Universidad de las Ciencias Informáticas
Dirección Técnica, Infraestructura Productiva

Definición de una arquitectura de referencia para una línea de productos de software

Tesis para optar por el título de Master en Gestión de Proyectos Informáticos

Autor: Lic. Manuel Vázquez Acosta
Tutor: DrC. Pedro Y. Piñero Pérez

Ciudad de la Habana
2011

Para mi hijo y esposa.
Para mi madre.

Gracias a varias personas este trabajo ha llegado a un final feliz.
Mi tutor y cuñado Pedro Piñero merece un reconocimiento especial por todo su empeño y ayuda.
A mis colegas de la Dirección Técnica, cuya compañía y apoyo son invaluables.
Al equipo del departamento de Integración de Soluciones de DATEC, y al equipo conjunto de Caxtor les debo el apoyo para la realización de las ideas de este trabajo.
También fue importante la ayuda de varios colegas del centro CEIGE y el UCID: René Lazo, Oiner Gómez, y David Martínez, con quienes fueron desarrollados mano a mano varios proyectos que contribuyen significativamente a este trabajo.

A todos mi eterna gratitud.

Resumen

Este trabajo expone la concepción y desarrollo de una arquitectura de referencia para la línea de productos de software Paquete de Herramientas para la Toma de Decisiones y Sistemas Inteligentes (PATDSI) del Centro DATEC, en la Universidad de la Ciencias Informáticas.

Las líneas de producto de software son modelo de desarrollo de software que ha sido aplicado con éxito en muchas empresas a nivel mundial desde la década del 1990. Este trabajo también implica que este modelo de desarrollo puede ser aplicado con resultados positivos en proyectos de desarrollo de la UCI.

El principal aporte del trabajo es que contribuye al *aumento de la productividad* de la línea PATDSI. De forma adicional se brinda un grupo de actividades que fueron imprescindibles para la introducción del cambio en DATEC. Se destaca que, si bien la arquitectura de una línea de productos es necesaria para lograr los resultados deseados en cuanto a la reutilización y la reducción del esfuerzo, no es suficiente por sí sola y hay que establecer un grupo de políticas y prácticas para lograr que todos los proyectos contribuyan al desarrollo de los diferentes productos de la línea de productos.

Abstract

This thesis describes the development of a reference architecture for a software product line called Suite of Tools for Aiding Decision Making and Intelligent Systems (PATDSI, by its Spanish spelling) produced by the center DATEC at the University of Informatics Sciences.

Software Product Lines are a kind of development model which has been successfully applied in many commercial and government institutions since the 1990's. The results of this thesis entail that this model can be applied for organizations within the University, and it can achieve positive results.

The main result of this thesis is that it has helped to *gain more productivity* in the development of products. Also it is provided a set of activities that were of the up-most importance to adopt the SPL on DATEC. It is noteworthy that, despite being a necessary assets, architecture alone can't improve the organization's productivity; it's needed a set of policies and practices to make project work together to achieve the overall goals of the SPL.

Índice general

Índice de cuadros

Índice de figuras

Introducción

Durante las últimas décadas, en la industria del software se han creado varios modelos de desarrollo que soporten todo el trabajo que se requiere para construir, implantar y mantener sistemas de software. Comenzando con el modelo de desarrollo en cascada, se comienza a hacer énfasis en el diseño de los sistemas utilizando lenguajes de modelado específicos para el software. Estos modelos han ido evolucionando, y en los últimos años la tendencia es a utilizar modelos que sean iterativos y/o incrementales (Piñero Pérez y Leyva Vázquez, 2010).

El objetivo final de estos modelos es garantizar la ejecución de *proyectos de desarrollo de software*; y al decir «proyecto» se entiende que existen varias limitaciones que deben ser consideradas, incluyendo plazos a cumplir y niveles de calidad que se deben alcanzar. Desde muy temprano en la industria del software (Brooks, 1987; Dahl y otros, 1972) se ha reconocido que es importante reducir la repetición de conceptos, lo cual ha evolucionado en el principio DRY (*Don't Repeat Yourself*) (Hunt y Thomas, 1999) que indica que cada elemento (incluyendo los de configuración, documentación, y pruebas) deben definirse en único lugar.

En su esencia este principio introduce cuestiones sobre cómo utilizar y reutilizar un elemento en diferentes contextos. En su forma más básica, la reutilización se manifiesta como una función identidad; es decir, el elemento se utiliza sin transformar. Esta es una de las formas más tempranas de reutilización y muchos paradigmas de programación la soportan: las rutinas o procedimientos en la programación estructurada (Dahl y otros, 1972), las clases en la programación orientada a objetos (Booch, 1998), los componentes en sistemas basados en componentes (Crnkovic y Larsson, 2002), etc. Otras formas más elaboradas han sido desarrolladas en la llamada *programación literaria* (Knuth, 1992) y también la programación orientada a aspectos (Kiczales y otros, 1997). En estos casos, los elementos a reutilizar son transformados según el contexto para producir el resultado deseado.

De forma concurrente estos paradigmas han impactado y han sido influidos por varios modelos de desarrollo de software. Una vista de la evolución de los modelos de desarrollo en función de cómo se garantiza la reutilización se brinda en la figura 1. Muchos de los modelos en esta figura asumen algún mecanismo de reutilización, o son definidos por la unidad de reutilización fundamental.

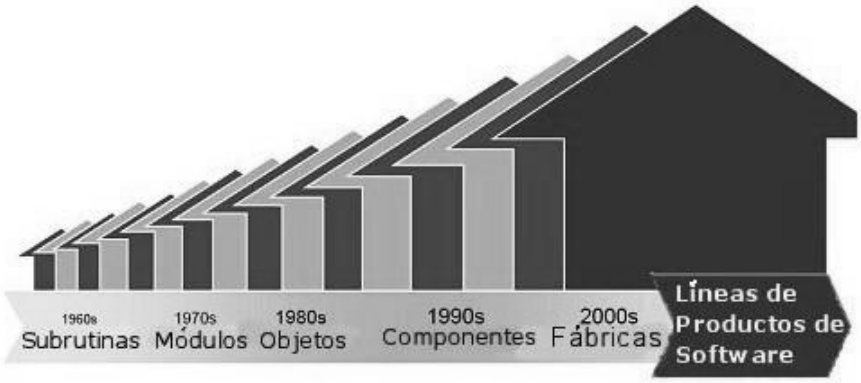

Figura 1: Una vista de la evolución de la industria del software en función de la reutilización. Tomada de (Piñero Pérez y Leyva Vázquez, 2010).

El objeto de la reutilización

La reutilización por sí misma no es un objetivo de las empresas de desarrollo de software. Sin embargo, emplearla es un medio para garantizar el cumplimiento de otros objetivos, entre los cuales se pueden mencionar:

- Reducción del tiempo y costos para el desarrollo de nuevos productos
- Cumplir con los niveles de calidad deseados para cada producto

Las últimas décadas reflejadas en la figura 1 muestran tres esquemas relevantes:

- La ingeniería basada en componentes
- Las fábricas de software
- Las líneas de productos de software (LPS)

La última de estas variantes, las líneas de productos de software, es una que promociona al extremo la reutilización[1] para responder a las necesidades de una sección específica del mercado (Clements y Northrop, 2001).

[1] Los autores de (Lormans y van Deursen, 2006) indican, por ejemplo, que los nuevos productos de la empresa Philips Applied Technologies reutilizan entre el 80 % y 90 % de las producciones de proyectos anteriores, y es conocido que esta empresa es una de las que aplica las LPS para la creación de sus productos (Clements y otros, 2005).

Los principales promotores de las líneas de productos de software la definen como: «un conjunto de sistemas con uso intensivo de software que comparten un *grupo común y regulado* de características que satisfacen las necesidades de un *mercado o misión particulares*, y que son desarrollados a partir de un *núcleo común de activos* siguiendo un *método prescrito*» (Northrop y Jones, 2008).

La definición de una LPS se compone de, al menos, tres elementos:

- El *dominio* o misión de los productos de la línea.
- El conjunto de *activos* con los cuales se construyen los productos.
- *Planes* de construcción que indican cómo desarrollar un producto a partir del conjunto de activos.

De estos elementos, los activos constituyen la base de una LPS y son la *unidad básica de reutilización* en las LPS. Los activos no son solamente código ejecutable, sino también cualquier artefacto que pueda ser sistemáticamente reutilizado. Como ejemplos de activos se pueden mencionar: la arquitectura base, modelos del dominio, documentos de requisitos, planes de pruebas, listas de chequeo, componentes ejecutables, y muchos otros.

En (Bass y otros, 2003; Clements y Northrop, 2001) se reconoce que quizá el activo imprescindible es la arquitectura. En (Zubrow y Chastek, 2003) también se apoya la idea al decir que «un componente clave para la solución efectiva de esos problemas [se refiere a los problemas que intentan resolver las LPS] es el uso de una *arquitectura de línea de productos* que permita a la organización identificar y reusar los artefactos. La arquitectura es, en ese sentido, la que asegura la cohesión de la línea de productos» (Zubrow y Chastek, 2003, p. 2)[2].

En esta tesis se presenta el diseño de una arquitectura de referencia para la línea de productos de software Paquete de Herramientas para la Ayuda a la Toma de Decisiones y Sistemas Inteligentes (PATDSI) del Centro de Gestión de Datos (DATEC). Esta línea tiene como misión la construcción de sistemas de ayuda a la toma de decisiones. Durante la definición de la LPS PATDSI se han identificado un grupo de activos necesarios para cumplir con los objetivos de la línea, y también se ha necesitado la construcción de activos de procesos como cronogramas, diseño de roles y responsabilidades, etc.

El aporte fundamental de la tesis se puede localizar en un aumento de la *productividad*. Aunque tampoco se puede ignorar su aporte en la normalización de la producción, el establecimiento de un proceso más repetible y acorde con los modelos de madurez.

En las siguientes secciones se introduce más detalladamente el problema y se hace un resumen del diseño de la investigación.

[2] En el original la última oración dice literalmente «The architecture, in a sense, is the glue that holds the product line together».

Paquete de herramientas para la ayuda a la toma de decisiones

Desde su creación, DATEC se ha propuesto la creación de líneas de productos de software para así garantizar el cumplimiento de sus objetivos estratégicos. Si bien algunas de las líneas creadas han tenido éxito y se han consolidado, otras no han terminado su formación. En particular la línea Paquete de Herramientas para la Ayuda a la Toma de Decisiones todavía no tiene identificado algunos de sus activos más importantes:

Arquitectura base: La arquitectura es uno de los activos claves en cualquier LPS (Bass y otros, 1997, 2003). Esta brinda, entre otras cosas, una forma unificada de representación de los componentes y, sobretodo, las maneras de integrarlos. «La arquitectura da una estructura única a aquello que es común a todos los productos y cómo pueden variarse» (Bass y otros, 2003, cf. ep. 2.4). Por otra parte, la arquitectura brinda a los sistemas de atributos críticos como fiabilidad, eficiencia, modificabilidad etc. (Garlan y Shaw, 1994).

Componentes reutilizables: Aunque se han identificado parcialmente *necesidades* similares en varios de los proyectos, no se han generalizado como activos de la línea; sino que cada proyecto ha sido enfrentado de forma casi independiente.

Modelos de dominio: De la misma manera, el análisis se ha estado realizando de forma específica para productos y no se estudia un dominio general.

Procedimientos de liberación y configuración: En la gestión de configuración se tiene actualmente varios lineamientos generales del centro, pero necesitan revisarse dado que no se considera cada activo; en particular, solamente se crean versiones para los productos y no los activos y componentes.

Esta situación provoca muchos problemas con la gestión de la línea. En DATEC se han tomado medidas para mitigar estos problemas. Por ejemplo:

- Se han puesto en vigor el uso de herramientas de gestión de proyectos como el Redmine de conjunto con algunos lineamientos mínimos para la gestión de configuración. Sin embargo, esto no cubre todas las posibilidades; especialmente no atiende completamente la configuración de un producto particular.
- Se han ido construyendo los modelos del dominio en la medida en que se van desarrollando los mismo productos y contando con la asistencia de expertos funcionales.

Ninguna de estas alternativas constituye una solución real al *problema* que se está presentando. El cual se define en esta tesis como: *la ausencia de mecanismos de integración a nivel arquitectónico y también el bajo nivel de reutilización en la línea de productos PATDSI está provocando baja productividad.*

Existen varias evidencias del problema en la situación de DATEC:

- Existe poca reutilización entre los productos de la línea. Básicamente cada producto se diseña aislado del resto y no se ha logrado identificar conceptos unificadores en los productos. Por ejemplo, los productos SIGE 1.0 y GDR 1.5 emplean «modelos» para representar los datos que deben ser manipulados, sin embargo, el concepto de modelo de uno es radicalmente diferente al del otro. Esto ha provocado que se invierta mucho tiempo a la hora de integrar estos dos sistemas.

- A pesar de que muchas de las soluciones de PATDSI involucran componentes de captura y visualización de datos, la integración entre estos componentes existe solo a nivel de datos y esto dificulta la relación entre estos componentes.

  Este nivel de integración dificulta la integración a nivel de aplicación y no atenta contra la reutilización de componentes de la interfaz de usuario.

  Configurar los componentes de generación de reportes para datos capturados por SIGE 1.0 es un trabajo que toma varias semanas, precisamente porque la forma en que SIGE almacena estos datos no se puede «transportar» a nivel arquitectónico hacia los componentes del GDR.

Objeto de estudio y campo de acción

El objeto de estudio son *las líneas de productos de software* y también *las arquitecturas de software* aplicadas a las líneas de productos de software. El campo de acción de esta tesis es la línea de productos para las herramientas de ayuda a la toma de decisiones PATDSI.

Objetivos

Se plantea como objetivo general de este trabajo la *definición de una arquitectura de referencia para la línea de productos de software PATDSI* del Centro de Tecnologías de Gestión de Datos (DATEC) en la UCI, que permita mejorar la productividad de la línea.

Para el cumplimiento de este objetivo se plantean los siguientes objetivos específicos:

- Elaborar el marco teórico-conceptual de los temas relativos a las líneas de productos de software, la arquitectura de software, y la ingeniería para líneas de productos de software mediante un estudio del estado del arte en esas materias.
- Definir la arquitectura de referencia de la línea PATDSI y sus activos principales
- Validar la arquitectura de referencia mediante su aplicación en varios proyectos del Centro DATEC.

Hipótesis y variables de la investigación

Para resolver el problema dado se plantea la siguiente hipótesis: «La definición y establecimiento de una arquitectura de referencia para la línea PATDSI, de conjunto con la identificación de sus activos; tendrá un impacto positivo en la productividad de esta línea de productos de software»; en particular, la productividad se elevaría al:

- Aumentar el nivel de reutilización de componentes dentro de la línea. Debido a que el esfuerzo empleado para construir un componente tendría efectos positivos en la mayoría de los productos de la línea.
- Disminuir el esfuerzo necesario para la integración de productos y componentes. Una vez que existan componentes pre-fabricados, y especialmente diseñados para la misión de los productos de la línea, la labor de integración de estos componentes es la que dominaría en los cronogramas para la construcción de productos. Las arquitecturas de referencias, pueden establecer mecanismos para la integración de componentes y facilitar la labor de integración.

Esta tesis trabaja fundamentalmente con dos variables:

- la variable independiente «Arquitectura de referencia» que mide cómo se ha ido avanzando en la definición e introducción de la arquitectura dentro de la línea PATDSI;
- y la variable dependiente «Productividad de la línea» que mide cómo ha ido cambiando la productividad de la línea en función de la introducción de los principios de la arquitectura de referencia.

La variable «Productividad de la línea» es desglosada como se muestra en la siguiente tabla:

Cuadro 1: Desglose de la variable «Productividad de la línea»

Dimensión	**Indicador**	**Medida**
Reutilización	Nivel de reutilización media por componente	El promedio de la cantidad de productos en que se utiliza un componente.
Integración de productos y componentes	Esfuerzo de integración	Horas-hombres empleadas en la integración de componentes.

La variable «Arquitectura de Referencia» se desglosa en la tabla que aparece a continuación:

Cuadro 2: Desglose de la variable «Arquitectura de Referencia»

Dimensión	**Indicador**	**Medida**
Conceptual	Definición	Sí/No
	Implementación	Porcentaje de implementación según el diseño.
Organizativa	Nivel de introducción	Porcentaje de los proyectos que usan la arquitectura.
Activos	Activos definidos	Número que identifica los activos definidos.
	Porciento de implementación	Porciento de activos que han sido implementados.

Métodos de la investigación

Durante el desarrollo de esta tesis se han empleado varios métodos. Los métodos teóricos empleados son:

- El histórico-lógico que ha permitido detectar las tendencias en la bibliografía; en particular, se ha utilizado durante el estudio del estado del arte para realizar un seguimiento evolutivo de los conceptos de líneas de producto de software, y arquitectura de software.
- El hipotético-deductivo, pues se planteó una hipótesis que ha centrado los esfuerzos de la investigación.

De los métodos empíricos se utilizaron:

- La observación y medición científicas, puesto que fue necesario observar objetivamente cómo se comporta la línea a medida que se iban diseñando e introduciendo los elementos de la arquitectura de referencia.
- Los métodos estadísticos-matemáticos que se utilizaron para estudiar los resultados obtenidos durante la observación.

Población y muestra

La población estudiada fueron *todos los proyectos en la línea PATDSI*. El muestreo de estos proyectos se realizó de forma selectiva. No se consideró una muestra aleatoria debido a que:

- Los proyectos de una LPS están relacionados temporalmente por lo que no se pueden considerar resultados independientes.
- La cantidad de proyectos de la línea no sobrepasa a la media docena y por tanto la estrategia de consolidación de la LPS debe ser conservadora en amplitud pero radical en profundidad.

