Arquitectura Base en una Línea de Productos de Software

M. Cecilia Bastarrica
cecilia@dcc.uchile.cl
Universidad de Chile
Departamento de Ciencias de la Computación
CEIS, Centro Experimental de Ingeniería de Software
Av. Blanco Encalada 2120, Santiago, Chile

Resumen

La definición de la arquitectura base es un paso esencial en el desarrollo de una línea de productos. Sin embargo, no existe ninguna forma estándar de hacer esta definición. En este artículo proponemos la definición de la arquitectura base en dos niveles: un nivel de interfaz que establece los tipos de componentes con su semántica y sus interfaces, y un nivel de implementación, consistente con el anterior, pero que incluye además las diferentes implementaciones posibles para cada tipo de componente. La arquitectura particular de cualquier producto de la línea será la instanciación de parte de la defincón al nivel de implementación. La arquitectura base definida en dos niveles sirve para guiar el desarrollo de productos planeados así como también ayuda a construir productos aún no anticipados.

Palabras claves: Arquitectura de software, líneas de productos de software, especificaciones formales.

Abstract

Defining the base architecture is an essential step in the development of a software product line. However, there is no standard notation for defining the base architecture. In this paper, we present the definition of the base architecture in two levels: an interface and an implementation level. The interface level includes the component types of the system with their semantics and interactions. The implementation level must be consistent with the interface level, but it also defines all the available implementations for each component type in the interface level. The architecture of a particular product of the software product line is necessarily the instantiation of part of the implementation level of the base architecture. This definition in two levels helps developing planned products as well as building non anticipated products.

Keywords: Software architecture, software product lines, formal specifications.

1 Introducción

Muchos esfuerzos se han realizado para potenciar la productividad en el desarrollo de software, y tal como lo dice Brooks [6], algunos de los logros obtenidos han sido circunstanciales y otros esenciales. Al momento de escribir su ya clásico artículo, Brooks no incluyó la reutilización planificada de componentes de software porque esta práctica era entonces aún muy incipiente. Hoy en día existe una fuerte tendencia hacia la reutilización y esto está dando un salto cualitativo y cuantitativo en la productividad. Se reutiliza tanto el código como todos los otros artefactos producto del proceso de desarrollo: diseño, especificación de requisitos, manuales de usuario, planes de pruebas, procedimientos de instalación, etc. A estos artefactos se les llama comunmente activos esenciales o core assets.

Una nueva tendencia basada en la reutilización masiva y planificada de activos es el desarrollo de líneas de productos de software. Las líneas de productos de software son colecciones de productos que se construyen a partir de un conjunto común de activos y que satisfacen las necesidades de un segmento de mercado [19].

Uno de los principales activos esenciales en una línea de productos es su arquitectura base. Esta define las componentes que formarán todos los productos de la línea, así como su interrelación. La definición de esta arquitectura base desde una temprana etapa en el proceso de desarrollo guiará a diseñadores y programadores para que los productos desarrollados estén dentro del alcance definido para la línea de productos. Pero la elección de una estrategia de líneas de productos se debe generalmente a que se desea obtener no solamente una serie de productos planeados con un alto grado de reutilización, sino que además todos los activos que se desarrollen en el proceso de construcción puedan servir para potenciales desarrollos futuros de productos aún no identificados. Es por eso que la definición de la arquitectura base debe ser lo suficientemente precisa para guiar sin ambigüedades el desarrollo de los productos planeados, y también debe ser lo suficientemente flexible como para dar cabida a nuevos productos. Esta tensión entre las necesidades de precisión y flexibilidad es la que motiva nuestra investigación.

En este artículo proponemos la idea que la arquitectura base no debiera ser única, sino definida en dos niveles. Para eso usamos los dos primeros niveles del lenguaje de especificación de arquitecturas $\mathcal{I}^5$ [1, 2]: el nivel de interfaz y el nivel de implementación. En el nivel de interfaz se definen los tipos de componentes y su forma de interacción. El nivel de implementación debe ser un refinamiento del nivel de interfaz de modo que sea compatible con él, pero incluyendo las diferentes implementaciones posibles para cada uno de los tipos de componentes antes identificados. Cada uno de los productos de la línea de productos de software será una instanciación particular de la definición de la arquitectura a nivel de implementación, donde se incluye solamente un subconjunto de los componentes definidos en la arquitectura base.

Existen muchas formas de definir la arquitectura de un software, desde un diagrama informal de cajas y líneas hasta una formalización completa usando un ADL [15]. Nosotros proponemos definir la arquitectura base usando parte de $\mathcal{I}^5$ [1]. Definimos la arquitectura en dos niveles de abstracción diferentes: interfaz e implementación. También en [16] se usan varios niveles de abstracción para definir la arquitectura: funcionalidad de las componentes, interfaces, configuración y reglas del estilo de arquitectura. Nuestra propuesta se orienta más a la definición de alto nivel en el nivel de interfaz, que incluiría parte de estos cuatro niveles, y a la definición de una arquitectura más orientada a la reutilización directa de componentes, un nivel no abordado en el otro trabajo.

