

AUTORIZACION
PUBLICACION EN FORMATO ELECTRONICO DE TESIS

El que suscribe
Autor(s) de la tesis: Heriberto Ortega Domínguez

Título de la tesis: Attribute Driven Design y Software Product Line

Institución y Lugar: CIMAT Unidad Zacatecas

Grado Académico: Licenciatura ( ) Maestría (X) Doctorado ( ) Otro ( )

Año de presentación: 2010

Área de Especialidad: Ingeniería de Software

Director(es) de Tesis: Dr. Cuauhtémoc Lemus Olalde

Correo electrónico: hortega@cimat.mx

Domicilio: Calle de Ingeniería No. 22 Fracc. SPAUAZ, Guadalupe, Zac.

Palabra(s) Clave(s): Atributos de calidad, drivers arquitectónicos, patrones arquitectónicos

Por medio del presente documento autorizo en forma gratuita a que la Tesis arriba citada sea divulgada y reproducida para publicarla mediante almacenamiento electrónico que permita acceso al público a leerla y conocerla visualmente, así como a comunicarla públicamente en la Página WEB del CIMAT.

La vigencia de la presente autorización es por un periodo de 3 años a partir de la firma de presente instrumento, quedando en el entendido de que dicho plazo podrá prorrogar automáticamente por periodos iguales, si durante dicho tiempo no se revoca la autorización por escrito con acuse de recibo de parte de alguna autoridad del CIMAT

La única contraprestación que condiciona la presente autorización es la del reconocimiento del nombre del autor en la publicación que se haga de la misma.

Atentamente

Heriberto Ortega Domínguez

Nombre y firma del tesista

CALLE JALISCO S/N MINERAL DE VALENCIANA APDO. POSTAL 402
C.P. 36240 GUANAJUATO, GTO., MÉXICO
TELÉFONOS (473) 732 7155, (473) 735 0300 EXT. 49609 FAX. (473) 732 5749
E-mail: biblioteca@cimat.mx