Estrategia de la investigación y aporte práctico

Este trabajo asume una estrategia experimental respecto a la investigación e intenta explicar las causas de los problemas de productividad que se están presentando en la línea. A la vez, este trabajo está completamente involucrado en el mismo proceso productivo como parte de la estrategia de internalizar la investigación científica dentro de los procesos productivos. Por esta razón esta tesis influye directamente en la producción del centro DATEC y su principal aporte es *la reducción del esfuerzo* necesario para la creación de nuevos productos dentro de la línea PATDSI, o dicho de otra forma, *un aumento en la productividad de la línea.*

El resto de este documento se estructura en tres capítulos: En el capítulo 1 se hace un estudio del estado del arte en relación a las LPS, la arquitectura de software, y los modelos de desarrollo de software en general. En el capítulo 2 se expone el diseño de la arquitectura de los productos de la línea, se identifican los activos, y se brinda el método (junto con un listado de actividades claves) para la creación de los productos de la línea. Finalmente en el capítulo 3 se hace una valoración sobre el resultado de la aplicación de la propuesta en la práctica.

Capítulo 1

Fundamentación teórica

En este capítulo se hace una revisión de la literatura sobre líneas de productos de software, arquitecturas de software, y modelos de desarrollo de software. Se comienza con una revisión del tema de las líneas de producto de software, en la cual se brindan los conceptos más relevantes para este trabajo, incluyendo la gestión de proyectos dentro de una LPS y la representación formal de la variabilidad de los productos como un elemento clave en la arquitectura de las LPS.

Luego se hace una revisión de la disciplina de arquitectura de software, en la cual se exponen las definiciones de arquitectura, y estilo arquitectónico. Se incluye una revisión de varios estilos arquitectónicos que son relevantes para el objetivo de esta tesis.

Se finaliza el capítulo con una breve discusión del rol que juega la arquitectura en la gestión de proyectos de software.

1.1. Líneas de productos de software

Una línea de productos de software (LPS) es un modelo de desarrollo de software que se basa en la creación de varios productos similares a partir del «ensamblaje» de componentes pre-fabricados (Montilva, 2006). Las LPS son una evolución de ideas anteriores que habían dado sus frutos en la creación de sistemas basados en componentes y las factorías de software (más información en Crnkovic y Larsson, 2002; Piñero Pérez y Leyva Vázquez, 2010).

El hilo conductor de la evolución hacia las líneas de producto de software pasa por la *reutilización* como aspecto clave para reducir los costos, aumentar la productividad y asegurar la calidad de los productos. Sin embargo, como se detalla en (Montilva, 2006), en las LPS la reutilización se caracteriza por ser:

- estratégica en el sentido de que consolida lo común entre los productos de la línea, se maneja estratégicamente la variación de los productos, y elimina la duplicación de esfuerzos de ingeniería;
- se gestiona de forma sistemática, se planifica y se mejora continuamente.

De cierta manera, la reutilización en las LPS deja de ser un aspecto puramente técnico y pasa a ser, además, un elemento que los directivos en la LPS persiguen.

Las formas de organización de la producción para lograr este nivel de reutilización también se han documentado detalladamente. Comenzando por los trabajos de Clements y Northrop (Clements y Northrop, 2001), la consolidación en un marco de trabajo que es continuamente actualizado (Northrop y Clements, 2007a), y la aparición de modelos más ligeros compatibles con metodologías de desarrollo ágiles (Krueger, 2007).

Todo este movimiento hacia las LPS no viene únicamente dado por razones de mejora en la reutilización y las formas de producción, sino porque a las LPS se le atribuyen ventajas notables. En (Bass y otros, 1997) y (Montilva, 2006) se enumeran las siguientes:

- Reducción del tiempo para el mercado.
- Incremento de la productividad.
- Reducción de costos en la producción.
- Mejor retorno de inversión.
- Aumento de la calidad de los productos.

En otro reporte realizado por el Instituto de Ingeniería de Software de la Universidad de Carnegie Mellon (SEI, por sus siglas en inglés) (Bergey y otros, 1998) se enumeran varias organizaciones que reportan beneficios:

- La compañía sueca CelciusTech, que construye sistemas de comando y control de la navegación, ha ampliado su alcance a doce tipos diferentes de navíos, desde veleros hasta submarinos, y ha logrado desarrollar hasta cincuenta sistemas a partir de la misma arquitectura y un grupo común de componentes. Reportan, además, una mejora en la tasa del costo de hardware sobre el software de un 35:65 a un 60:20[3].
- Hewlett Packard reporta que en un proyecto lograron incrementar cinco veces el número de productos, que eran cuatro veces más complejos, tenían tres veces más funciones y todo con una tasa de productos por persona cuatro veces más alta.

A pesar de estos beneficios, no se deja de señalar un grupo de riesgos que se pueden asumir con ciertas prácticas relacionadas con las LPS. En particular estos autores (Bergey y otros, 1998) indican que existen riesgos de alcance, de cambio cultural y también de carácter tecnológico. Otro autor (Krueger, 2007), además, indica que las prácticas expuestas inicialmente, principalmente aquellas documentadas en (Clements y Northrop, 2001), han probado ser muy complejas y difíciles de adoptar por las organizaciones.

Para resolver este problema Krueger adopta un modelo de tres niveles (Krueger, 2007), el cual será revisado más profundamente en la sección 1.2.4. Se reporta que usando este modelo la compañía HomeAway logró obtener los primeros resultados en el nivel básico de la implantación de una LPS en sólo sesenta días (Krueger y otros, 2008).

A pesar de estos posibles riesgos se puede observar que las LPS son ventajosas para las organizaciones de desarrollo de software; y que en muchos casos, la adopción del modelo es obligatorio para que las empresas sigan respondiendo a las demandas del mercado.

1.1.1. ¿Qué son las líneas de productos de software?

Las líneas de productos de software, según las definiciones dadas en (Clements y Northrop, 2001; Clements y otros, 2005; Northrop y Jones, 2008; Pérez Lamancha y Polo Usaola, 2009), son un conjunto de sistemas con uso intensivo de software que comparten un *grupo común y regulado* de características que satisfacen las necesidades de un *mercado o misión particulares*, y que son desarrollados a partir de un *núcleo común de activos* de una *manera preestablecida*.

En esta definición se pueden detectar tres elementos fundamentales para identificar una LPS:

- El *dominio* o misión de los productos de la línea. Los productos de una LPS están enfocados a un mercado particular, o con un propósito o misión bien determinado. En varios reportes (Bergey y otros, 1998, 1999), artículos y/o libros (Pohl y otros, 2005) se indica que en las LPS se pueden clasificar las actividades en aquellas de *ingeniería del dominio* e ingeniería de aplicaciones. Esto significa que una porción importante del trabajo que se realiza dentro de una LPS está enfocada al estudio sistemático del dominio de aplicación y la creación de componentes para ese dominio. Krueger llega a afirmar que la actividad de ingeniería de aplicaciones desaparece en las LPS maduras (Krueger, 2007).
- El conjunto de *activos* con los cuales se construyen los productos. Dentro de una LPS se debe reutilizar la mayor cantidad de trabajo posible. Los activos son la unidad básica de reutilización en una LPS y están estrechamente ligados a las actividades de ingeniería del dominio. Los procesos de construcción de activos, en los que se abundará más adelante, son una parte esencial de muchos de los modelos de LPS (Krueger, 2007; Northrop y Clements, 2007a; Pestano y Piñero Pérez, 2011; Pohl y otros, 2005).
- *Planes* de construcción que indican cómo desarrollar un producto a partir del conjunto de activos. En varios casos estos planes de construcción se subsumen en una arquitectura de referencia, la cual es, a su vez, unos de los activos fundamentales desarrollados dentro de la ingeniería del dominio (Ocharán Hernández y Cortes Verdin, 2010). Estos planes son los que en algunos modelos se automatizan como *configuradores de productos* (Krueger, 2008a,b; Montilva, 2006).

En las LPS se promueve la reutilización al extremo: se reutiliza cualquier artefacto y no solo el código fuente o algún componente ejecutable. Linda Northrop indica, haciendo referencia al artículo *«No silver bullet»* (Brooks, 1987), que la reutilización de código en sí no es la ventaja de una LPS, sino *la reutilización de decisiones* hechas mucho antes de la fase de programación (Bergey y otros, 1998, cf. 2.2.3, p. 9); en particular la reutilización de los modelos de dominio y sus correspondientes requisitos, además de otros

[3] En el original aparece exactamente «60:20»; esto puede deberse a un error al escribir y, tal vez, pudiera ser 60:40. De cualquier manera es un situación notablemente diferente.

artefactos de análisis. Pohl y otros, al introducir el concepto de *plataforma de software*, aclaran que ésta abarca los requisitos, arquitectura, código fuente y otros artefactos (Pohl y otros, 2005, p. 15). En ambos casos queda claro que los autores indican que hay mucho valor al reutilizar elementos que no son el código ejecutable.

Krueger y otros indican que el valor de la reutilización de las LPS consiste en que se logra evitar la construcción de sistemas *uno-para-todos*, los cuales intentan resolver todas las variantes con un único ejecutable, los cuales sufren de problemas que dificultan la gestión de los mismos (Krueger y otros, 2008).

En una LPS más que reutilizar componentes o artefactos se reutilizan *decisiones de cómo afrontar la variabilidad* de los productos en varios aspectos claves (Northrop y Clements, 2007a; Northrop y Jones, 2008; Pohl y otros, 2005). En la sección 1.1.3 se brindan varios elementos sobre la variabilidad en las LPS.

1.1.2. Los activos de una LPS y la arquitectura

En las LPS se les llama *activos* a los elementos que se reutilizan a lo largo de la vida de la línea. Cada activo es desarrollado de forma particular, pero debe garantizarse su utilidad de forma horizontal a toda la línea. De lo contrario puede fracasar su aplicación en los productos.

Los activos no son solamente componentes ejecutables. Dentro de los activos podemos encontrar, además de componentes de software, otros artefactos como son:

- modelos de dominio,
- planes de desarrollo y pruebas.
- cronogramas y presupuestos,
- listas de requisitos,
- la arquitectura de los productos, entre otros.

En mucha de literatura revisada existe consenso sobre que la arquitectura es uno de los activos más importante (Bass y otros, 1997; Clements y otros, 2005; Northrop y Clements, 2007a). Otros autores también apoyan la idea al decir que «un componente clave (...) es el uso de una *arquitectura de línea de productos* que permita a la organización identificar y reusar los artefactos. La arquitectura es, en ese sentido, la que asegura la cohesión de la línea de productos» (Zubrow y Chastek, 2003, p. 2). La arquitectura da una forma estable a todos los productos, indica los elementos de diseño que son comunes y también cómo se soporta la variabilidad de cada producto. De cierta manera, la arquitectura evoluciona a la par de los activos del dominio, e incluye componentes que sean necesarios para garantizar la variabilidad y otros atributos de calidad deseados.

La arquitectura también debe asegurar que existan mecanismos de variabilidad en los productos. Las formas de presentación de la variabilidad dentro de la arquitectura son esenciales para una LPS y forman la base de los mecanismos de adopción (Krueger, 2007).

En la arquitectura para una LPS convergen muchos de los criterios de calidad que son compartidos por todos los productos. Dependiendo del alcance de la línea deberá prestarse mayor o menor atención a cada atributo de calidad. No obstante, este tipo de decisiones, una vez tomadas, son aplicadas a muchos de los productos. Este tipo de reutilización de decisiones es que lo que hace particularmente atractivo el uso de las LPS.

1.1.3. La variabilidad de los productos en una LPS

En (Lauenroth y Pohl, 2005) se introduce un modelo formal para representar la variabilidad en los artefactos de requisitos, diseño, implementación y pruebas. Este modelo se basa en los conceptos de *punto de variación* y *variante*. Los puntos de variación son representaciones de aspectos que pueden cambiar, por ejemplo, el formato de salida de un reporte es un punto de variación. Las variantes son las posibilidades que existen para los puntos de variación. En el ejemplo anterior, las variantes del punto de variación «formato de salida del reporte» pueden ser PDF, DOC y ODT. La figura 1.1 muestra la notación utilizada por estos autores.

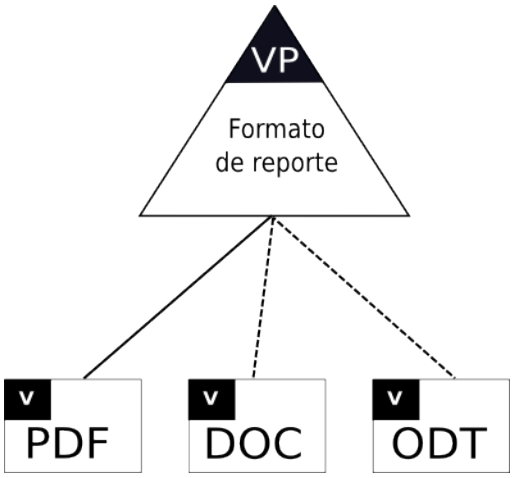

Figura 1.1: Notación empleada por (Lauenroth y Pohl, 2005) para representar los puntos de variación y las variantes.
Los puntos de variación se representan con triángulos y las variantes con rectángulos. Las líneas relacionan los puntos de variación con sus variantes. Una línea discontinua significa que esa variante es opcional, una línea completa indica una variante obligatoria.

Algunos investigadores (Hotz y otros, 2006) han demostrado que la representación formal de la variabilidad en los artefactos de una LPS es vital para su éxito.

Otros autores han trabajado en la aplicación práctica de modelos formales para la representación de la variabilidad, y reportan resultados alentadores (Sinnema y Deelstra, 2008). Estos autores presentan el modelo COVAMOF (Deelstra y Sinnema, 2008). Este modelo no es esencialmente diferente al que se describe en (Lauenroth y Pohl, 2005) pero sí ha tenido un soporte básico de herramientas (Sinnema y otros, 2004). Si

bien las herramientas presentadas solamente cubren algunas tecnologías, esto es suficiente para que empresas que usen esas tecnologías puedan aplicar el modelo.

El resultado fundamental que se encuentran en estos trabajos es que tener un modelo formal de la variabilidad de los productos produce incrementos en la productividad de la línea, y también que se reduce la necesidad de expertos para derivar productos a partir de los activos de la LPS (Sinnema y Deelstra, 2008).

1.2. Gestión de proyectos en las LPS

En esta sección se realiza una revisión de los conceptos de gestión de proyectos en el contexto de las LPS. Se discute en primer lugar la posición del Instituto de Gestión de Proyecto (PMI, siglas del inglés *Project Management Institute*) y se hace una revisión de las necesidades particulares de una LPS.

El PMI es una organización que se ha convertido en puntera en las prácticas de gestión de proyectos. En los últimos años ha promovido la publicación de una extensiva base de conocimientos sobre gestión de proyectos que intenta agrupar las mejores prácticas alrededor de este tema. Su «Guía de los Fundamentos de la Dirección de Proyectos» (Project Management Institute, 2004, 2008) conocida como guía del PMBoK (*Project Management Base of Knowledge*) es especialmente reconocida y estudiada.

En este texto se caracteriza un proyecto por ser un *esfuerzo temporal* para construir un *producto, servicio o resultado único*. Esto implica que los proyectos poseen:

- puntos de comienzo y fin bien definidos,
- metas y objetivos claros,
- requisitos de tiempo y recursos establecidos.

En la opinión de los promotores de las LPS, los proyectos en una LPS no siempre cumplen con estas características (Clements y otros, 2005). Estos autores indican que:

- En una LPS se pueden establecen proyectos para la creación de activos reutilizables en los productos de la línea. Estos proyectos satisfacen la definición del PMBoK.

  Estos proyectos pueden o no ser los responsables de evolucionar los activos creados; o se pueden crear proyectos específicamente para la evolución de estos activos.

  Por otra parte, indican los autores, que es más común pensar en toda la producción de los activos como un único proyecto; y que este «proyecto» no cumple con la característica de tener límites temporales definidos sino que existe a la par de la LPS mientras sea necesario crear o mantener los activos.
- De la misma manera plantean que el desarrollo de todos los productos dentro de una LPS suelen concentrarse en único proyecto cuyo propósito es supervisar la producción de cada producto. Este proyecto tampoco tiene un fin definido y se extiende durante toda la vida de la LPS.

Figura 1.2: Actividades de una línea de productos de software. Tomado de (Clements y otros, 2005).

- Finalmente indican la existencia de un nuevo tipo de proyecto que se concentra en tareas de gestión organizacional para asegurar que los productos utilicen los activos al máximo, y que estos satisfagan altos estándares de calidad. Esto puede implicar la creación de unidades de trabajo que están fuera del alcance de cualquiera de los proyectos anteriores.

En la figura 1.2 se ilustran las tres actividades que, de forma muy abstracta, describen Clements y Northrop. Esta intenta dar un argumento a las objeciones de estos autores sobre la definición de proyectos dentro de las LPS según se han descrito en los párrafos anteriores. Las actividades se representan de forma cíclica para ilustrar la idea de un continuo en estos «proyectos», y reflejar también la fuerte inter-relación que existe entre estos procesos.

A diferencia de estos, los autores de (Pohl y otros, 2005) agrupan las actividades de una LPS en dos procesos ingenieriles: Ingeniería de dominio y de sistema; aunque también atienden las necesidades de gestión específicas de una LPS. Esencialmente esta división no difiere de la descrita más arriba. Se puede establecer una equivalencia directa entre las actividades de desarrollo de activos y la ingeniería de dominio, y entre el desarrollo de productos y la ingeniería de sistemas.