1.1 Ejemplo: Generación y Refinamiento de Mallas Geométricas

A lo largo del artículo ejemplificamos nuestras afirmaciones con la arquitectura base definida para una línea de productos para la generación y manejo de mallas geométricas. Se trata de una serie de productos que permiten, a partir de una malla de superficie de un objeto (malla de triángulos), mejorar esta malla de superficie, generar una malla de volumen (malla de tetraedros), mejorar esta malla de volumen y refinarla interactiva y/o adaptivamente. Cada producto de la línea contiene una funcionalidad parcial o podrá incluirlas todas con diferentes

implementaciones.

Los algoritmos de generación, manejo y refinamiento de mallas geométricas tridimensionales son de muy alta complejidad [12, 17]. Tener la capacidad de reutilizar las implementaciones de todos estos algoritmos incorporándolos a una línea de productos de software dará una oportunidad comercial a estos productos [7].

1.2 El Artículo

En la sección 2 hacemos una descripción general del desarrollo de líneas de productos de software y sus implicancias técnicas y organizacionales. En la sección 3 nos referimos al concepto de arquitectura de software y su particular aplicación a la definición de la arquitectura base de una línea de productos. Damos aquí una breve descripción del lenguaje de definición de arquitecturas $\mathcal{I}^5$ y definimos los dos niveles propuestos para la formalización de esta arquitectura. En la sección 4 establecemos la relación entre la arquitectura base y la arquitectura particular de cada uno de los productos de la línea, indicando las diferentes formas de usarla dependiendo si se trata de un producto planeado o uno nuevo. Finalmente, en la sección 5 exponemos algunas conclusiones y describimos nuestro trabajo en curso.

2 Líneas de Productos de Software

En general, las empresas desarrolladoras de software se especializan en tipos particulares de aplicaciones o ciertos nichos de mercado. Es así como de un desarrollo al siguiente existen muchas partes que son o podrían potencialmente ser reutilizadas, estos son sus activos. Sin embargo, la real magnitud de esta reutilización depende en gran medida de la generalidad y la anticipación del cambio que se haya usado en el desarrollo de los activos [11]. Desarrollar activos de software configurables, generales, portables, robustos, documentados y probados es claramente más caro que desarrollarlos específicamente para una única aplicación. Sin embargo, si estos activos son reutilizados un número de veces suficiente, estos costos se ven compensados. Se gana además en otros aspectos tales como confiabilidad en la calidad del nuevo producto ya que se integra de activos ya probados, la planificación y la administración del desarrollo es más fácil debido a que existen menos "partes" a desarrollar, y se gana también en oportunidad de comercialización ya que desarrollos grandes y complejos pueden hacerse en un plazo mucho menor. Los clientes también ganan en familiaridad con el nuevo software y la curva de aprendizaje se hace más corta, sobre todo si se reutilizan manuales de usuario, diseño de interfaces, y mecanismos de interacción.

Las líneas de productos de software (LPS) son un nuevo paradigma en el desarrollo de software [7, 19] que captura esta idea de reutilización de activos de software a gran escala. La premisa esencial es que no se puede hacer una reutilización efectiva si ésta no se ha planificado. Es así que para desarrollar líneas de productos de software se requiera de una estructura organizacional particular donde distintas personas con distintos perfiles profesionales se encarguen de desarrollar los activos (ingeniería del dominio), integrar los productos (ingeniería del producto) o tomar decisiones estratégicas (administración) dentro de la empresa, todo de una manera coordinada. La Figura 1 muestra esquemáticamente la relación entre estas áreas en una empresa de desarrollo de LPS.

2.1 Ingeniería del Dominio

La ingeniería del dominio se encarga del desarrollo, mantenimiento y custodia de los activos de la línea de productos, o sea todos aquellos elementos potencialmente reutilizables. Ejemplos de estos activos son: planes de producción, componentes de software, especificaciones de requisitos, diseño de algoritmos, arquitectura base, planes de pruebas y casos de prueba.

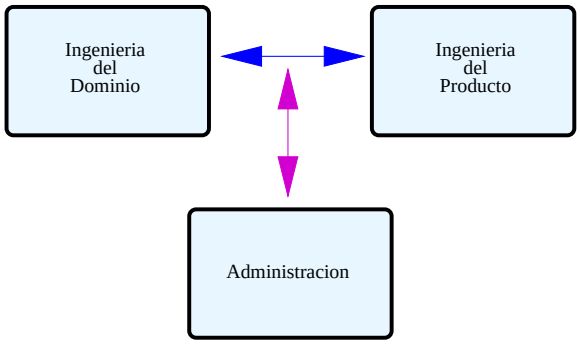

Figura 1: Organización de una Empresa de Líneas de Productos.