En la opinión del autor de este documento, dentro de una LPS pueden existir proyectos bien definidos. Sin embargo, concuerda con los autores del SEI[4] en que la misma existencia de la LPS implica un ciclo continuo de producciones que no tiene (desde un punto de vista exterior) límites temporales definidos.

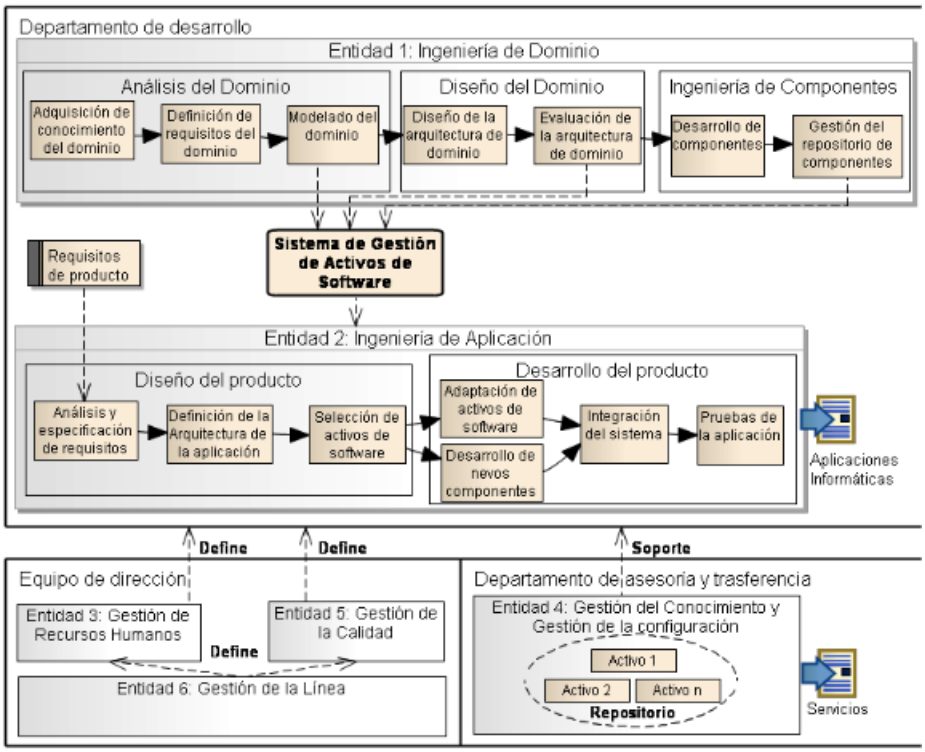

Figura 1.3: Modelo de LPS centrado en la arquitectura de empresas. Tomado de (Pestano y Piñero Pérez, 2011).

No obstante cuando se comienzan desarrollos de activos, y en los inicios de una LPS se pueden definir fases claras de evolución de la LPS y organizarlas en proyectos bien definidos.

Es importante notar, que las agrupaciones dadas aquí no son contrapuestas a las dadas por el PMI. En el PMBoK se identifican cinco grupos de procesos y nueve áreas de conocimientos y estos son aplicables a proyectos de diversa índole. El SEI, al concentrarse mucho más en proyectos de desarrollo de sistemas con uso intensivo de software, puede identificar grupos de procesos más específicos. En particular se puede observar que las actividades de desarrollo de activos y las de desarrollo de productos se pueden calificar dentro del grupo de procesos de ejecución. En el resto de esta sección, siempre que sea apropiado se señalarán los elementos del PMBoK que sean aplicables al contexto específico de una LPS.

En una revisión más profunda, (Pestano y Piñero Pérez, 2011), describe un modelo de LPS que abarca mucho de los aspectos brindados pero, además, basa en modelo en la arquitectura de empresas. En la figura 1.3 se brinda una representación de este modelo. En este modelo permite también la creación de proyectos para el desarrollo de los activos y otros para el desarrollo de productos.

En las siguientes secciones se explica con mayor profundidad el modelo de (Northrop y Clements, 2007a). Más adelante, en la sección 1.2.4, se expone el modelo de (Krueger, 2007). Luego se abunda en el paralelo de estos elementos con el modelo de (Pestano y Piñero Pérez, 2011).

[4] Instituto de Ingeniería de Software de la Universidad Carnegie Mellon, la siglas son por su nombre en inglés *Software Engineering Institute*.

1.2.1. Actividades de desarrollo de activos

En (Clements y otros, 2005) se identifican como entradas del proceso de desarrollo de los activos a:

- *Restricciones del producto*: ¿Qué deben tener en común los productos y qué los distingue entre sí? ¿Cuales son las funcionalidades que deben exponer? ¿Qué funcionalidades serían beneficiosas introducir según los análisis de mercado y las predicciones tecnológicas? ¿Cuáles son los requisitos de calidad para estos productos? Estas son algunas de las preguntas que pueden conducir a la definición de las restricciones sobre los productos (Northrop y Clements, 2007b).
- *Restricciones del proceso de producción*: ¿Cuándo los productos que actualmente se deben construir a partir de los activos deben estar terminados y para qué clientes se desarrollan? ¿Qué procesos se deben seguir en el desarrollo de los activos?
- *Inventario de activos pre-existentes*: ¿Cuáles componentes o sistemas legados o construidos por terceros están disponibles para ser usados como activos? Otra posibilidad puede ser que no se tengan los activos completamente listos, pero que existan sistemas legados que puedan ser minados para construir los activos necesarios (Northrop y Clements, 2007b).
- *Estrategia para la producción*: La forma general para construir los activos y productos.

  La estrategia afecta de forma significativa cómo son gestionados los proyectos dentro de la línea. Se pueden identificar dos grupos: las *pro-activas* en las cuales se construyen los activos y luego los productos se ensamblan, y las *reactivas* en las cuales se comienza con los proyectos y los activos se construyen generalizan los componentes desarrollados para los productos.

  La estrategia de producción de una LPS, además, «describe factores de producción críticos para el éxito de la línea e indica alternativas generales sobre cómo actuar sobre esos factores» (Chastek y otros, 2009, p. 1). Según (Northrop y Clements, 2007b) la estrategia de producción dicta la génesis de la arquitectura y sus componentes, y su línea evolutiva.

En el proceso de desarrollo de activos, además de los activos creados, también se obtiene:

- *Una definición del alcance de la línea de productos*, que es una descripción de los productos potenciales que se pueden construir. En el texto del alcance se suele incluir aquellas características que son comunes a los productos y también la forma de variar los productos. El alcance debe ser suficiente para identificar sin ambigüedad si un producto (a partir de su enunciado) pertenece o no la línea.
- *Un plan de producción* en el cual se describe paso a paso cómo deben ser construidos los productos a partir de este conjunto de activos.

Evidentemente esto implica que el desarrollo mismo de los activos modifica dialécticamente sus entradas; de ahí que se dibujen en ciclos que se intersecan los procesos de gestión y desarrollo en la figura 1.2.

1.2.2. Desarrollo de los productos

Los productos se «instancian» a partir de los activos generados en la actividades de desarrollo de activos. Se puede identificar como entradas en estos proyectos a:

- *Los requisitos del producto en cuestión.* Si es pertinente esto se puede desarrollar a partir de un modelo de dominio, y un listado de requisitos típicos para los productos de la LPS.

  Quizá en este proceso las actividades fundamentales no sean la elicitación de *nuevos* requisitos, sino aquellas centradas en la comunicación de los requisitos soportados por la LPS y su aprobación (Referirse a (Nuseibeh y Easterbrook, 2000) para una introducción sobre ingeniería de requisitos).

- *El alcance de la línea de productos.* Una vez más, esta definición puede usarse tanto para (1) comprobar que el producto solicitado se puede construir dentro de la LPS, y para (2) generar un alcance del producto en cuestión.

  Un listado de requisitos con anotaciones de variabilidad también puede utilizarse.

- *El conjunto de activos.* Este es tal vez la entrada obvia en este proceso. No obstante, es importante dejarla explícita, dado que dentro de los activos, habrá un grupo importante que habrá que seleccionar para construir el producto.

  La trazabilidad entre los requisitos de una línea de productos y su realización a través de la combinación de activos es una actividad que necesita ser gestionada y esto tiene implicaciones tanto en la forma en que se utilizan y se producen los activos.

- *El plan de producto.* En este plan de detallan cómo ensamblar los activos para construir un sistema; también puede incluirse las limitaciones actuales, o referencias a planes subsidiarios como los planes de pruebas, plan de aseguramiento de la calidad, etc.

En una línea en construcción, con una estrategia reactiva, este proceso sería el iniciador de la construcción de muchos de los activos de la línea; o, al menos, este proceso añadiría presiones de cambio sobre los activos ya construidos para adecuarlos al producto en cuestión. Por tanto el proceso de *control del alcance* (Project Management Institute, 2004, 2008, cf. 5.5) donde se aprueban los cambios debe ser generalizado para el caso de las LPS: El PMI identifica los cambios dentro del alcance de un proyecto, sin embargo, dado que en las LPS los cambios dentro de un activo se transfieren a muchos productos, los cambios a los activos deben ser aprobados cuidadosamente.

1.2.3. Las actividades de gestión de la LPS

El éxito de una LPS depende en gran medida de una coordinación efectiva entre el desarrollo de activos y los productos. Los productos deben utilizar a aquellos a la máxima extensión siempre que sea práctico. A su vez, los activos deben constituir artefactos de alta calidad construidos de forma predecible en el tiempo.

Más allá de esto, las LPS necesitan una *dirección estratégica*[5]. La dirección de la LPS necesita estar constantemente invirtiendo recursos para el desarrollo y mantenimiento de los activos y catalizando el cambio cultural necesario para «visualizar» los productos en términos de la aplicación de activos.

Esto implica un número de responsabilidades entre las que se pueden mencionar:

- Asegurar un organigrama apropiado para la LPS, con el personal y recursos apropiados.
- Mantener y gestionar el entrenamiento del personal.
- Establecer políticas y procesos necesarios que, desde una perspectiva integradora, definan cómo es el trabajo diario en la LPS.
- Planificar la transición hacía el modo de líneas de productos de software.

Este último punto, es de especial importancia en las líneas en creación. Cualquier esfuerzo de crear una LPS en la dirección no se involucre directamente está condenado y tarde o temprano fracasa. La creación de una LPS es una decisión que *no es técnica* sino estratégica.

1.2.4. El modelo de tres niveles de una línea de productos de software

El modelo de los tres niveles propuesto por Krueger (Krueger, 2007), es una alternativa ligera al modelo de Clements y Northrop (Clements y Northrop, 2001; Northrop y Clements, 2007a) explicado con anterioridad. Krueger indica que estos primeros modelos de adopción de las LPS son muy complejos, y se necesita de mucho trabajo para lograr determinar las interrelaciones entre las actividades de ingeniería del dominio y del sistema; y también para comprender cómo los cambios propuestos determinaban los beneficios indicados (Krueger, 2007).

El modelo propuesto se organiza en tres niveles los cuales indican un grupo de capacidades y beneficios concretos. Cada nivel se basa en las capacidades logradas con el nivel anterior. Esto significa que la aplicación de un nivel ya provee beneficios directos, pero además, habilita capacidades más estratégicas en niveles superiores. En cada nivel existe un problema a resolver y una solución dada por el modelo. Los niveles se puede describir con:

- El problema que se debe resolver
- Los roles o trabajadores que intervienen
- La solución prevista por el modelo
- Los beneficios obtenidos al aplicar la solución

[5] En el original se utiliza la frase «*strong, visionary management*».

El nivel básico es «La gestión de la variabilidad, y la producción automatizada». En este nivel el problema que se intenta resolver es que se gasta mucho tiempo en la corrección de defectos o el mantenimiento, y no queda tiempo para añadir valor a los productos. La fuente de este problema es que se usan mecanismos de variación ad-hoc, la producción es totalmente manual y pueden existir código duplicado o código muerto. La solución propuesta es la utilización de técnicas de variabilidad de primer nivel. El beneficio esperado es la optimización de la productividad y la reducción de los costos de producción.

El nivel medio es «El desarrollo centrado en los activos». En este nivel el problema que se intenta resolver es la inoperancia de los equipos centrados por productos; los cuales pueden perder oportunidades de reuso y tener cronogramas y costos más elevados. La solución que se propone es la creación de equipos por activos y no por productos. El beneficio esperado es la mejora en la calidad de los productos mediante la reducción de errores, y por tanto la reducción de los tiempos necesarios en pruebas.

El nivel más alto es la «Evolución del Portafolio basada en Funciones». En este nivel el problema es que el portafolio basado en productos limita la capacidad real de la LPS y reduce sus utilidades, además de que coarta la cantidad de productos que se pueden mantener. La solución propuesta es que el portafolio debe centrarse en especificaciones de funciones, junto con perfiles de funciones para los productos. La funcionalidad es primer elemento en el portafolio; los productos son secundarios. Los beneficios que se esperan son la reducción del tiempo para el mercado y la expansión de la línea misma.

Es de notar que en este modelo no existe un proceso de creación de productos, ni de ingeniería de sistemas. Los autores indican que la construcción de productos debe quedar automatizada en el nivel básico, y que se deben introducir mecanismos *configuradores de productos* para la automatización de este proceso.

En la opinión del autor de este documento, si bien este modelo es más fácil de adoptar que las prácticas expuestas en (Clements y Northrop, 2001; Northrop y Clements, 2007a), también presenta sus retos, puesto que hay que adquirir (a riesgo) una herramienta de configuración de productos, o desarrollarla internamente.

Por otra parte, el modelo dado en (Pestano y Piñero Pérez, 2011), es suficientemente flexible como que su adopción es tan simple como el de Krueger. En este sentido, al comenzar construyendo las entidades de Ingeniería del Dominio, y dando responsabilidades de construcción de productos a grupos ad-hoc, se puede alcanzar un nivel similar al primero. Por esta razón en esta tesis, si bien se basa fundamentalmente en Pestano, se han utilizado ideas de ambos modelos, las cuales se expondrán más adelante, en donde sea pertinente.

1.2.5. Seguimiento y control en una LPS

La gestión de una LPS no estaría completa sin mecanismos de seguimiento y control de sus proyectos. En (Zubrow y Chastek, 2003) se identifican tres *áreas de responsabilidad* que la dirección de la organización debe cubrir. En estas áreas de responsabilidad los autores identifican varios roles y le asignan actividades de seguimiento y control. Las áreas propuestas son:

- La gestión de la línea de productos desde una perspectiva global de la empresa.
- La gestión del desarrollo de los activos, que incluye también el mantenimiento de estos activos y el desarrollo de la infraestructura de los productos.
- La gestión del desarrollo y mantenimiento de los productos individuales.

Se puede notar que estas tres áreas de responsabilidad se alinean directamente con las tres dimensiones dadas por Clements y Northrop (Clements y Northrop, 2001; Northrop y Clements, 2007a), y también se corresponden con el modelo de adopción de Krueger (Krueger, 2007).

Aunque Zubrow y Chastek reconocen que pueden existir múltiples formas de manejar una LPS, enfocan su discusión basados en la existencia de, al menos, tres roles:

- el gestor de la LPS (*product line manager*),
- el gestor del desarrollo de activos (*asset development manager*), y
- varios gestores para el desarrollo de productos (*product development manager*).

Parte de las responsabilidades que se concentran en estos roles son la definición del alcance de la LPS, y el seguimiento y control de los procesos de la LPS. En este sentido, estas tres áreas de responsabilidades tienen impacto en los grupos de procesos de inicio, planificación, seguimiento y control y cierre de los proyectos dentro de una LPS.

En tanto al seguimiento, los autores de (Zubrow y Chastek, 2003) identifican tres categorías de mediciones que pueden (deben) realizar estos gerentes, las cuales son métricas relativas a:

- La *eficiencia* de la línea (*performance*), la cual caracteriza el grado en que la línea cumple con sus objetivos de tiempo, costo y calidad en comparación con una línea base relativa a la forma tradicional de desarrollo por productos.
- La *conformidad* (*compliance*) caracteriza el grado en que los proyectos utilizan los procesos, prácticas y estándares diseñados para la explotación y reutilización de los activos de la línea.
- La *efectividad* de la línea (*effectiveness*) caracteriza el grado en la LPS completa cumple con sus objetivos globales.

Los autores dan luego un grupo ilustrativo (y no exhaustivo) de métricas. Utilizan una matriz que clasifica las métricas por las categorías mencionadas en la lista anterior y por el rol "consumidor" de esta métrica. La siguiente tabla recoge un grupo de estas:

Cuadro 1.1: Métricas brindadas por (Zubrow y Chastek, 2003) para las líneas de productos de software

Objetivo	**Gestor de la línea**	**Gestor de desarrollo de activos**	**Gestor de producto**
Eficiencia	Costo total de desarrollo de productos Tendencia en densidad de defectos Productividad	Costo de producción de los activos Densidad de defectos en activos Desviación del cronograma	Costo *directo* del producto Densidad de defectos en los artefactos de los productos.
Conformidad	*Mission focus* Conformidad con la arquitectura	*Mission focus* Conformidad con los procesos	Conformidad con los procesos
Efectividad	Retorno de la inversión Satisfacción del mercado Cubrimiento de las características (demandadas) del mercado	Utilidad de los activos Costo de uso de los activos Por ciento de reutilización.	Satisfacción del cliente

Véase la fuente (Zubrow y Chastek, 2003) para la definición completa de estas métricas. Nótese además, que estas no son todas las métricas que se pueden utilizar en el contexto de una LPS; en particular, se nota que no se dan métricas relacionadas con el desarrollo de los recursos humanos.