Dada la diversidad de los activos posibles, las personas que trabajen en la ingeniería del dominio serán especialistas: diseñadores, programadores expertos, ingenieros de pruebas, expertos en minería de activos [4]. En particular, los encargados del diseño y el mantenimiento de la arquitectura base deberán tener una visión general del alcance de la línea de productos de modo que la arquitectura sea lo suficientemente estricta como para ceñirse a este alcance y así limitar y guiar el desarrollo, pero también lo suficientemente flexible como para dar cabida a nuevos productos aún no identificados. El diseño de la arquitectura base es un asunto crítico para el éxito de la línea de productos.

2.2 Ingeniería del Producto

La ingeniería del producto o de la aplicación se encarga de identificar las necesidades de clientes potenciales, y de acuerdo con estas, analizar los activos disponibles, negociar ciertas variaciones a las peticiones de los clientes, determinar si es necesario el desarrollo de nuevos activos, integrar los productos y probarlos. Claramente, las personas encargadas de esta función deberán ser expertas en el desarrollo basado en componentes y en pruebas de software, sobre todo pruebas de caja negra y de integración [11].

Cada uno de los productos integrados deberá ceñirse a la arquitectura base predefinida para toda la línea, o bien negociar una modificación con la ingeniería del dominio y la administración. Si los nuevos productos usan la arquitectura base ya definida tendrán ciertas garantías de calidad, pero no siempre es fácil hacer que nuevos productos calcen armoniosamente con esta arquitectura. La definición apropiada de la arquitectura base ayuda a hacer más efectiva y controlada la tarea de la ingeniería del producto.

2.3 Administración

La administración es quien se encarga de mediar entre las otras dos actividades esenciales. Es la administración quien decide si vale la pena realizar un nuevo desarrollo o en qué dirección evolucionará la línea de productos.

Lo más frecuente es que un nuevo producto se ajuste a la arquitectura base, pero sea necesario implementar algunos de sus componentes; la administración deberá hacer en este caso un análisis esencialmente económico de la conveniencia del nuevo desarrollo. Es posible también que un nuevo producto pueda reutilizar una serie de componentes ya desarrollados pero no se ciña estrictamente a la arquitectura base; esto es un cambio mayor y riesgoso en la estrategia de la empresa y deberá ser cuidadosamente evaluado. Los productos que no sean compatibles con la arquitectura base y tampoco reutilicen muchos de los activos disponibles, están generalmente fuera del alcance de la LPS.

3 Arquitectura Base

La arquitectura de un software es la definición de las componentes que forman el sistema, la interrelación entre estas componentes, los patrones que guían su composición, y las restricciones de estos patrones [18].

Se llama arquitectura base de una LPS a la definición de una arquitectura general que describe todos los productos que caben dentro del alcance de la línea [19]. Es así que todos los productos de la LPS son instanciaciones parciales de esta arquitectura. Aquí vemos claramente que existe una relación estrecha entre la arquitectura base de la línea de productos y la arquitectura de un producto en particular: todas las componentes definidas para el producto deben ser parte de las componentes definidas dentro de la arquitectura base y deben interactuar de la forma allí indicada. Pero dado que los productos de la línea son similares pero esencialmente diferentes, la arquitectura de un producto no es idéntica a la arquitectura base.

3.1 $\mathcal{I}^5$: Un Lenguaje de Descripción de Arquitecturas

El lenguaje de descripción de arquitecturas $\mathcal{I}^5$ [1, 2], está formado por una serie cinco lenguajes que, en forma integrada, permiten definir la arquitectura de un sistema de software en cinco niveles de abstracción considerando aspectos tanto de hardware como de software. Es un lenguajes especialmente apropiado para la definición de la arquitectura de sistemas distribuidos. Sus cinco niveles de especificación son *Interfaz*, *Implementación*, *Integración*, *Instanciación* e *Instalación*, de ahí las cinco *I*. La figura 2 muestra los distintos niveles de especificación y la relación entre ellos.