1.3. Desarrollo basado en componentes

Aunque el desarrollo basado en componentes no es en sí mismo un modelo de desarrollo de software, el autor considera importante realizar una revisión de este tema dado a su ubicuidad en las líneas de productos de software. Las LPS son, en efecto, un modelo de desarrollo basado en componentes (aunque no siempre que se desarrolla basado en componentes se está en el contexto de una línea de producto de software).

En (Crnkovic y Larsson, 2002) se brinda una introducción al tema de la ingeniería basada en componentes (IBC); la cual, de forma similar a las LPS, intenta construir sistemas mediante el ensamblaje de componentes previamente fabricados. A diferencia de las LPS, la IBC se limita a componentes de software. Sin embargo, como en las LPS también existen los componentes de software, mucho del conocimiento generado en aquella disciplina es «heredado».

En el mismo libro se brindan algunos riesgos que se pueden asumir con el desarrollo basado en componentes, los cuales también pueden ocurrir en las LPS. Dos de estos riesgos son de interés particular para las LPS en construcción:

- *Costo de mantenimiento de los componentes.* Aunque los costos de mantenimiento de las soluciones se ven reducidos, los relativos al mantenimiento de los componentes tienden a elevarse. Esto es así porque, al ser utilizados en varios contextos, los componentes deben responder a una mayor variedad de requisitos y se necesitan emplear más recursos para mantener la trazabilidad, y la adecuación de estos componentes a un universo mayor de consumidores.

  Por lo general es más rápido (y por tanto menos costoso) hacer un componente específico que uno general. En una LPS hay que evaluar cuidadosamente qué componentes serán reutilizados en varios de los productos. No se debe intentar generalizar aquello es particular de un grupo reducido de productos.

  Las métricas sobre el costo directo del desarrollo de un producto, y también las métricas de desarrollo de componentes y su por ciento de reutilización pueden ayudar a mitigar este riesgo.

- *Se requiere tiempo y esfuerzo para desarrollar componentes reutilizables.* Como los componentes a crear deben desarrollarse con requisitos no funcionales que aumenten su reutilización, se puede incrementar el tiempo inicial de desarrollo de este tipo de componentes.

  También esta es una decisión de compromiso. Se espera que esta mayor inversión en tiempo se revierta en una reducción en el tiempo necesario para construir nuevos productos basados en estos componentes.

El libro menciona otros riesgos que el autor considera importantes, pero que de alguna medida son mitigados siguiendo buenas prácticas dentro de una LPS.

No obstante, el autor quisiera añadir el problema de la visualización de la utilidad de este tipo de desarrollo. Los componentes en este nivel resuelven, por lo general, problemas *de diseño*. Esto complica la comunicación a gerentes y patrocinadores de cómo se están invirtiendo los recursos durante el proyecto. Una jerarquía de activos siguiendo una taxonomía del dominio (o negocio) puede ayudar en este sentido.

Además de estos riesgos otras dificultades siguen estando presentes:

- No existe consenso sobre qué es y cómo especificar un componente (Crnkovic y Larsson, 2002, p. XXXI).
- El ciclo de vida de los sistemas basados en componentes, es por lo general, separado del ciclo de vida de los componentes que lo forman. Esto crea, entre otros problemas, uno particular con el seguimiento de las versiones de un componente que hay en explotación. En esencia este el mismo asunto que tratan los autores de (Clements y otros, 2005) cuando explican los ciclos de vida de los proyectos dentro de una LPS.

Finalmente, el principal riesgo para la creación de una LPS es que la dirección no se comprometa con este esfuerzo y lo perciba como una acción técnica, cuando en realidad es una decisión estratégica.

1.3.1. Sistemas basados en componentes

En (Crnkovic y otros, 2002) se brindan conceptos básicos sobre la ingeniería de software basada en componentes. En (Lüders y otros, 2002), por otra parte se brinda un modo de especificar los componentes a partir de la descripción de sus interfaces.

Los conceptos brindados por estos autores permiten la creación de modelos cuyos principales beneficios son:

- La posibilidad de intercambiar componentes desarrollados por diferentes equipos para distintos productos.

  Esta posibilidad es potencial y no una garantía. Depende en gran medida de las herramientas para la descripción de los contratos que expone el componente, y también de las restricciones arquitectónicas que se establezcan de forma efectiva (Lüders y otros, 2002).

- Un lenguaje común para los desarrolladores, de modo que los artefactos de diseño sean codificables (por ejemplo en un perfil específico de UML).

En el mismo libro, se dedican varios capítulos al tema de la arquitectura de sistemas basados en componentes. En particular (Bosch y Stafford, 2002) afirman que la arquitectura y los componentes de un sistemas están estrechamente relacionados, y por esta razón, no se puede analizar un sistema basado en componentes sin revisar la arquitectura del mismo. De lo que se infiere que no se puede hablar de los componentes de una LPS sin referirse a su arquitectura.

Los autores arremeten contra los componentes comerciales y los caracterizan por ser complejos, idiosincrásicos e inestables. Se cuestionan incluso si existe un mercado para tales tipos de componentes. Sin embargo, debemos considerar componentes comerciales exitosos: la VCL de Borland Delphi, el .NET de Microsoft, CORBA, GTK, QT, y otras muchas colecciones de componentes útiles para la creación de aplicaciones.

Es importante notar que varios de los mencionados aquí son componentes libres y otros no; algunos siguen procesos públicos para la gestión de cambios y otros son cambiados de forma unilateral por el fabricante. Estos elementos afectan en gran medida la gestión de configuración y por esa razón la decisión de usar un componente comercial (o realizado por terceros) debe ser firme y analizarse bajo el prisma de la gestión de riesgos y otros áreas de la gestión de proyectos:

- ¿Cual es la garantía real de mantenimiento de la librería? ¿Existen casos de estudio de empresas con similares necesidades?

- ¿Se tiene la capacidad técnica y/o financiera para asumir el desarrollo de este componente y evolución para la LPS en caso de que la comunidad abandone el componente?

- ¿Cuánto costosa sería la integración del componente como un activo en la LPS? ¿Cuán complejo sería la prescripción de la realización de productos basados en él?

- ¿Se ha utilizado antes este componente de forma exitosa? ¿Qué lecciones se han aprendido?

Este tipo de preguntas pueden darle contexto a una decisión sobre la selección de un componente desarrollado por terceros. De cualquier manera, se puede re-utilizar la experiencia posible que tenga el equipo con esos componentes en proyectos anteriores e incorporar y formalizar este conocimiento como activos de entrenamiento y capacitación e incorporarlos en los procesos de desarrollo de capital humano.

En la literatura el autor no ha encontrado referencia a activos que tengan relación con la gestión de recursos humanos más allá de los que tienen que ver con los organigramas. Sin embargo, en el contexto de una universidad productiva, es evidente que deben existir sólidos programas de entrenamiento orientados a facetas particulares de la LPS donde se implique cualquier persona.

1.4. Arquitecturas y estilos arquitectónicos

En esta sección se introducen los conceptos de arquitectura y otros relacionados con esta disciplina. Se hace una revisión de los principales estilos, vistas y métodos de representación.

Al brindar los conceptos relacionados con la arquitectura de software, esta sección sienta las bases para los elementos técnicos explicados en el capítulo 2.

1.4.1. Evolución de la arquitectura de software

La disciplina de la arquitectura de software fue establecida como tal durante la última década del siglo XX. En un artículo de gran influencia Dewayne Perry y Alexander Wolf preveían que la década del 1990 sería la «década de la arquitectura de software» (Perry y Wolf, 1992).

En la figura 1.4 se muestra una línea del tiempo de los resultados encontrados por Google desde 1950 hasta 2010 para la búsqueda «software architecture». Se nota cómo es precisamente a partir del período previsto por Perry y Wolf que aumenta considerablemente la actividad sobre esta disciplina emergente.

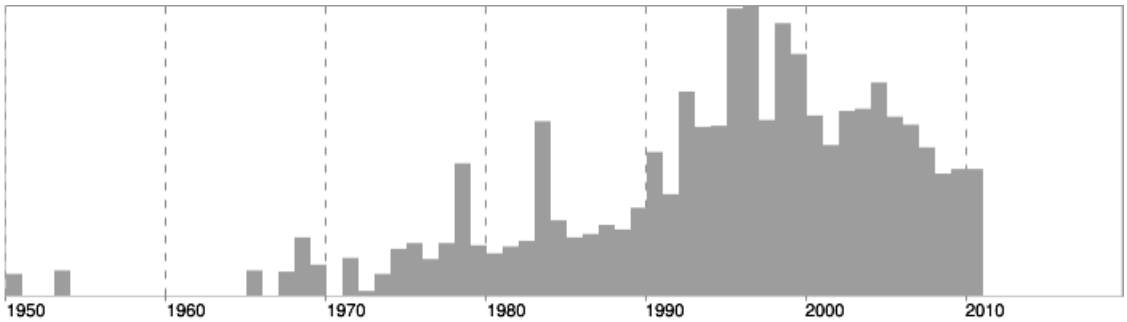

Figura 1.4: Línea del tiempo de referencias construida por Google para la frase «software architecture».

Garlan y Shaw en (Garlan y Shaw, 1994, pp. 3–5) realizan un resumen histórico del surgimiento de la disciplina como tal. Estos autores cubren desde los años 50 hasta el inicio de la década del 90. Básicamente revisan el surgimiento inicial de los lenguajes de programación en los años 50 y los tipos de datos abstracto a finales de los 60. Con la aparición de la programación estructurada (Dahl y otros, 1972) en la década del 70, y también el surgimiento de la programación orientada a objetos en el lenguaje Smalltalk, comienza una sistematización en el *diseño* de los sistemas.

Luego de los años 80 es que la arquitectura comienza a distinguirse del diseño de sistema. Es precisamente después del año 1990 que se introduce y sistematiza el término «estilo arquitectónico» en el contexto de la ingeniería de software. En la figura 1.5 se muestra una línea de tiempo similar a la anterior, pero esta vez para la frase «architectural styles, software».

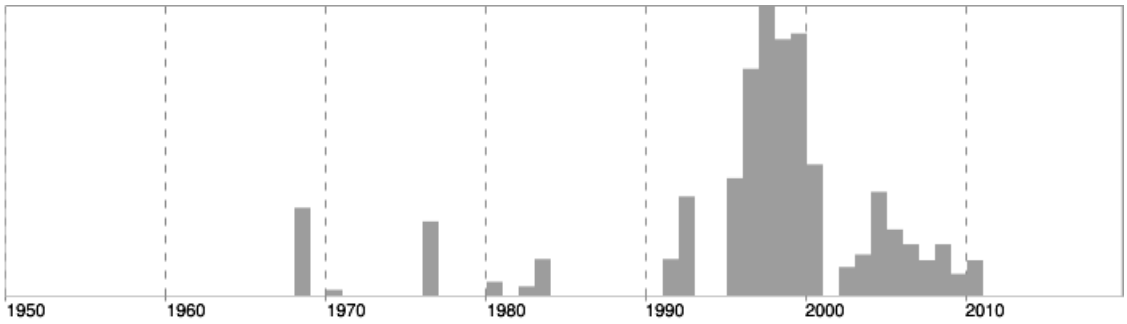

Figura 1.5: Línea del tiempo de referencias construida por Google para la frase «architectural style, software».

La aparición de este concepto es, en la opinión del autor, el indicativo real sobre el surgimiento de una nueva disciplina dentro de la ingeniería de software. Alrededor de este concepto se centra el principal esfuerzo que se hace inicialmente sobre la arquitectura de software. Esta afirmación es visible en la línea de tiempo; en la cual se nota que la mayor parte del trabajo sobre estilos se concentró durante la mitad final de la década de los 90. De hecho, una inspección más minuciosa de las referencias durante los años 2000–2010 indica que la mayoría de los estilos fueron identificados con anterioridad y que las investigaciones durante este último período se concentran en otros aspectos pero no identifican nuevos estilos.

En los siguientes epígrafes se abunda sobre el concepto de arquitectura de software y sobre los estilos arquitectónicos.

1.4.2. Arquitectura de software

En su artículo Perry y Wolf (Perry y Wolf, 1992) no dan una definición completa de arquitectura sino que la caracterizan en el contexto del sistema de software como la parte del proyecto que se dedica a identificar los elementos arquitectónicos, sus interacciones, y las restricciones impuestas a ambos para satisfacer los requisitos y, a la par, servir como base para el diseño. Luego clasifican los elementos en tres tipos:

- elementos de procesamiento,

- elementos de datos, y
- elementos de conexión.

Los elementos de procesamiento son definidos como los que realizan transformaciones a los datos. Los elementos de datos son los que contienen la información que se usa y/o transforma. Los elementos de conexión son aquellos que habilitan la comunicación.

Perry y Wolf completan su modelo indicando que además de los elementos, la arquitectura tiene una *forma* definida por la ponderación de sus propiedades y relaciones. Esta ponderación puede indicar una de dos cosas: la importancia del elemento dentro de la arquitectura, o la posibilidad de elegir entre alternativas, unas más apreciadas que otras. Finalmente, consideran que en la arquitectura también debe incluirse los *elementos de juicio* para tomar las decisiones hechas.

Fielding, para su trabajo con REST (Fielding, 2000; Webber y otros, 2010), toma como base el trabajo de Perry y Wolf, sin embargo, elimina los *elementos de juicio* como un elemento dentro de la arquitectura en sí misma. Argumenta que las propiedades de la arquitectura no varían si se elimina este aspecto. Fielding se concentra en la arquitectura como una vista del sistema en *tiempo de corrida* y, aunque reconoce la importancia de documentar las decisiones, no incluye la documentación como parte de la arquitectura en sí misma. Para él, la arquitectura es inherente al sistema y no se encuentra fuera de este; esté o no documentada correctamente. En sus palabras si la documentación de la arquitectura de un sistema «se quema» esto no implica el colapso inmediato del sistema. Reconoce, no obstante, el efecto negativo que puede causar la escasez de documentación, especialmente cuando el sistema se expone ante la demanda de cambios. En los últimos años, sin embargo, hay más exponentes sobre la importancia de considerar los elementos de juicio, o decisiones arquitectónicas como elementos de primer nivel en las arquitecturas (Zalewski y Kijas, 2010).

Varios autores del SEI (Bass y otros, 2003) abarcan, además de los elementos antes expuestos, los procesos en los cuales se desarrolla una arquitectura de software. En su argumentación indican que una única estructura no es suficiente desde el punto de vista ingenieril (Bass y otros, 2003, cf. 2.2) y que es inherente a la arquitectura la existencia de varias estructuras, las cuales, de conjunto, *exponen* la arquitectura del sistema, pero ninguna de ellas lo es por sí misma. A diferencia de Fielding, Bass y otros critican las definiciones que se concentran en los aspectos de tiempo de corrida. Arguyen que esta visión deja de tener en cuenta aspectos relevantes como la posibilidad de dividir el trabajo en particiones, que puede tener relevancia en la modificabilidad futura del sistema, y también en la entrega a tiempo; así como influye también en los procesos de pruebas.

Estos autores organizan la arquitectura en *vistas*:

- Las concernientes a los módulos. Los elementos de esta vista son los módulos, que son unidades (básicas) de implementación; y permiten capturar información sobre las responsabilidades de un módulo y las dependencias entre ellos.

- Las que abarcan las estructuras de componentes y conectores (C&C). Esta sería equivalente a la que promueve Fielding. Aunque este último hace énfasis en la necesidad de descripción de los datos. En el caso del sistema arquitectónico presentado por Bass y otros, la naturaleza de los datos puede ser extraída del resto de las vistas.
- Las vistas que detallan las estructuras de ubicación. Sirven para presentar las relaciones que existen entre los elementos del software y el ambiente externo. Esto incluye, si es necesario, los procesadores en los cuales corre cada componente, en qué archivo es almacenado durante las distintas fases del proyecto, etc.

En (Bass y otros, 2003; Clements y otros, 2002) también se identifican un grupo de *estilos arquitectónicos* representados por una selección particular de los elementos de cada una de las vistas dadas. Fielding (Fielding, 2000) también identifica muchos de estilos, aunque se concentra más en aquellos que son posibles en las vistas C&C.

Las posiciones de Fielding y las de Bass y Clements son radicalmente diferentes, pero pueden ser explicadas por los tipos de sistemas que cada uno intenta representar/explicar. Fielding (Fielding, 2000) está más interesado en la presentación arquitectónica de sistemas hiper-media descentralizados, mientras que Bass y otros (Bass y otros, 2003) tienden a enfocarse a sistemas empotrados.

Es comprensible entonces que Fielding se interese más por aquellas vistas (tomando el concepto de Bass y otros, 2003) de componentes y conectores con el aditivo de la descripción de los datos; debido a que estas vistas son las que exponen mejor las características críticas de los sistemas hiper-media descentralizados. Sin embargo, es factible pensar que el diseño de estos sistemas, las vistas de módulos (especialmente dentro los límites de los subsistemas) y las vistas de ubicación pueden ser útiles y en ocasiones necesarias.

Otros autores indican que la arquitectura (de muchos tipos de sistemas) está dada por un grupo de estructuras, cada una definida por componentes de cierto tipo y sus relaciones (Klein y Weiss, 2009, p. 5). A diferencia de Bass y otros, Klein y Weiss capturan la arquitectura de software en dos tipos de decisiones de diseño: decisiones estructurales y de comportamiento. Klein y Weiss identifican las siguientes estructuras arquitectónicas:

- Aquellas relativas al ocultamiento de la información; donde los componentes son los *módulos* que pueden ser asignados a los desarrolladores. Los atributos que se pueden comprobar con esta estructura son los de modificabilidad, modularidad y la propiedad de ser realizable.

  Esta estructura es la misma que (Clements y otros, 2002) definen como la vista de módulos.