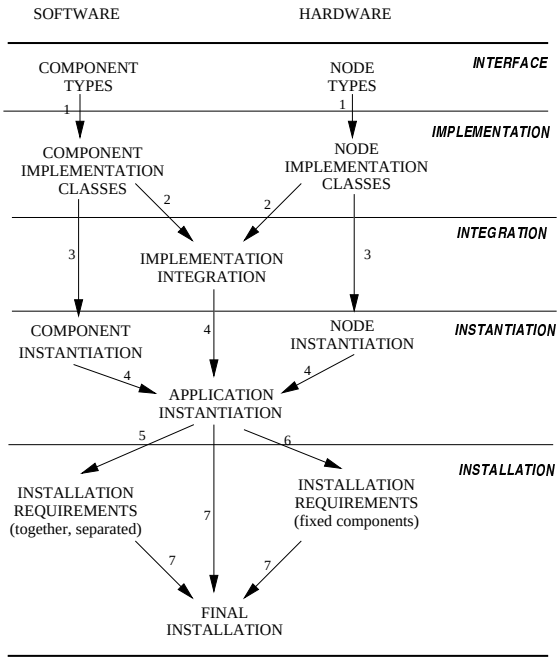

Figura 2: $\mathcal{I}^5$: Un Lenguaje de Definición de Arquitectura en Cinco Niveles.

A continuación describimos brevemente el uso de cada uno de los niveles de especificación en $\mathcal{I}^5$. Para la especificación gráfica de cada uno de los niveles usamos diagramas de componentes y de deployment de UML [5, 9].

Interfaz Incluye la definición de los diferentes *tipos* de componentes de la aplicación y *tipos* de nodos en la red,

así como sus dependencias y conexiones. La definición de tipos incluye tanto su interfaz como su semántica. Gráficamente se usan diagramas UML de componentes para especificar el nivel de interfaz del software y diagramas de deployment a nivel de tipos para especificar el hardware.

Implementación La *Implementación* se refiere a las clases que realizan los tipos de componentes y nodos identificados en el nivel de Interfaz.Las clases incluyen semántica e interfaz, al igual que los tipos, pero también tienen una implementación particular. Se usan también diagramas de componentes para especificar el nivel de *Implementación* del software, pero como las componentes que se incluyen son *clases* y no tipos de componentes, se las identifica con un estereotipo que indicamos con una caja de color. Un mecanismo similar se usa para especificar clases de nodos en el hardware.

Integración Este nivel establece los requisitos y restricciones de mapeo entre las clases de componentes y las clases de nodos que forman parte del sistema. Se define una relación *"soporta"* que indica que una instancia de una clase de componente puede ejecutar en una instancia de una clase particular de nodo. Las restricciones establecidas a este nivel limitan y dirigen la futura instalación del sistema.

Instanciación Las instancias concretas de componentes y nodos del sistema se especifican en el nivel de *Instanciación*. Solamente puede haber instancias de las clases de componentes y nodos identificados en el nivel de *Implementación* en el producto final. Para la especificación gráfica de este nivel se usan diagramas de deployment de UML de nivel de instancias incluyendo solamente componentes o solamente nodos para especificar la instanciación de software o el hardware, respectivamente.

Instalación En este nivel se define dónde precisamente será instalada cada instancia de componente en cada instancia de nodo. Obviamente solamente podrán instalarse instancias de componentes identificadas en el nivel de *Instanciación* sobre instancias de nodos también definidas en este nivel, y todo sujeto a las restricciones impuestas a nivel de *Integración*. Para la instalación se usan diagramas de deployment de UML a nivel de instancias integrando tanto componentes como nodos.

Para la definición de de arquitectura base de una línea de productos de software usaremos los niveles de *Interfaz* e *Implementación* para software. Para definir la arquitectura de un producto particular de la línea de productos usaremos el nivel de *Instanciación* de software. Describiremos cada uno de estos niveles de especificación en más detalles en las secciones 3.2, 3.3 y 4 proporcionando ejemplos de cada uno de ellos.

3.2 Definición de Arquitectura Base a Nivel de Interfaz

En nuestra línea de productos de generación y manejo de mallas geométricas hemos definido una arquitectura base usando el nivel de interfaz de $\mathcal{I}^5$. Esto se muestra en la figura 3.

La definición de las componentes se hace usando una restricción sobre los diagramas de componentes de UML [8, 13]. El diagrama muestra tipos de componentes, o sea que su definición está dada por una interfaz y una semántica. En nuestros diagramas las interfaces están claramente definidas con las primitivas de UML para definir interfaces, y las únicas dependencias que se indican son las llamadas a interfaces en otras componentes. La semántica está dada solamente por el nombre de las componentes; UML no provee una forma para especificar semántica de componentes. Nosotros hemos explorado la utilización de Input/Output Autómatas [10] para especificar la semántica tanto de tipos como de clases de componentes [3].

En nuestro ejemplo de la figura 3 existe una componente que representa un sistema CAD que genera una malla de superficie con un cierto formato y lo almacena en una componente CAD_DATA. Para poder manipular la malla es necesario cargarla en nuestras estructuras de datos DATA; para ello la componente LOAD_DATA_CAD lee la información de la malla de superficie y carga las estructuras de datos. Una vez que la información de la malla de superficie está disponible, se puede mejorar con IMP_SURF_MESH, y generar una primera malla

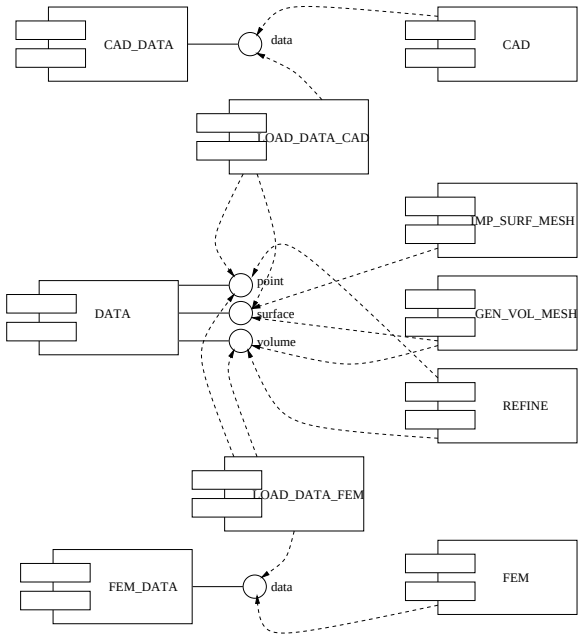

Figura 3: Arquitectura de nivel de interfaz para LPS de generación y manejo de mallas geométricas.

de volumen con GEN_VOL_MESH. Para que la malla de volumen sea usada por un software de manejo de elementos finitos (FEM) es necesario generar la información de la malla de volumen en un formato apropiado FEM_DATA; de esto se encarga la componente LOAD_DATA_FEM. La componente REFINE permite refinar adaptiva o interactivamente una malla de volumen.

A partir de esta definición de la arquitectura base es claro ver que se puede construir diferentes productos. Por ejemplo, con las componentes de LOAD_DATA_CAD e IMP_SURF_MESH, además de DATA, podríamos construir un producto tal que permita a un producto CAD generar mallas de superficie de buena calidad. Otro producto posible estaría formado por DATA, LOAD_DATA_FEM y REFINE y permitiría a un software FEM tener un módulo de refinamiento de mallas.

La arquitectura base de nuestra línea de productos definida al nivel de interfaz es clara y es posible visualizar en ella diferentes productos planeados. Sin embargo, para hacer una reutilización efectiva de componentes de código, además de la estructura general, las interfaces y la semántica de las componentes, es necesario definir diferentes implementaciones. Por ejemplo, si se desea producir módulos para generación de mallas de volumen para diferentes CAD que usen diferentes formatos para definir mallas de superficie, tendríamos productos con una arquitectura idéntica, consistente con la de la figura 3, pero las implementaciones deberían ser distintas. Para esto, la definición de la arquitectura base a nivel de interfaz no es suficiente.

3.3 Definición de Arquitectura Base a Nivel de Implementación

Tal como lo discutíamos en la sección 3.2, es necesario incluir dentro de la definición de la arquitectura base las distintas posibles implementaciones de todas las componentes, de modo de hacer más directa la reutilización efectiva de componente de código concretas. La figura 4, muestra un refinamiento posible de la arquitectura de la figura 3 usando el nivel de implementación de $\mathcal{I}^5$ [2]. Aquí las componentes se representan mediante un estereotipo de tipos de componente [13] que es una *clase* de componente. Una clase es un tipo que además de interfaz y semántica, contiene una implementación específica.

Toda clase de componente que aparece en el diagrama del nivel de implementación corresponde a una implementación particular de un tipo de componente que aparece necesariamente en el nivel de interfaz. Como las clases son refinamientos de los tipos, éstas deben preservar tanto las interfaces como la semántica definida para los tipos que refinan. También si un tipo de componente invoca una interfaz en otro tipo, la clase que refina el primer tipo solamente podrá invocar la misma interfaz en la clase que refina el segundo tipo.

Es posible que exista un tipo de componente en el diagrama de interfaz para el cual no exitan implmentaciones disponibles y, por lo tanto, no habrá una clase que lo refine en el nivel de implmentación. Sin embargo, si en el futuro se provee una implementación, ésta deberá ser consistente con la interfaz y la semántica definida para ese tipo de componente en el nivel de interfaz.