- Las estructuras de uso; donde los módulos son los elementos de software en la misma *fase de resolución*[6]. Cuando el sistema está en operación (tiempo de corrida) se dice que un componente usa a otro si se necesita que el segundo esté corriendo correctamente para el correcto funcionamiento del primero.

  Los autores reconocen que en ocasiones los arquitectos no toman toda las decisiones sobre las posibles relaciones de uso entre los componentes, pero sí establecen restricciones o jerarquías de «uso permitido».

En esta estructura se mezclan elementos de las vistas de C&C y módulos dadas por Clements y otros.

- Las estructuras de procesos. Esta vista intenta capturar el dinamismo del sistema en tiempo de ejecución. Los autores identifican relaciones como «dar a trabajo a», «obtener recursos de» y «compartir recursos con»; las cuales argumentan que son esenciales para analizar posibles condiciones de carrera[7].

  Esta estructura es un caso especial de la vista de ubicación dadas por (Clements y otros, 2002).

- Las estructuras de acceso a datos. Los componentes en esta vista son los programas y los segmentos de datos, la relación que se establece es si un programa *tiene acceso a* un segmento de datos.

  Los autores indican que esta estructura permite visualizar la seguridad del sistema. Sin embargo, se debe notar que esto es solo una parte de la seguridad. Un programa puede tener acceso a un segmento de datos[8] pero la seguridad de cada dato está dada por el contexto local en tiempo de corrida.

Para el desarrollo de este trabajo, se utilizará un grupo (no completo) de las estructuras descritas en Bass y otros. Se dará mucha importancia a las estructuras de componentes y conectores, pues, como se verá más adelante, capturan muchas de las decisiones importantes para la LPS PATDSI, y también son estas las decisiones que pueden tomarse en fases iniciales del proceso de creación de la línea.

1.4.3. Estilos arquitectónicos

Los estilos arquitectónicos son una forma de capturar formas recurrentes para la organización de un sistema, sus estructuras posibles y tipos de relaciones. Otros autores utilizan el término patrón arquitectónico para definir un concepto muy similar. Si bien ambos no son intercambiables del todo, en ambos bandos se observan los mismos patrones/estilos, incluso con idénticos nombres.

A continuación se describirán varios de los estilos que, en la opinión del autor, son más relevantes tanto para el problema en cuestión como por su frecuente aparición en la literatura. Por lo general se tomará la definición dada en (Buschmann y otros, 1996)[9], aunque se mencionarán otras fuentes cuando se necesario.

Tuberías y filtros

Este estilo identifica un único tipo de componente y un único tipo de conector. Es un estilo que se describe como una vista C&C. Son útiles para sistemas que procesan flujos (continuos) de datos. Los componentes

[6] En el original se utiliza la expresión «*binding time*» que se refiere al momento en que un símbolo cualquiera dentro de un programa obtiene el valor que le corresponde. Dependiendo del tipo de plataforma utilizada hay resolución en tiempo de corrida, en tiempo de enlace y en tiempo de compilación.

[7] Otras personas opinan que la concurrencia de los sistemas se modela mejor con abstracciones diferentes a las de hilos, y procesos; tales como los agentes y colas de mensajería que expone Erlang (Armstrong, 2003).

[8] Por ejemplo, el Almacén de Reportes es el único programa que tiene permitido leer y escribir los reportes desde y hacia la base de datos en la arquitectura de referencia de PATDSI, pero la seguridad del acceso a un reporte dado no se puede capturar en esta simple relación.

[9] Como se nota en la figura 1.5 dada con anterioridad, muchos de los estilos fueron creados durante la década del 90. Aunque en ocasiones se refiera al libro (de la Torre Llorente y otros, 2010), se prefiere la cita de (Buschmann y otros, 1996) por su exhaustividad y mayor generalidad.

(llamados filtros) reciben varios flujos de datos y a su vez producen varios como salida. A los conectores que «transportan» estos flujos de datos se les llama tuberías.

Los autores de (Buschmann y otros, 1996) identifican cuatro «escenarios» en los cuales es posible observar este patrón. Se basan en si los filtros halan activamente los datos desde sus flujos de entrada, o los reciben pasivamente; y el comportamiento análogo para los flujos de salida.

De los cuatro, reconocen que quizá el más común es aquel en cada filtro activamente hala los datos desde sus entradas y empuja los resultados activamente hacia la salida. Lo que, por lo general, implica que cada filtro corre en su propio proceso (o hilo). Fielding falla en identificar que este estilo puede tener ventajas en algunos contextos de los sistemas hiper-mediales (Vázquez Acosta, 2010a).

Arquitectura en capas

La arquitectura en capas ayuda a estructurar las aplicaciones mediante la descomposición de sus funciones más externas en sub-tareas en un nivel de abstracción diferente. Cada capa da una vista coherente de un grupo de funciones. Esta es una vista modular: cada capa constituye un módulo (las cual quizá se pueda refinar en otros módulos).

En (Bass y otros, 2003) indican que este tipo de arquitectura surge al imponer ciertos principios organizativos y concentrar responsabilidades de forma cohesiva.

Las arquitecturas en capas suelen facilitar la creación de sistemas donde las funciones deseadas del sistema se pueden descomponer en funciones de menor nivel; o también se pueden entender un sistema de abstracciones de las funciones del sistema (de la Torre Llorente y otros, 2010, pp. 14–16).

En el mismo libro los autores, además, identifican como estilos a la «Presentación Desacoplada» (de la Torre Llorente y otros, 2010, pp. 17–18) y «N-Niveles (N-Tiers)» (de la Torre Llorente y otros, 2010, pp 19–20). Sin embargo, estos estilos son en realidad derivaciones del primero con ciertos componentes más específicos. En el primer caso, sencillamente se define tres capas: vista de interfaz, lógica de presentación, y lógica de negocio. El segundo, indica que hay componentes de separación física entre varias de las capas del sistema. Para los propósitos de esta tesis, basta con la definición general de este estilo.

Modelo-Vista-Controlador (MVC)

Este patrón es altamente recomendado para sistemas con interfaces de usuario. Aunque existen variantes para el caso de los sistemas con interfaces Web, es importante revisar las características de este patrón.

En el patrón MVC se tipifican tres tipos de componentes: el modelo, el controlador, y la vista. El modelo encapsula los datos y funciones relevantes del sistema. La vista muestra información al usuario, y obtiene del

modelo los datos a representar. Los controladores reciben las entradas del usuario y redirigen estos eventos hacia la vista o hacia el modelo. Además, el modelo puede notificar cambios a la vista.

En (Reenskaug, 1979) se encuentra una referencia histórica invaluable sobre este patrón, y se brinda un ejemplo claro sobre su uso. En (Vázquez Acosta, 2010a, cf. 2) se hace un análisis de la forma en que se implementa este modelo en varios marcos de trabajo para la Web.

Invocación implícita

Este estilo no se encuentra dentro los descritos en (Buschmann y otros, 1996). Se puede encontrar su definición en (Garlan y Shaw, 1994).

En este estilo los componentes no invocan directamente ningún servicio de otro, sino que son invocados implícitamente ante la ocurrencia de eventos (externos) a los cuales cada componente puede responder. La principal invariante de este estilo que un componente no conoce quienes pueden reaccionar ante los eventos que él emite.

Buschmann y otros identifican a este como un patrón de diseño llamado Publicador-Subscriptor. Sin embargo, (Garlan y Shaw, 1994) indican que este tipo de conectores es arquitectural y que existen sistemas que todos sus componentes reflejan este tipo de comunicación. Además, indican que existen otros patrones de diseño (inversión de control, por ejemplo) que reflejan la llamada implícita y no directa.

Garlan y Shaw revelan que una de las principales ventajas de este estilo es su potencial para la reutilización de componentes. Por esta razón, en este documento se tomará como referencia la definición dada por estos autores (Garlan y Shaw, 1994, pp. 9–11).

Cliente-Servidor

Este estilo no aparece dentro de los enumerados por (Buschmann y otros, 1996), pero tanto (Bass y otros, 2003) como (Fielding, 2000) y otros autores (Garlan y Shaw, 1994; Garlan y otros, 1997) sí lo identifican como un estilo arquitectónico válido.

En este estilo se identifican dos tipos de componentes: los clientes y los servidores. Por lo general se tiene un único servidor y muchos clientes, aunque no es necesariamente así. En este modelo el servidor es un ente que espera peticiones desde los clientes y envía respuestas. El inicio de la interacción siempre ocurre en el cliente; aunque esto no quiere decir que una vez establecido el vínculo entre el cliente y servidor, el servidor no pueda activamente enviar datos al clientes; en última instancia esto depende del *protocolo* que se establezca.

En una versión pura de este estilo los clientes solamente se comunican con el servidor y nunca directamente entre ellos. Este caso no es raro, muchos de los sistemas que se utilizan en la actualidad funcionan de esa manera: las aplicaciones de correo electrónico, los navegadores web, etc.

El objetivo fundamental de este estilo es delimitar responsabilidades. Es mucho más simple diseñar un servidor y garantizar su escalabilidad cuando han sido bien identificados los servicios necesarios y su naturaleza. También se suele trasladar toda la responsabilidad de la interfaz de usuario al cliente, y esto lo hace propenso a crear varios clientes para distintas plataforma de presentación (Fielding, 2000, p. 46).

1.5. Conclusiones parciales

Al finalizar este estudio se puede concluir que:

- Las líneas de productos de software son un modelo atractivo para ser aplicado en contextos donde:
  - Exista un dominio determinado o se pueda construir uno.
  - Se puedan definir un grupo central de artefactos que son construidos para dar respuesta a las demandas del dominio y que son reutilizables en todos los productos.
- Las líneas de productos de software suponen un cambio paradigmático en la forma en que se dirigen y controlan los proyectos, especialmente aquellos cuyo objetivo es la construcción y/o el mantenimiento de activos.
- La dirección de un LPS debe estar comprometida con este cambio; puesto que la variabilidad de los productos es un elemento estratégico a mantener por la dirección de la línea.
- La arquitectura es uno de los activos indispensables y críticos para los productos de una LPS. De aquí se puede concluir que la dirección necesita prestar especial atención al equipo dedicado para este artefacto.
- Los activos que son derivados del proceso de desarrollo de una arquitectura de referencia son también de mucha importancia para el funcionamiento correcto de la arquitectura. Ellos capturan lo esencial del dominio y son altamente reutilizados.
- Existen varias formas para presentar la variabilidad de los activos. Resultados experimentales sugieren que una representación formal ayuda a obtener resultados positivos en la derivación de productos a partir de los activos.
- Los estilos arquitectónicos son el mecanismo fundamental para definir muchas de las propiedades de los sistemas. Sin embargo, el autor no ha encontrado estilos desarrollados para las arquitecturas de líneas de productos de software. En el mejor de los casos, se intenta representar la variabilidad, sin llegar al nivel de patrón arquitectónico.
- El modelo de LPS dado en (Pestano y Piñero Pérez, 2011) es abarcador y a la vez flexible para permitir el desarrollo de una arquitectura de referencia para PATDSI.

Capítulo 2

Arquitectura de referencia para la línea de productos

En este capítulo se describe la arquitectura de referencia para la LPS PATDSI, y también se da una definición conceptual de la línea misma y su estrategia de producción.

Se comienza con una descripción del proceso empleado para la creación de la arquitectura, enfatizando los aspectos que son esenciales para las LPS. Esto da paso a la descripción de la línea de productos y otros elementos del proceso de desarrollo de activos, porque este proceso es más crítico durante el curso de este trabajo.

Luego se procede a la descripción conceptual de la arquitectura de referencia basados en dos salidas fundamentales: una que brinda el estilo arquitectónico empleado, y otra que enumera los componentes de la arquitectura que se consideran activos de la LPS.

2.1. Desarrollo de la arquitectura de referencia

Cualquier desarrollo arquitectónico tiene como entradas múltiples requerimientos sobre el producto a crear (Bass y otros, 2003, cf. 4.4). En el caso particular de las LPS estos requerimientos tienen la particularidad de establecerse para un grupo *posible* de productos y no para un único caso. Por estas razones el proceso para crear una arquitectura de referencia para la línea de productos de software PATDSI, ha tenido como guía fundamental el proceso de Desarrollo de Activos descrito en la sección 1.2.1. Este proceso ha sido descrito utilizando un grupo de entradas y salidas al proceso, entre los cuales sobresale la arquitectura. Como se puede observar en (Northrop y Clements, 2007b), la descripción de este proceso es muy abstracta, por esa razón se ha empleado también la descripción de los procesos de la Entidad 1: «Ingeniería del Dominio» dada por (Pestano y Piñero Pérez, 2011).

La figura 2.1 se ilustra el proceso utilizado para desarrollar la arquitectura de referencia. Se han resaltado aquellos procesos en los cuales el autor ha hecho énfasis, y cuyas actividades concretas se describen en esta tesis.

En particular, el proceso que el autor ha seguido para la creación de la arquitectura de referencia ha sido iniciado con:

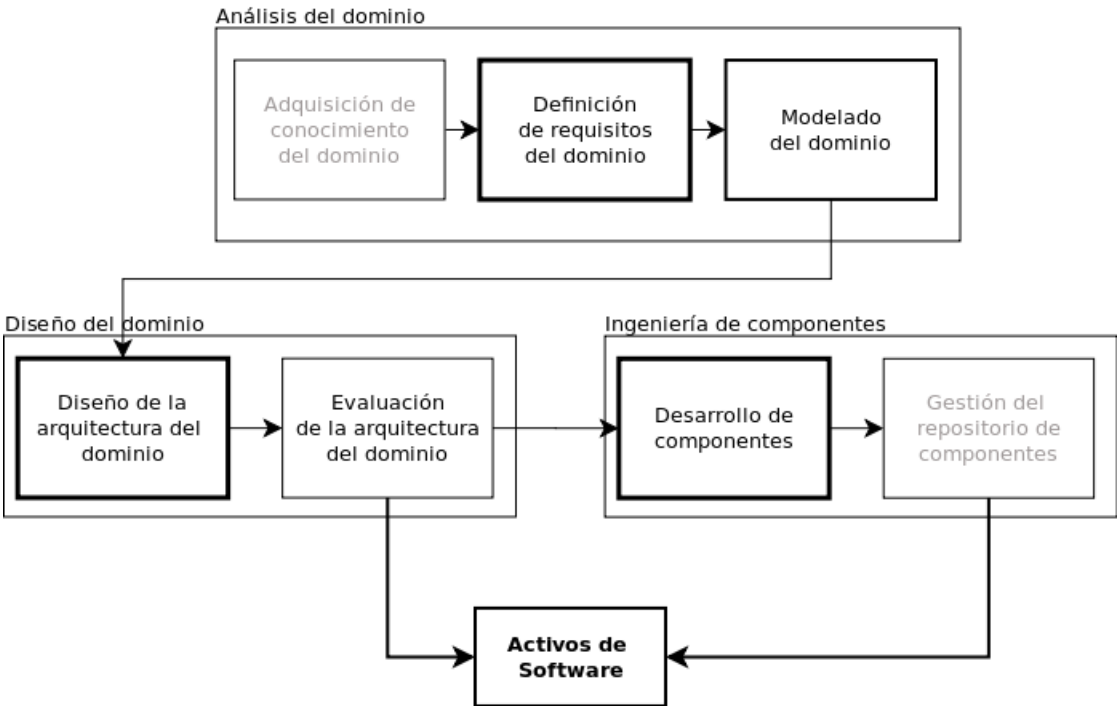

Figura 2.1: Ilustración del proceso de desarrollo de la arquitectura de referencia en el contexto de la entidad de Ingeniería del Dominio de (Pestano y Piñero Pérez, 2011).

- Definición de requisitos del dominio, mediante una caracterización de la LPS PATDSI (sección 2.2).

  Esta caracterización incluye la determinación de los productos que se quieren realizar dentro de la LPS, y también un grupo de restricciones sobre el proceso productivo general, que no pueden ser obviadas a la hora de realizar la arquitectura y sus componentes.

- Luego se procede a la determinación de un grupo de sistemas, subsistemas y componentes que se pudieran utilizar como activos dentro de la arquitectura o que se pudieran tomar para refactorizarlos y generalizarlos (sección 2.3).

- Siguiendo a estas actividades el autor realizó actividades sobre la definición de la estrategia de desarrollo de la LPS, la cual es recomendada por (Northrop y Clements, 2007c).

- Le sigue la definición del alcance de la LPS PATDSI que es un actividad que se realiza dentro del contexto de Modelado del dominio.

  Esto se describe en la sección 2.5, y aquí el autor define conceptualmente todos los activos de software que conforman la arquitectura de la LPS.

- Se continúa con la Definición de la arquitectura de referencia para la LPS PATDSI en la sección 2.6.

La salida fundamental de todas estas actividades fueron los siguientes activos:

- Una modificación del alcance de la línea PATDSI, la cual incluye una definición conceptual de todos los activos de software que se deben construir.

- La definición conceptual de una arquitectura de referencia, basado en la combinación de varios estilos, que permite la creación de productos a partir del ensamblaje de activos pre-elaborados.

Aunque en la figura se señalan también los procesos de desarrollo de componentes, estas actividades y las concernientes a la evaluación del impacto de la arquitectura se brindan en el capítulo 3.

2.2. Requisitos del dominio: Caracterización de la línea de productos PATDSI

La línea de productos «Paquete para la Ayuda a la Toma de Decisiones» (PATDSI) comenzó a concebirse desde hace ya varios años, como parte de los trabajos del tutor de esta tesis. Se concibe como una *suite* de soluciones que permita a gerentes, empresarios, y otros tipos de ejecutivos tomar decisiones basados en la información. Incluso se proyecta la inclusión de herramientas para la sugerencia de soluciones factibles ante interrogantes en *áreas temáticas* como, por ejemplo, la gestión de proyectos informáticos.