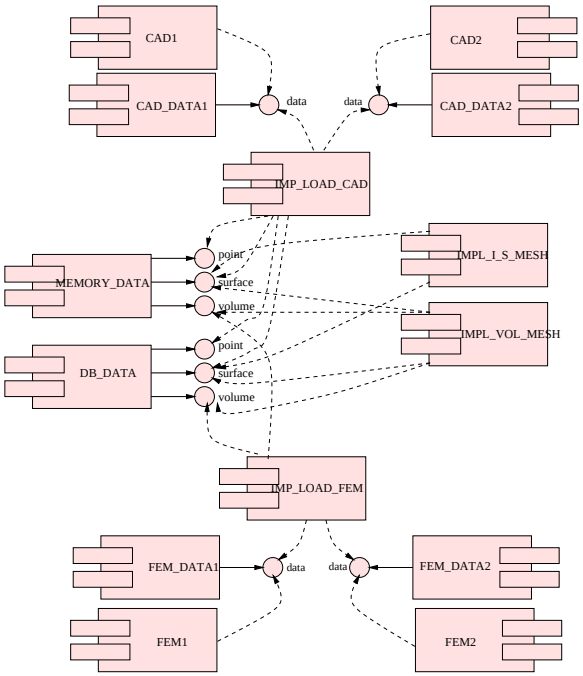

Figura 4: Arquitectura de nivel de implementación para LPS de generación y manejo de mallas geométricas.

En nuestro ejemplo tenemos dos clases de componentes del tipo CAD: CAD1 y CAD2, con sus correspondientes componentes de datos CAD_DATA1 y CAD_DATA2. Dado que es posible querer manejar los datos de las mallas en memoria o bien en una base de datos, habrá dos posibles implementaciones del tipo DATA: MEMORY_DATA y DB_DATA. Habrá una única implementación del tipo LOAD_DATA: IMP_LOAD_DATA, que se encarga de cargar las estructuras de datos cualquiera sea su implementación; nótese que esto es posible porque ambas implementaciones de DATA tienen necesariamente una interfaz idéntica definida en la arquitectura de nivel de interfaz. Análogamente habrá implementaciones diferentes para permitir interactuar con dos software FEM diferentes. Por el momento hay una única implementaciónde IMP_SURF_MESH: IMPL_I_S_MESH, una implementación de GEN_VOL_MESH: IMPL_VOL_MESH, y ninguna implementación de REFINE.

La definición de la arquitectura base de la LPS a nivel de implementación debe ser tal que si se colapsan todas las implementaciones de un mismo tipo de componente, se obtiene la definición del nivel de interfaz, o parte de la misma. Esto es evidente en nuestro ejemplo de las figuras 3 y 4 considerando que no existe una implementación para REFINE.

4 Nivel de Instancia o Producto

La arquitectura de un producto de software específico, y en particular un producto de una LPS, es el conjunto concreto de sus componentes y la interacción entre las mismas. Tal como lo indicáramos en la sección 2, la arquitectura base de una LPS debe delimitar el alcance y guiar la construcción de los productos. Veamos entonces cómo se comporta nuestra especificación en dos niveles tanto para desarrollar productos previstos como para dar cabida a productos no anticipados.

4.1 Producto Previsto: Generador de Mallas de Volumen para un CAD

Dado un producto CAD particular, por ejemplo CAD1, queremos desarrollar un producto tal que le permita a este software ser capaz de generar mallas de volumen. Para ello seleccionaremos de la arquitectura de la figura 4 las clases de componentes: CAD1, CAD_DATA1, MEMORY_DATA, IMPL_VOL_MESH. Instanciamos cada una de estas clases de componentes y las combinamos de acuerdo a lo indicado en la arquitectura base de nivel de implmentación para dar lugar a nuestro producto que se describe en la figura 5.

La arquitectura de un producto particular se define usando el nivel de instanciación de $\mathcal{I}^5$. Aquí los elementos son instancias de componentes, con la restricción de que solamente pueden instanciarse las clases de componentes que aparecen en el nivel de implementación. Para estos diagramas se usan los diagramas de *deployment* estándar de UML a nivel de instancias [8], incluyendo solamente las instancias de componentes. Nótese que en la figura 5, las instancias de componentes se identifican con un nombre en letra minúscula, seguida del nombre de la clase de componente a la cual pertenece, todo subrayado, tal como es estándar en la notación de UML para instancias.

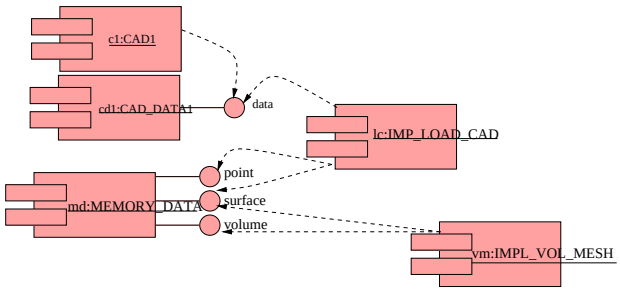

Figura 5: Arquitectura de nivel de instanciación para producto de generación de mallas de volumen para CAD1.

Nótese que para este producto, que claramente había sido previsto ya que existían en la arquitectura base al nivel de implementación clases de componentes específicamente para CAD1, la arquitectura concreta del producto es solamente la instanciación de un subconjunto de la arquitectura al nivel de implementación.