En esta tesis, sin embargo, nos dedicaremos a la construcción de una LPS más general y que se puede aplicar en casi cualquier campo: un paquete de herramientas para la ayuda a la toma de decisiones *basada en datos*.

En el alcance de esta línea no se encuentran la codificación de *modelos semánticos* relacionados con áreas de conocimiento específicas que permitan a las herramientas facilitar decisiones más efectivas en distintas en estas áreas; pero está diseñada de forma que *puedan incorporarse estos modelos en el futuro*.

En las siguientes secciones se brindarán descripciones genéricas de los productos de la línea y también se darán los elementos necesarios que describen el estado de partida en la construcción de la LPS.

2.2.1. Productos para PATDSI

El *objetivo fundamental* de la línea de productos PATDSI es la construcción de productos que permitan la *captura de datos* y su recuperación en formas factibles a la *interpretación y el análisis*, con el propósito fundamental de facilitar la *toma de decisiones* a partir de la *información* extraída.

Se excluyen de la lista de posibles productos a:

- Sistemas de control y comando.
- Sistemas de gestión empresarial. Los productos de la LPS PATDSI no permiten modificar los datos capturados, aunque las decisiones que puedan tomar los ejecutivos pueden derivar modificaciones de estos datos en las fuentes originales.

- Sistemas empotrados y/o con alta dependencia de dispositivos externos (pe. sensores). Será posible conectar los productos de la LPS a estos sistemas, pero no construir tal sistema íntegramente dentro de la LPS.

De forma precisa los productos de PATDSI poseen:

- Formas de captura de datos. Ya sea por entrada por formularios o consultas directas a fuentes de datos.
- Formas de análisis de datos. En particular algoritmos de análisis estadístico.
- Formas de visualización de la información con gráficos estadísticos, mapas temáticos, y tableros digitales. Además de un mecanismo para la generación de reportes.

Estas tres categorías determinan en buena medida las *restricciones de los productos* en el proceso de desarrollo de activos como se explica en la sección 1.2.1.

Además de estas categorías, se definen también un grupo concreto de productos que debe ser construidos en la línea. Al definir un grupo de productos concretos, se obtiene metas que son verificables en el tiempo. La lista de los productos que han sido formulados como parte de la estrategia para la creación de la línea incluye:

- Generador de Reportes (GDR).
- Sistema de Captura de Datos Estadísticos.
- Sistema para la creación de Tableros Digitales (Dashboards).

A falta de un modelo de dominio maduro, estos productos brindan características del dominio para todos los productos.

2.2.2. Restricciones de la producción en PATDSI

Como se explica en el epígrafe 1.2.1, en la LPS se definen varias condiciones del proceso de producción para la construcción de los productos.

En todos los casos se seguirán las siguientes restricciones:

- Todos los proyectos para la creación de los activos serán desarrollados bajo la supervisión del Grupo de Arquitectura de la Línea. Los equipos de desarrollo trabajarán de forma ágil, pero se requiere que se registren en el Redmine proyectos por cada activo y exista un repositorio de código para cada activo.

  En este caso se utilizan varios proyectos para la creación de activos, en lugar de un único proyecto como se sugiere en (Clements y Northrop, 2001; Clements y otros, 2005; Krueger, 2007). La idea es

que, en el estado actual del centro, es más manejable tener varios proyectos siguiendo un esquema de iteraciones pequeñas, aunque coordinados por el Grupo de Arquitectura de la Línea.

Además, como se prevé la introducción de tecnología nueva en algunos de los activos, al organizar los activos en proyectos separados es más manejable el riesgo de atrasos en los cronogramas de los productos ya comprometidos con los clientes.

- Se deben evitar usar componentes COTS de terceras partes que requieran pago y/o impongan alguna limitación comercial, de uso, de atribución, y/o de limitación de derechos sobre el producto final. En las circunstancias actuales el centro DATEC no tiene asignado un presupuesto claro para el desarrollo de productos. Es otro departamento dentro de la UCI quien decide si debe comprar o no un componente COTS.

- La inclusión de un componente COTS libre es permitida siempre que la licencias de uso del componente no imponga limitaciones comerciales, de uso, de distribución, de atribución, y/o de limitación (parcial) de derechos sobre el producto final.

  Esta restricción, no excluye el uso de componentes con licencias como la LGPL, MPL, MIT, y otras muchas; pero limita el uso de componentes que se obtengan exclusivamente bajo la licencia GPL y otras similares.

- Se debe evitar el uso de componentes COTS no portables. No se pueden usar componentes que funcionen solamente sobre plataformas Windows.

  Actualmente los productos de la línea están orientados a plataformas GNU/Linux; sin embargo, existen razones comerciales para en el futuro incluir las plataformas Windows también.

  Si bien es posible incluir componentes que no funcionen en Windows, esta decisión debe considerar que en el futuro es posible que se necesite adoptar soporte para Windows, y por tanto un análisis de factibilidad o encontrar alternativas portables no estaría de más.

La documentación de estas restricciones es importante debido a que impactan las decisiones que se pueden tomar en cuanto al diseño de los componentes y productos.

2.3. Inventario de sistemas legados

En esta sección se enumeran todos los componentes que pueden ser utilizados para el proceso de construcción de activos. Esta lista recoge los sistemas que actualmente se desarrollan dentro de DATEC y otros centros de la UCI, así como componentes COTS.

De cada componente se hace un resumen sobre la posibilidad de incorporarlo íntegramente a la arquitectura de referencia de PATDSI, o si se puede utilizar como base para la creación de un activo más apropiado. Este análisis fue guiado tanto por las restricciones de los productos, especialmente las categorías de activos

determinadas en la sección 2.2.1, y también por un grupo de principios arquitectónicos explicados en la sección 2.6.

En esta lista, por cada activo candidato se da una descripción necesaria para ubicarlo dentro del contexto de la LPS, y también se brinda una medida sobre su estado actual en dos dimensiones: terminación y adecuación. En la primera dimensión se dirá si el activo está en desarrollo o no, y en la segunda se pretende informar sobre su adecuación a la arquitectura de referencia de la LPS PATDSI.

En algunos casos se verá que el activo aunque está «terminado» es, además, «parcialmente» adecuado: esto significa que es probable que se esté explotando este componente en productos reales, y concurrentemente exista otro proyecto donde se esté refactorizando/generalizando para incorporarlo íntegramente dentro del grupo de activos de la línea.

ExtJS 3.0

Descripción: Este es un activo COTS desarrollado por una empresa extranjera. Es una biblioteca de componentes que facilita la creación de interfaces de usuario para aplicaciones Web.

Estado de terminación: Terminado.

Estado de adecuación: Parcial. No se adapta naturalmente al esquema de llamadas RPC que promueve la arquitectura de referencia.

Este componente, además, se obtiene bajo una licencia GPL u otra comercial que requiere de pago. Esto no se corresponde con las restricciones de la producción para el desarrollo de activos. Sin embargo, este es un caso de negocio complejo debido a varias fuerzas: Por una parte muchos de los desarrollos actuales que se han realizado en la línea y que son la base para minar componentes reutilizables están basados en ExtJS 2.2, y sería mucho más fácil migrar hacia la versión 3.0 de esta librería que pasar todos los componentes para alguna alternativa con una licencia adecuada.

Acaxia

Descripción: Es un proveedor de identidad basado en el estándar SAML 2.0 (SAM, 2005) y que incluye un modelo de roles RBAC (Sandhu y otros, 2000). Es desarrollado y mantenido por la UCI.

Estado de terminación: Terminado.

Estado de adecuación: La implementación actual no es compatible con el estándar SAML2.0. El equipo de desarrollo de Acaxia tiene prevista una implementación de este estándar para finales del año 2011.

Generador Dinámico de Reportes (GDR)

Descripción: Es un sistema completo que permite diseñar reportes tabulares de forma dinámica. Se puede conectar a bases de datos en sistemas PostgreSQL, MySQL y SQLite. Permite diseñar reportes y exportarlos en varios formatos.

Estado de terminación: Terminada la versión 1.5.1. En pruebas de liberación la versión 1.6.1.

Estado de adecuación: Parcial. Este es un producto cuya integración con el resto está limitada aún; solamente la versión 1.6.1 consume los servicios del ChartServer y R-Server; sin embargo, todavía adolece de:

- Integración como un proveedor de servicios en SAML 2.0 para la integración con Acaxia.
- Basar su interfaz con Caxtor para poder empotrar sus componentes visuales en aplicaciones más complejas.
- Identificar claramente los servicios básicos de este sistema y documentar sus IDL.
- Planificar la integración con el Generador de Mapas Temáticos y crear los componentes para la creación de reportes que incluyan mapas.
- Permitir la creación de reportes complejos con varios tipos de fuentes de datos y varias formas de visualización.

Este activo es, además, un subsistema completo. En (Clements y Northrop, 2001) se documenta que «el secreto» de la empresa CelciusTech para mantener una LPS compleja era reutilizar la integración de activos, es decir, mantener un grupo grande de activos pre-integrados listo para su inclusión en otros productos más grandes. El GDR es precisamente uno de estos subsistemas, pero tiene además la cualidad de un producto en sí mismo con un mercado particular. A la par es el sistema con el cual se diseñan y visualizan los reportes del resto de los productos de la línea; por lo que constituye además una herramienta del proceso de desarrollo.

Note además que si bien en la actualidad este es un activo parcialmente adecuado, se proyecta la refactorización del mismo para obtener activos más genéricos:

- Visor de reportes
- Diseñador de reportes
- Constructor de reportes
- Diseñador de entidades de datos
- Mapeador de entidades de datos
- Inspector de bases de datos
- Optimizador de reportes

En la práctica este producto y Caxtor (Martínez Alarcón y otros, 2010) son los productos motrices de la LPS PATDSI. El GDR es el producto comercial que sustenta toda la estrategia de negocio de la línea y Caxtor es el producto tecnológico que permite la creación y composición de nuevos productos a partir de los activos desarrollados. Esto no significa que PATDSI se limite a estos dos productos, sino que hasta la fecha estos dos han sido los que más han influido en el diseño de la línea.

Apache Thrift

Descripción: Es una tecnología que permite la creación de servicios y clientes de estos en varios lenguajes. Utiliza un IDL para describir los servicios y generar el código que pueden utilizar los clientes para consumir el servicio, y también genera el código base para implementar los servidores.

Estado de terminación: Es un producto en continuo desarrollo aunque se considera suficientemente maduro para su explotación en proyectos reales.

Estado de adecuación: Parcialmente adecuado. Permite la integración de servicios basados en conectores RPC como se indica en la arquitectura de referencia. Quedan pendientes elementos de seguridad de las conexiones entre los componentes de interfaz y los servicios remotos.

2.4. Estrategia para la producción en la línea

En el caso de la línea de productos para PATDSI no se podía aplicar ninguna de las estrategias identificadas en (Clements y otros, 2005) de forma pura. En algunos casos se han creado los activos reaccionando ante las necesidades de los proyectos, y en otros casos se iniciaron proyectos con el propósito explícito de construir activos que luego serían integrados al proceso de creación de productos. No obstante, es notable que en (Northrop y Clements, 2007a) los autores caracterizan que las LPS con una estrategia pro-activa realizan el alcance de la LPS mediante la definición del «conjunto (más frecuentemente, *un espacio*) de sistemas que constituirían la línea de productos» (Northrop y Clements, 2007c)[10].

En cierto sentido se puede asegurar que los proyectos para la construcción de activos altamente relacionados con los elementos funcionales de PATDSI (ChartServer, R-Server, Generador de Formularios, etc) fueron constituidos bajo una estrategia reactiva, por ejemplo, el ChartServer se creó respondiendo a la necesidad de incluir gráficos en los reportes generados por el Generador de Reportes. Por otra parte, los proyectos para la construcción de los activos «tecnológicos» (Caxtor, Componente de comunicación en tiempo real, etc.) se han iniciado pro-activamente.

Cualquier formulación rígida de lo que se explica en el párrafo anterior sería impropio de la realidad. Ninguno de estos proyectos se creó de forma aislada y todos tenían algún grado de relación. El reto no era, por

[10] La oración original es: «They define their product line scope to define the set (more often, *a space*) of systems that will constitute their product line.»

tanto, seguir una estrategia formal para la creación de los activos, sino adoptar aquella que respondiera apropiadamente a las necesidades que se estaban presentando, ya fuera dentro de un proyecto o de un grupo de ellos.

Dentro de las estrategias dadas en (Chastek y otros, 2009, p. 14), se puede enmarcar este trabajo como una estrategia *ágil*:

- Para cada activo se creó un proyecto.
- Estos proyectos funcionaban de forma relativamente independiente, en el sentido de que sus cronogramas, e iteraciones parciales no se correspondían en tiempo necesariamente.
- Las dependencias entre activos se manejaron a nivel de línea y no por cada proyecto en particular. Es decir, la fecha de comienzo de un proyecto en particular se definía en portafolio general que iban siendo seguido a medida que la línea de activos madurara.
- El plan general de la LPS PATDSI se utilizaba como mecanismo de evaluación del cumplimiento de los objetivos de cada proyecto.

Siguiendo esta estrategia, en el curso de cinco meses se desarrollaron completamente dos activos (el Chart Server y el R-Server), y fueron integrados a una nueva versión del Generador Dinámico de Reportes (GDR). Los tres proyectos coexistieron en tiempo y, siguiendo los lineamientos estratégicos establecidos por el autor de este trabajo de conjunto con su equipo de dirección, se fue realizando exitosamente la integración de los primeros activos con el segundo. Utilizando una métrica simple de $activos \div meses$ se obtiene una taza de $3 : 5$, cuando los datos anteriores de la línea indican que se obtenía una versión nueva de un producto en unos seis meses, para una razón de $1 : 6$. En el capítulo 3 se expondrá una métrica más adecuada basada en el esfuerzo y no solamente en el tiempo; sin embargo, esta valoración nos indica que con esta nueva visión se obtienen más productos/activos. Es factible pensar que los nuevos requisitos del GDR para los cuales se desarrollaron los activos en cuestión se hubieran logrado sin cambiar el modo de producción hacia una LPS en formación; sin embargo, la misma concepción de estos activos ya como elementos reusables con una interfaz definida y con propiedades arquitectónicas conocidas, los ha puesto en el plan de desarrollo de otros productos de la línea como el Sistema Integral de Gestión Estadística.

Si bien la estrategia que prima dentro de la línea es la ágil, no se descartan acciones relacionadas con otras estrategias como son: la abierta con la posible creación de proyectos comunitarios para parte de los activos tecnológicos; la automatizada al introducir técnicas de integración continua; y la estrategia de generación al introducir conceptos de MDD y MDA en la línea de producción (Chastek y otros, 2009, pp. 14–15).

2.5. Modelado del dominio: Definición del alcance de la línea PATDSI

El Paquete de Ayuda para la Toma de Decisiones incluye un conjunto de herramientas que permiten capturar datos de distintas fuentes, consultar estos datos en forma de reportes, visualizarlos y monitorizar datos o

indicadores y programar acciones de respuesta ante estos eventos. Las aplicaciones reales que se pueden dar con este grupo de características involucran desde la captura de un formulario simple, hasta la representación en tiempo real, sobre un mapa temático de indicadores geo-referenciados.

Dentro del núcleo de PATDSI se encuentran los siguientes grupos de activos:

- Componentes de captura y almacenamiento de datos,
- Componentes de consulta y visualización de información,
- Componentes del Cuadro de Mando de Integral,
- Componentes tecnológicos y creados para el soporte mismo a la LPS.

En el primer grupo de activos se encuentran:

El diseñador de unidades de información o entidades. Este componente permite diseñar *entidades* que son relevantes para capturar sus atributos. Por ejemplo, de una `Persona` (que sería una entidad) nos interesa registrar su `edad`, `peso`, `color de la piel`, etc...

Este componente permite además, establecer relaciones entre entidades diferentes. Es, en cierto sentido, un componente que permite crear modelos del tipo entidad-relación.

Este activo supliría las deficiencias que hoy tiene el Generador Dinámico de Reportes para representar los datos. Actualmente el GDR utiliza un sistema demasiado acoplado a la base de datos subyacente lo que provoca que:

- Sea complicado integrarlo a otros sistemas, incluso sistemas de la misma línea de productos.
- Los reportes no son transportables de una base de datos a otra sin modificaciones.
- No es extensible el GDR para incluir ontologías sobre los conceptos modelados.

El diseñador de formularios Este componente permite editar y componer formularios de captura de datos. Los formularios pueden contener diversos tipos de controles que luego son ejecutados para capturar datos de entidades. Este componente permite asignar las relaciones entre las entidades creadas por el diseñador de unidades de información a controles de captura de datos.

Este componente supliría las deficiencias actuales de SIGE. Actualmente SIGE tiene un único modelo conceptual de formulario basado en páginas, indicadores y aspectos. Este modelo además, se representa inyectivamente en la base de datos[12] lo que provoca que el almacenamiento de estos datos se "desnaturalice" en la base de datos.

El generador de formularios Este componente simplemente genera formularios que pueden luego ser modificados por el diseñador, a partir de modelos creados por el diseñador de entidades. Este componente es una facilidad para agilizar la creación de formularios una vez que ya están definidas las entidades.

De cierta manera, este componente es un "enlace" entre el primero y el segundo. Se concibe para aumentar la *usabilidad* de los productos de PATDSI, pero no aporta comportamiento realmente nuevo. Sin embargo, no deja de ser importante para los requisitos de calidad percibida de los sistemas.