4.2 Producto no Previsto: Mejoramiento Paralelo de Grandes Mallas de Superficie

Consideremos ahora un producto que permita a un producto CAD2 producir grandes mallas de superficie de buena calidad, proporcionando para ello no solamente almacenamiento de los datos en una base de datos, sino también un algoritmo paralelo de mejoramiento de la malla. Nuestra arquitectura base de nivel de implementación contiene las clases de componentes CAD2, CAD_DATA2, IMP_LOAD_CAD y DB_DATA, que podrían instanciarse para dar lugar a nuestro nuevo producto. Pero la clase de componente que contiene el algoritmo de mejoramiento de mallas de superficie es una implementación secuencial; no contamos con una implementación paralela. Sin embargo, una instancia de componente que implemente el algoritmo paralelo deberá tener la misma

interfaz y semántica que el tipo de componente IMP_SURF_MESH que aparece en la arquitectura base del nivel de interfaz. El desarrollo de este nuevo producto implicará el desarrollo de la componente PAR_IMP_MESH como un nuevo refinamiento de IMP_SURF_MESH que será agregado a la arquitectura de nivel de implementación e instanciado para nuestro producto. La figura 6 muestra la arquitectura del producto de mejoramiento paralelo de grandes mallas de superficie.

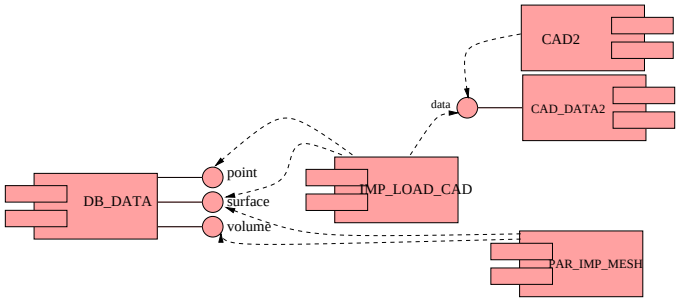

Figura 6: Arquitectura de nivel de instanciación para producto de mejoramiento paralelo de grandes mallas de superficie.

Este nuevo desarrollo, si bien no contaba con todas las componentes implementadas como activos de la LPS, claramente caía dentro del alcance de la LPS, dado que sí era compatible con la arquitectura del nivel de interfaz. Para desarrollar este nuevo producto es necesario hacer una nueva implementación de un tipo de componente que ya estaba definido; así, además del producto en sí, nos queda enriquecida la arquitectura base del nivel de implementación con una nueva clase de componente que podrá ser reutilizada en otros productos futuros.

5 Conclusiones y Trabajo Futuro

Disponer de una arquitectura base general que describa el alcance de la LPS como la arquitectura de nivel de interfaz presentada en la sección 3.2 ayuda a decidir cuáles de los nuevos productos caen dentro del alcance de la LPS, pero no ayuda mucho en la reutilización de componentes concretas de software. Una arquitectura base solamente orientada a la implementación como la presentada en la sección 3.3, ayuda en la evaluación del costo de desarrollar nuevos productos, pero no en la determinación de si un nuevo producto cae o no dentro del alcance de la LPS. Tener la arquitectura base definida en ambos niveles permite lograr los dos objetivos: determinar si un nuevo producto está dentro del alcance de la LPS independientemente de las componentes disponibles para reutilizar en su construcción, y guiar la implementación de productos concretos proporcionando la información de las componentes reales disponibles.

Ambos niveles de la arquitectura base deben ser consistentes entre sí, y también consistentes con la arquitectura particular de cada uno de los productos de la LPS. El lenguaje de especificación de arquitecturas $\mathcal{I}^5$ nos permite describir gráficamente la arquitectura de la línea de productos en estos tres niveles: interfaz, implementación e instanciación, asegurando su consistencia.

Todas las metodologías iterativas de desarrollo se basan en definir tempranamente una arquitectura que luego se irá poblando. Si esto es algo desafiante en cualquier desarrollo, mucho más lo es en una la línea de productos de software en que la definición de la arquitectura es muchas veces anterior a la definición de cualquiera de los productos, y deberá ser tal que todos los productos que se definan se adapten a esta arquitectura.

En este esquema de dos niveles de arquitectura base y un nivel de arquitectura concreta de los productos, cada parte de la organización se ve afectada. La ingeniería del dominio deberá definir la arquitectura de nivel de interfaz inicialmente para definir el alcance de la LPS. A medida que se vayan desarrollando productos

concretos, se mantendrá la definición del nivel de implementación de la arquitectura base, de acuerdo con las nuevas implementaciones de las componentes de la interfaz que vayan siendo desarrolladas. La ingeniería del producto tendrá un claro marco para el desarrollo de nuevos productos en la arquitectura de nivel de interfaz, así como también una guía concreta de los componentes disponibles para implementar el nuevo producto. La administración, por su parte, podrá tener elementos concretos para evaluar cuándo un producto cae dentro del alcance de la línea de productos, y el esfuerzo necesario para el desarrollo de nuevos productos usando los dos niveles de definición de la arquitectura base.