El mapeador de entidades Este componente recibe un modelo de entidades y crea (siguiendo un conjunto de reglas) los elementos de persistencia en una base de datos. Este componente aisla al resto de las complejidades de la BD y permite que exista un lenguaje único para representar instancias de entidades. Otra función de este componente es la consulta de los datos. Debe permitir realizar consultas complejas.

Este componente debe eliminar la deficiencia de la integración de los modelos de diferentes productos de la LPS. Actualmente es complejo integrar el GDR con SIGE porque estos sistemas no comparten los modelos conceptuales de representación de datos. Al SIGE y el GDR compartir este componente entonces cualquier modelo ER que pueda ser capturado por SIGE puede ser reportado por el GDR con muy poco esfuerzo del usuario. Por otra parte esto aumenta la flexibilidad de los productos de la LPS en el sentido que se pueden construir productos mucho más específicos: Un pequeño sistema de votación tiene un modelo ER y un grupo de reportes definido; no sería necesario incluir los diseñadores de formularios, generadores de formularios, diseñadores de reportes, etc...; sino que estos *se usarían durante el proceso de construcción*, pero no formarían parte del producto final.

Esta es una característica importante del diseño de la LPS PATDSI, dado que estos activos no solo se utilizarían cuando se incluyan como parte de un producto final, sino el proceso de producción.

El inspector de bases de datos Este componente permite inspeccionar una base de datos ya creada y extraer las entidades que representa. Este componente permite que bases de datos ya creadas puedan representarse en el mismo lenguaje de entidades que comparten el resto de los componentes y así poder generar, por ejemplo, generar formularios para trabajar con la BD, o poder realizar reportes sobre la misma.

La función de este activo dentro de la arquitectura de referencia es poder integrar los componentes de visualización (especialmente el GDR) con otros sistemas que no necesariamente compartan el módulo de representación de entidades. Para el GDR como producto en sí mismo esto es vital, dado que este de integrarse a las bases de datos de las empresas. La interfaz de usuario de este componente le brindará a los miembros del departamento de IT de las empresas la posibilidad de inferir un modelo ER a partir de la BD.

Sin embargo, hay elementos dentro de las bases de datos que no se corresponden con los conceptos de un modelo ER; por ejemplo, las vistas y los procedimientos almacenados. Como este componente se utilizaría fundamentalmente para la generación de modelos ER para el GDR, las vistas se pueden traducir a "entidades". Con los procedimientos almacenados no se pueden dar las mismas garantías porque estos pueden tener efectos colaterales en la BD.

Se define entonces que los modelos ER y este activos tiene dos puntos de variabilidad:

- Un modelo ER puede incorporar nuevos tipos de entidades y asociaciones.
- Este activo puede incorporar nuevos tipos de reglas para inferir elementos físicos de la BD.

El objetivo es que en este activo no se encuentren directamente los problemas particulares de un producto.

En el segundo grupo de componentes (de consulta y visualización) podemos enumerar los siguientes:

El servidor de gráficos estáticos (ChartServer) Este componente permite la inclusión de gráficos estáticos en varios productos de la línea. También es un componente muy flexible para integrarse.

Tanto el generador de reportes como otros productos que necesiten mostrar datos de forma gráfica utilizarían este componente para generar los gráficos.

El servidor de análisis de datos (R-Server) Este componente permite procesar los datos capturados o almacenados y ejecutar varios algoritmos de análisis estadísticos como son los algoritmos de regresión lineal, binomial; las tablas cruzada, entre otros.

Este componente es un necesidad para integrarse tanto a SIGE como al posibles versiones del generador de reportes.

El Generador Dinámico de Reportes Este sistema permite realizar consultas a las bases de datos (mediante la inferencia de sus entidades o recibiendo directamente el modelo de entidades) y construir reportes en HTML y PDF sobre los datos. Es un componente de gran utilidad para la toma de decisiones pues representa un elemento de consulta de información operativa.

Internamente este sistema utiliza el R-Server y el ChartServer. El primero lo utiliza para la construcción de tablas cruzadas, incluir análisis de regresión lineal, u otros tipos de análisis estadísticos en los reportes. El segundo se utiliza para inclusión de gráficos en los reportes. Ambos se consideran otros activos de la LPS.

Este es un activo preintegrado que incluye:

- El mapeador de entidades
- El inspector de bases de datos
- El visor de reportes
- El diseñador de reportes
- El generador de consultas
- El constructor de reportes

El diseñador de tableros digitales Este componente permite componer varios tableros digitales con gráficos vivos que representen los valores de los indicadores de un cuadro de mando integral.

Cada tablero digital se asocia con un CMI, al cual consulta para obtener los valores de los indicadores definidos. Esta consulta se hace a través del componente de comunicación por XMPP, de modo que una vez que en el CMI se actualice un valor, el tablero digital obtiene el nuevo valor y actualiza el gráfico apropiado.

[12] Esto quiere decir que en la BD lo que se representan son páginas, indicadores y aspectos y no los atributos del modelo real que se intenta modelar. Si bien funciona dentro de los límites de SIGE, falla a la hora de integrarse con otros sistemas.
La propuesta de un modelo conceptual similar al que se propone en (Vigna, 2002, 2003) serviría para eliminar esta barrera.

Dentro de los componentes del Cuadro de Mando Integral (CMI) se encuentran:

El sistema de gestión de indicadores Este sistema permite definir varios indicadores claves de procesos y mantener sus valores. Los indicadores pueden extraerse de otras fuentes, calcularse o entrarlos directamente por un formulario.

El tablero digital consulta a este sistema para poder determinar los indicadores que han sido definidos y también consultar su valor.

Este sistema también emite alertas sobre si un indicador no está siendo actualizado, o tiene valores fuera del rango aceptable. Para emitir estas señales se utiliza el componente de comunicación por XMPP.

En el cuarto grupo (de soporte o tecnología) tenemos un importante grupo de componentes que se usan en la construcción o como parte de los componentes anteriores:

- La arquitectura Caxtor para la creación de aplicaciones web.
- El IDE Caxtor, que contiene el diseñador de formularios, y el diseñador de entidades, el generador de formularios, etc...
- El componente de comunicación por XMPP, que se emplea en el tablero digital para mantener actualizados (en tiempo real) el valor de los indicadores del CMI.

2.6. Diseño del dominio: Definición de la arquitectura de la línea de productos

En este epígrafe se detallan los *principios* de la arquitectura de referencia desarrollada para la línea de productos. Se usa uno de los productos más complejos de nuestra línea para demostrar los distintos elementos de la arquitectura.

2.6.1. Estilo arquitectónico de alto nivel

En el más alto de los niveles los productos de PATDSI son sistemas Web que adoptan como *estilo básico* el estilo «cliente-servidor», con un subconjunto del sistema de restricciones (derivaciones) del estilo REST (Fielding, 2000, cf. cap. 5). Básicamente se intenta reflejar la idea de las aplicaciones que corren en el navegador están separadas físicamente del servidor y su única dependencia es similar a la que pudiera tener, por ejemplo, un cliente de correo con el servidor de correos.

Esto implica que se deban establecer un grupo de principios para mantener esta visión en un estado coherente. Uno de estos principios son los conectores definidos, los cuales, a diferencia del estilo cliente-servidor y de REST, se definen en PATDSI como:

1. Los conectores llamada-respuesta RPC entre los clientes y el servidor.
2. Los eventos en *todos* los componentes.
3. Las tuberías desde el servidor al cliente.

El orden dado en esta lista indica el nivel de preferencia. Es decir, se considera que la mayoría de las interacciones deben ser iniciadas por el cliente como una llamada remota (*Remote Procedure Call*, RPC). Esto está en perfecta alineación con el estilo cliente-servidor.

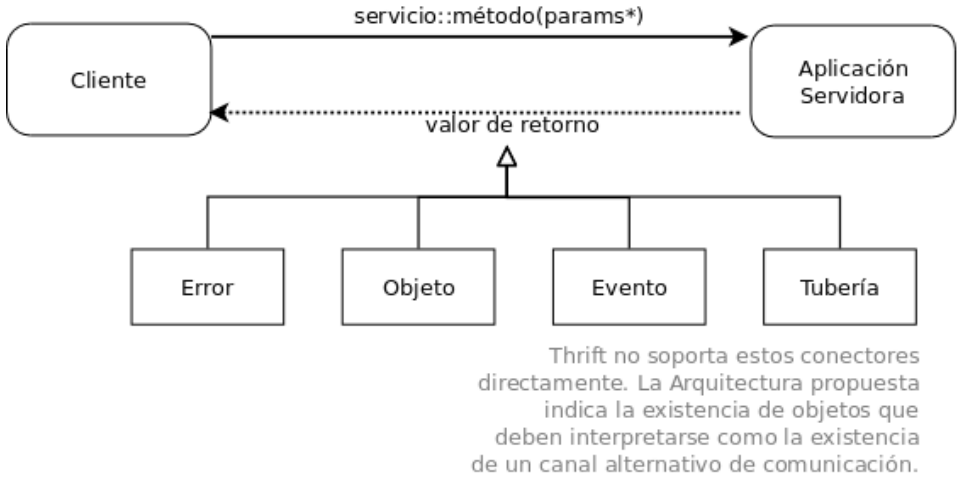

Figura 2.2: Diagrama C&C del estilo de alto nivel con el conector RPC

En segundo nivel, cada componente puede emitir y suscribirse a eventos. Este tipo de conectores es especificado en el estilo de invocación implícita; aunque en este sistema arquitectónico se utilizará la publicación/suscripción a *colas de mensajería*, y no a pares de `(evento, emisor)`.

Finalmente, hay operaciones que requieren que las respuestas del servidor al cliente adopten el estilo de tuberías y filtros. Este caso se considera poco común en el contexto de la línea, pero es necesario para garantizar el tiempo de respuesta percibido de varias operaciones (Vázquez Acosta, 2010a).

La figura 2.2 brinda un diagrama C&C con anotaciones sobre los tipos de datos que se intercambian y extensiones para soportar los otros dos conectores. Un componente en la aplicación Cliente inicia una llamada RPC invocando un método de un *servicio* que ha sido publicado en la aplicación Servidor. En el servidor se localiza el componente que procesa este pedido y se redirige al proveedor del servicio solicitado, el cual retorna el resultado de la operación que es enviada de vuelta al cliente.

Una de las invariantes del estilo que se propone además, sigue la derivación REST de que los dos primeros tipos de conectores están totalmente desprovistos de estado. El conector de tuberías, una vez establecido, mantiene un estado interno que en esencia indica si hay más datos o no en la tubería.

Otra de las características del estilo adoptado (que se deriva también de REST) es que el código que se ejecuta en el cliente se obtiene desde el servidor bajo demanda. En este caso se hace la delimitación de que el código será ejecutado por un navegador y la representación debe ser compatible con el estándar ECMAScript (ECMA International, 2009).

Esta característica es notable porque se hace uso de los conectores "normales" de HTTP, sin embargo este tipo de interacciones no se considera del modelo arquitectónico de los productos sino un efecto del contexto donde se ejecutan estos sistemas.

La plataforma Caxtor, que es parte integral la tecnología base de la arquitectura, provee una abstracción para garantizar:

- La ejecución de llamadas remotas
- La emisión de eventos y también suscribirse a ellos
- El procesamiento de tuberías en el cliente
- La carga bajo demanda de código

En cierto sentido esto simplifica la tarea de documentar la arquitectura de los productos, dado que basta con identificar las operaciones remotas que los clientes pueden invocar, los eventos que cada componente puede disparar, y las tuberías necesarias para recibir algunos flujos de datos desde el servidor. No sería necesario identificar conectores especiales para transportar el código desde el servidor al cliente, dado que esta es una característica «inherente» a Caxtor.

Por tanto, también es parte de la arquitectura de referencia de PATDSI, la creación de un perfil UML2 que permita representar los servicios que son expuestos a los clientes en una forma simplificada; los mecanismos de diseño por los cuales se logra la comunicación entre los clientes y el servidor es una cualidad que se abstrae del diseño y es parte integral de la arquitectura de referencia. La figura 2.3 muestra las ideas iniciales del perfil UML2 que se está desarrollando para poder especificar los productos de PATDSI.

2.6.2. Estructura de los clientes

Una aplicación cliente, por lo general, *se compone* de varios módulos de interfaz preintegrados. Cada módulo, a su vez, puede estar compuesto por otros módulos menores. Por lo general, cada módulo se corresponde con un único componente en un diagrama C&C. En la figura 2.4 se puede observar cómo la aplicación es estructurada en forma de árbol por componentes.

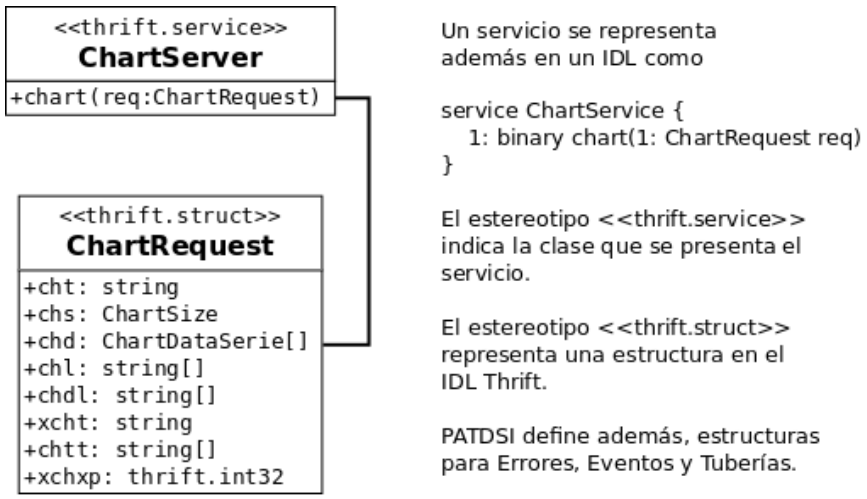

Figura 2.3: Un ejemplo inicial de la especificación de un servicio en UML2 para PATDSI.

Los conectores en este diagrama indican que los nodos padres pueden invocar todos los servicios de los hijos; y los hijos pueden lanzar eventos que, por lo general, solamente recibiría el nodo padre. Además, ningún componente puede comunicarse directamente con aquellos que son hermanos suyos en este árbol.

Esta estructura promueve eficientemente la reutilización de interfaces complejas preintegradas[13]. Cualquier interfaz compuesta de esta manera puede extenderse de forma muy simple para convertirse en un nodo de otro árbol; por lo general, solamente necesitará extenderse para emitir un conjunto de eventos. Lo que se puede lograr fácilmente mediante la herencia, o con un patrón Adaptador (Larman, 2002, p. 272).

Muchos de estos componentes se estructuran internamente siguiendo el patrón arquitectónico Modelo-Vista-Controlador (Buschmann y otros, 1996; Reenskaug, 1979). Sin embargo, es posible que algunos componentes sean exclusivamente Vistas que asumen un Modelo particular. En este último caso, la función de Controlador es asumida por el padre y el Modelo es probablemente un Adaptador del Modelo del padre. Idealmente esto sucede solamente en el último nivel del árbol; pero en ocasiones es necesario mantener el modelo de datos de muchos componentes en un nivel superior del árbol, y otras veces la anomalía surge de la integración de varias interfaces.

Otra cuestión surge en la comunicación con el servidor. Solamente los controladores pueden invocar un método RPC al servidor; y esto puede suceder en varios niveles del árbol.

Por tanto, a la hora de describir un nodo de este árbol se necesitaría:

- Indicar los métodos públicos de su interfaz.
- Indicar los eventos que el componente emite.
- Indicar los servicios y métodos RPC que invoca.

[13] De hecho, este árbol representa la integración de esos componentes en una interfaz completa.

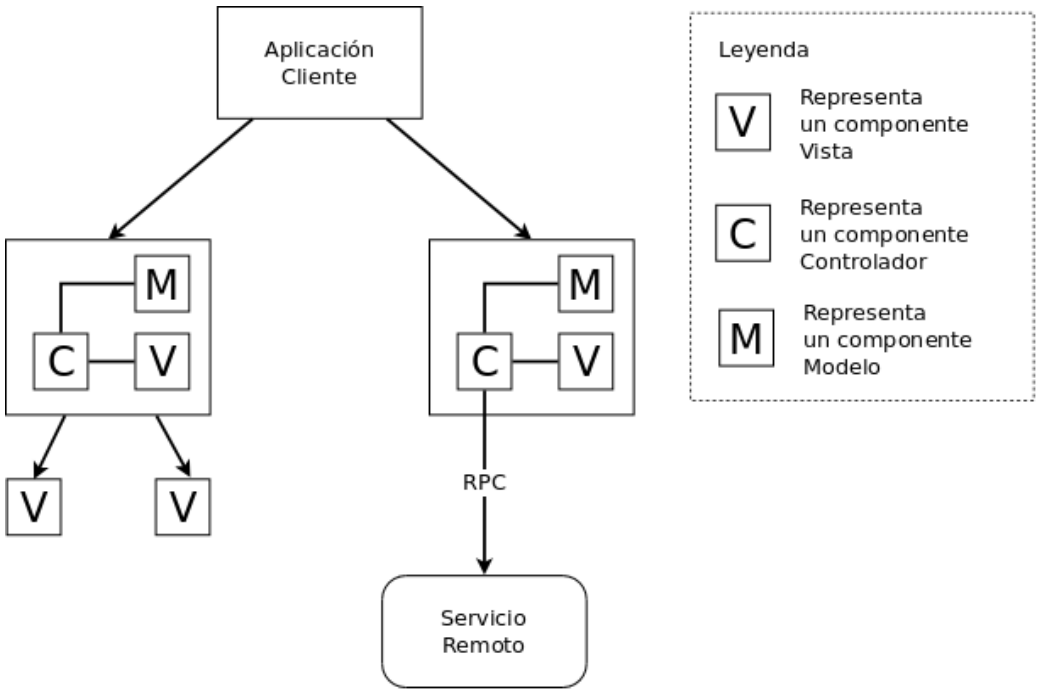

Figura 2.4: Representación de la estructura C&C de las aplicaciones clientes

- Indicar las dependencias del componente que pueden ser inyectadas; o también, indicar las *colaboraciones* en las cuales puede participar este componente.