5.1 Trabajo Futuro

Parte esencial de nuestro trabajo en curso consiste en refinar aún más las especificaciones de la arquitectura base agregándoles la semántica de las interacciones entre las componentes. Ya hemos avanzado en este sentido usando el lenguaje de I/O Autómatas [10] integrado a $\mathcal{I}^5$ y publicado en [3]. Ahora queremos profundizar en esta integración específicamente para los distintos niveles de especificación de la arquitectura base de una línea de productos de software.

También estamos estudiando las condiciones organizacionales necesarias para desarrollar una línea de productos con este tipo de arquitectura base definida en varios niveles. Como ya lo dijimos en nuestras conclusiones, esta definición afecta a todas las partes de la empresa. En este sentido tenemos ya algunos avances [14].

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto I-01-2/2001 del DID (Departamento de Investigación y Desarrollo) de la Universidad de Chile.

Referencias

[1] M. Cecilia Bastarrica, Steven A. Demurjian, and Alex A. Shvartsman. $\mathcal{I}^5$: A Framework for Architectural Specification of Distributed Object Systems. In *Proceedings of the 3rd International Conference On Principles Of DIstributed Systems, OPODIS'99*, Hanoi, Vietnam, October 1999.

[2] María Cecilia Bastarrica. *Architectural Specification and Optimal Deployment of Distributed Systems.* PhD thesis, University of Connecticut, January 2000.

[3] María Cecilia Bastarrica, Steven A. Demurjian, and Alex A. Shvartsman. *Comprehensive Specification of Distributed Systems Using $\mathcal{I}^5$ and IOA.* In *Proceedings of the XX International Conference of the Chilean Computer Science Society (SCCC)*, Santiago, Chile, November 2000.

[4] John Bergey, Liam O'Brien, and Dennis Smith. Mining Existing Assets for Software Product Lines. Technical Report CMU/SEI-2000-TN-008, Software Engineering Institute, Carnegie Mellon University, 2000.

[5] Grady Booch, James Rumbaugh, and Ivar Jacobson. Version 1.1 of the Unified Modeling Language (UML), September 1997. Rational Software Corporation. Available at `http://www.rational.com`.

[6] Frederick P. Brooks. No Silver Bullet. Essence and Accidents of Software Engineering. *Computer Magazine*, April 1987.

[7] Paul Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns.* Addison Wesley, first edition, August 2001.

[8] Hans-Erik Eriksson and Magnus Penker. *UML Toolkit.* John Wiley and Sons, Inc., first edition, 1998.

[9] Martin Fowler. *UML Distilled. Applying the Standard Object Modeling Language.* Object Technology Series. Addison Wesley, 7 edition, June 1998.

[10] Stephen J. Garland and Nancy A. Lynch. The IOA Language and Toolset: Support for Designing, Analyzing, and Building Distributed Systems. Technical Report MIT/LCS/TR-762, MIT Laboratory for Computer Science, Cambridge, MA, August 1998.

[11] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, 1991.

[12] Marc Halpern. Industrial Requirements and Practices in Finite Element Meshing: A Survey of Trends. In *Proceedings of the 6th International Meshing Roundtable'97*, Park City, Utah, October 1997. Sandia National Laboratories.

[13] Unified Modeling Language. UML Metamodel, September 1997.

[14] Marcelo López and M. Cecilia Bastarrica. Business Case for a Product Line of Legacy Application Data Middleware. Technical Report TR/DCC-2002-3, DCC, Universidad de Chile, 2002. `http://www.dcc.uchile.cl/ cecilia/publicaciones.html`.

[15] Nenad Medvidovic and Richard Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1), January 2000.

[16] Nenad Medvidovic, Richard Taylor, and E. James Whitehead Jr. Formal Modeling of Software Architectures at Multiple Levels of Abstraction. In *Proceedings of the California Software Symposium 1996*, pages 28–40, April 1996.

[17] National Science Foundation Information Technology Research (NSF/ITR). Adaptive Software Project, 2001. `http://www.erc.msstate.edu/ jcollins/ITR/index.html`.

[18] Mary Shaw and David Garlan. *Software Architecture. Perspectives on an Emerging Discipline.* Prentice Hall, 1996.

[19] Software Engineering Institute (SEI); Carnegie Mellon University. The Product Line Practice Initiative, 2002. `http://www.sei.cmu.edu/plp/`.