Los primeros tres elementos indican la interfaz exterior del componente; el último de ellos es un elemento de composición que hace más viable las integraciones: es posible que un componente necesite para su funcionamiento una instancia de otro; sin embargo, esta dependencia solo existe en la interfaz y por tanto cualquier implementación de esta interfaz sería adecuada para participar en la colaboración.

Por esta razón, aparece como un elemento de la arquitectura el concepto de *colaboración*. Se debe notar que en todos los diseños orientados a objetos hay diagramas de colaboración, sin embargo, en la arquitectura de PATDSI la colaboración no es solo un diagrama que describe la realización de un caso de uso o un escenario; sino que puede «codificarse» en el sistema, y los componentes pueden referirse en tiempo de ejecución a las colaboraciones. Por esta razón, se puede afirmar que el arquitectura de referencia de PATDSI existen *colaboraciones de primer nivel*[14] y que tienen prácticamente el mismo lugar que las clases en los diagramas de diseño de cualquier aplicación en esta LPS.

En la arquitectura de referencia las colaboraciones de primer son un elemento para garantizar la variabilidad en los productos. Siempre que se modele una colaboración de primer nivel se está dando cabida a que se empleen diferentes variantes de los objetos que participan en la misma. A diferencia de las representaciones

[14] Se utiliza la expresión *colaboración de primer nivel* para distinguirlas del resto de las colaboraciones las cuales simplemente existen en tiempo de ejecución, pero no pueden ser referidas durante la ejecución del sistema como un objeto más del mismo.
En el diseño de lenguajes de programación se suele decir que son elementos de primer nivel aquellos para los que el lenguaje da un soporte directo.

empotradas como se presenta en (Pérez Lamancha y Polo Usaola, 2009), una colaboración de primer nivel captura todo un punto de variación, y los componentes pueden *seleccionar la variante más apropiada* en tiempo de ejecución.

En la plataforma Caxtor (Martínez Alarcón y otros, 2010) se ha hecho una implementación de esto, mediante la formalización de los «*entrypoints*»[15]. Este el nombre que se le da en Caxtor a las colaboraciones. En Caxtor una colaboración tiene un nombre y un grupo limitado de participantes dados por varios roles. Al diseñar e implementar un componente se puede declarar que puede participar en una colaboración determinada jugando un rol. En esta declaración se incluye un objeto de configuración que habilita precisamente la participación del componente en la colaboración.

Este esquema en Caxtor es realmente muy simple, pero deja en manos de los desarrolladores la verificación de que los componentes designados para implementar una colaboración utilicen la formalidad de los *entrypoints*, que implica las actividades de:

- Identificar las colaboraciones candidatas para incluir en la arquitectura del sistema como colaboraciones de primer nivel, las cuales se representarán en el sistema con un *entrypoint*.

  Es importante distinguir entre todas las colaboraciones del sistema aquellas que son importantes codificar como colaboraciones de primer nivel.

  Desde un punto de vista metodológico se puede establecer que cualquier colaboración en la cual existan uno o más puntos de variabilidad (Pohl y otros, 2005, pp. 61–62, def. 4-3) de los productos de la LPS, es una colaboración candidata.

- Identificar las colaboraciones de primer nivel.

  Mediante el análisis (y también un diseño previo) de las colaboraciones candidatas se puede decidir si incluirla o no como una colaboración de primer nivel. Es posible que existan puntos de variabilidad para los cuales es mejor optar por otro mecanismo diferente al *entrypoint*.

- Diseñar las colaboraciones de primer nivel; identificar los participantes y sus roles.

  Durante esta actividad es importante identificar todos los participantes (especialmente aquellos que sean objetos de variabilidad) de la colaboración y documentar el rol que juegan, así como su interfaz.

Evidentemente estas actividades son parte de los flujos de diseño dentro de las metodologías de desarrollo del sistema, y están sujetas a las decisiones sobre los objetivos de cada iteración en cada equipo de trabajo.

En la figura 2.5 se muestra una representación de cómo capturar la decisión de que una colaboración es de primer nivel. Se utiliza el mecanismo de UML de extender el lenguaje para representar nuevos conceptos. En este caso, simplemente se anotan los mensajes entre objetos con un nuevo estereotipo y una clase de asociación.

[15] En este documento se escribirá el vocablo técnico *entrypoint* siempre que se quiera referir a la implementación específica de Caxtor para codificar las colaboraciones.

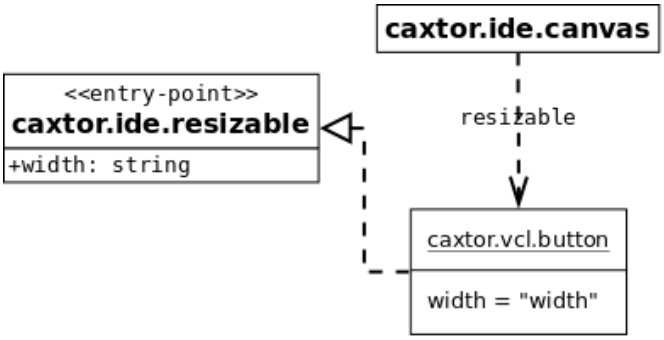

Figura 2.5: Una representación esquemática de una colaboración de primer nivel.

2.6.3. Vista externa del servidor

De lo explicado en el epígrafe 2.6.1 se puede inferir una parte de la estructura prevista para la aplicación servidora. En este caso el servidor se compone de dos niveles:

- El nivel de infraestructura
- El nivel de servicios

En el primer nivel el servidor sencillamente responde como un servidor REST, lo que permite que el código de las aplicaciones clientes sea transportado hacia los navegadores; así como otro tipo de recursos como imágenes, documentos, y otro tipo de ficheros.

Esto sucede en la fase en que la aplicación está siendo *iniciada* por el usuario. Una vez que el navegador tiene el código de la aplicación cliente y comienza su ejecución, la aplicación cliente utiliza los conectores RPC, los eventos y las tuberías para su interacción con el servidor. Es posible que, durante la ejecución del cliente, este necesite cargar otro módulo, cuyo código, aún no ha sido transportado hacia el navegador, y se opten por hacer una petición REST simple.

La plataforma Caxtor brinda facilidades para realizar la carga de módulos en demanda haciendo pedidos a un servidor web estándar; lo que permite concentrarse en el segundo nivel: el de los servicios.

Una aplicación se compone entonces por un grupo de *servicios*, de los cuales se conoce su interfaz, la cual se describe utilizando un lenguaje de descripción de interfaces (IDL, por sus siglas en inglés). Para la descripción de las interfaces, PATDSI utilizará el lenguaje IDL Thrift y también se pueden utilizar las herramientas de generación que este activo brinda para crear el código inicial de los servidores y clientes. Es importante notar que en el estado actual de Thrift no contempla formas para garantizar la seguridad; por esto, PATDSI impone las siguientes restricciones:

- Existe un único servicio (brindado por Acaxia) que mantiene los usuarios que están activos y los privilegios actuales de ellos.

Acaxia también brinda la posibilidad de mantener un grupo de *dominios* y *aplicaciones*. Los servicios de una aplicación deben estar registrados en Acaxia.

- Cuando el transporte de las invocaciones a los servicios sea el HTTP (esto es cuando la invocación se hace desde el cliente), se utilizará el mecanismo estándar de las aplicaciones Web para asegurar la *identificación* del cliente:

  - Al realizar el primer pedido la capa de infraestructura redirige al usuario hacia la página de identificación de Acaxia utilizando algunos de los mecanismos de SAML2.
  - Seguidamente, durante las invocaciones RPC, la capa de infraestructura consulta a Acaxia si el usuario que está identificado tiene la autoridad para consultar el servicio. Si el usuario no tiene la autoridad necesaria, la capa de infraestructura lanza la *excepción* `HttpForbidden` codificada apropiadamente por Thrift.

    Si el usuario tiene la autoridad necesaria entonces la capa de infraestructura, delega el pedido al servicio correspondiente; pero intercepta cualquier llamada a otro servicio que se pudiera producir desde la ejecución del primero de estos.

    Este último paso es necesario porque cuando un servicio invoca a otro utilizando Thrift no lo hace en el contexto de la invocación RPC original.

En este sentido la capa de infraestructura es también un *middleware* de todas las comunicaciones que se producen hacia y desde los servicios. La figura 2.6 muestra un diagrama de esta interacción.

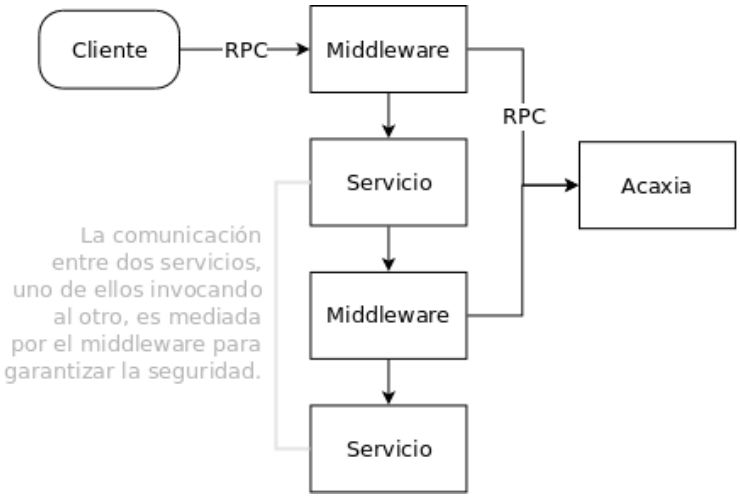

Figura 2.6: Diagrama de cómo se realiza una llamada RPC.

El punto débil de este diseño pudiera ser la interacción con la base de datos si cada servicio puede interactuar directamente con esta. Una opción para eliminar esta debilidad es crear servicios exclusivamente para el acceso a los datos y limitar la posibilidad de conexión a la base de datos desde este servicio. Actualmente, dado los requisitos de los productos que se tienen para la línea PATDSI, esto no supone un problema. Muchos de los productos previstos para PATDSI hasta la fecha solamente utilizan los activos de visualización; y se

despliegan en combinación con sistemas de captura de datos hechos por terceros, y un almacén de datos que colecta un grupo de datos desde las fuentes originales.

Además de las restricciones mencionadas en los párrafos anteriores, el servidor tiene el menor número de invariantes arquitectónicas en PATDSI. Sin embargo, sí se brindan un grupo de recomendaciones para aumentar la posibilidad de reutilización de los componentes el servidor:

- Las aplicaciones servidoras serán *esencialmente distribuidas*. Este recomendación intenta lograr tres efectos:
  - Ganar en flexibilidad en el diseño de cada subsistema para que se use el mejor hardware disponible para la funcionalidad que debe proveer el sistema.
  - Poder incorporar, en la medida de lo posible y sólo cuando sea necesario, más hardware al sistema y que sea utilizado eficientemente.
  - Poder utilizar hardware «ordinario» el cual es mucho menos costoso, aunque más propenso a fallar, por lo que cada subsistema debe ser, en cierto sentido, redundante.

  Este principio se materializa en la creación de los *servicios*, y la colaboración entre ellos para realizar una tarea.

  Otro beneficio de este principio es que mejora la división del trabajo en equipos y cada equipo puede tener mejores posibilidades de dar garantías de disponibilidad y rendimiento del servicio.

  Esto no significa que todas las funciones del sistema se distribuyan: por ejemplo, el activos de interacción con una base de datos a partir de un modelo entidad-relación, es una función que no tiene sentido distribuir en los productos PATDSI, y que es posible que se encuentre en muchos de los subsistemas.

- La segunda recomendación es garantizar el *cero acoplamiento* de los servicios. Con esto se intenta:
  - Que un servicio pueda ser reemplazado «en caliente» por otra implementación del mismo, o por una versión mejorada de la misma implementación.

    Esto hace factible la corrección de subsistemas completos si se hace evidente que están operando por debajo de las especificaciones (exceso de uso de memoria, por ejemplo).

    Esta correcciones se pueden realizar «en caliente» lo que hace que el sistema completo no tenga que ser apagado.

    Por ejemplo, el servicio de generación de gráficos ChartServer, tiene dos configuraciones de despliegue disponibles: (1) desplegarlo en un servidor Web que corra código PHP, (2) desplegar el servidor compilado por la herramienta `hphp` (Facebook, 2010). Como el GDR 1.6.1 utiliza los servicios del ChartServer para generar los gráficos dentro de un reporte, es posible cambiar el servidor de ChartServer de la variante (1) a la (2) sin tener que apagar el sistema GDR 1.6.1, ni hacerle modificaciones de ningún tipo[16].
  - Ganar la máxima reutilización. Como la implementación de los subsistemas no depende directamente de ninguna implementación concreta de los subsistemas de los que depende, se puede reutilizar la misma funcionalidad sin cambios en otros sistemas.

Además, como el beneficio de una actualización de un subsistema se transfiere a cada uno de los sistemas, se incrementa el beneficio de forma proporcional.

Este grupo de recomendaciones se ha utilizado efectivamente en el proyecto «Ecumene Pyxel» (Vázquez Acosta, 2009, 2010b). De hecho, utilizando la misma infraestructura se logró la distribución de varios algoritmos reportados en (Castillo y otros, 2009; Chirino y otros, 2009). Uno de los motivos de estos resultados fue, precisamente, la creación un *middleware* como el que se propone en la capa de infraestructura para PATDSI.

2.7. Conclusiones parciales

En este capítulo se han descrito la arquitectura de referencia y el proceso que se utilizó para concebirla. Ambos elementos son importantes para el buen funcionamiento de la LPS.

Otro resultado del proceso fue una modificación sustancial de la descripción y el alcance de la línea PATDSI. Donde ha quedado claramente descrita la cantidad de activos que forman el núcleo de esta LPS.

Dentro de la arquitectura misma se ha especificado un grupo de principios que han sido utilizado con éxito en otros proyectos. Pero se han añadido elementos para poder representar la variabilidad de los productos de la línea y también para poder diseñar sistemas con demandas de tiempo real más exigentes.

En el próximo capítulo se explica cómo se ha ejecutado la construcción de esta arquitectura y también cómo se organizó el trabajo de departamento alrededor de la construcción de la LPS PATDSI.

[16] Evidentemente si un servicio se sustituye por otro con más funcionalidades, los clientes no utilizarán las nuevas funcionalidades hasta que no sean actualizados apropiadamente; pero no dejará de funcionar el sistema.

Capítulo 3

Implantación y validación de la arquitectura de referencia

En este capítulo se brinda una descripción del estado actual el avance en la implementación de la arquitectura presentada en el capítulo 2. Además, se realiza un análisis del impacto que ha tenido la implantación progresiva de la arquitectura de referencia en varios despliegues de productos de DATEC y otros productos de la UCI.

Se procede en el capítulo a describir cómo se realizaron las mediciones pertinentes para este capítulo. Luego se describe *grosso modo* la estrategia de implantación de la arquitectura de referencia. Finalmente se realiza brindan los resultados de la mediciones realizadas y se analiza el resultado.

3.1. Explicación del diseño del experimento

La implantación progresiva de la arquitectura presentada en este trabajo comenzó a finales del año 2009. Luego de unos seis (6) meses de trabajo ya se podían medir los primeros resultados.

Inicialmente se comenzó con un diagnóstico que revelaba el estado del departamento en relación a la variable Productividad. La fecha en que se realizó el diagnóstico fue en el mes de noviembre de 2009.

Luego se procedió a un levantamiento de todos los proyectos y sus compromisos de forma que se pudieran seleccionar los proyectos adecuados para la muestra del experimento. Para esto se desarrolló una estrategia más abarcadora que permitiera seleccionar, implantar y medir los resultados. Esta estrategia está detallada en las secciones siguientes.

Se marcó el final del experimento como las últimas semanas del mes de julio de 2010. En este momento se colectaron nuevamente todas las medidas necesarias para poder valorar el trabajo.

3.2. Estrategia de implantación de la arquitectura

Como se explicó en el epígrafe anterior, se desarrolló una estrategia para la implantación de la arquitectura que también influía en la selección de la muestra y los métodos de recopilación de datos utilizados.

Debido a que el departamento donde se ha desarrollado la línea PATDSI tenía, durante el curso de esta tesis, compromisos de entrega con varios clientes, la transición necesaria del estado anterior de la línea al propuesto en esta tesis se planificó de forma gradual con una estrategia fundamentalmente reactiva (Krueger, 2006, cf. 3); aunque, como se indica en el epígrafe 2.4, también se tuvieron elementos de pro-actividad, especialmente en los componentes más tecnológicos, como Caxtor y el componente de comunicación en tiempo real.

Para poder desarrollar el plan para instaurar la arquitectura de referencia se definieron un grupo de acciones que, progresivamente, cambiaron el estado del departamento y lo situaron en una posición más firme para seguir con la adopción del modelo de líneas de productos de software. Los siguientes pasos indican las acciones que fueron definidas como parte de la estrategia de implantación:

1. Identificar los proyectos que tenían compromisos inmediatos de entregas.

   En estos proyectos, además, se debía:

   - Analizar cada entregable comprometido
   - Analizar el estado de avance de los compromisos
   - Determinar la cantidad de personas necesarias para cumplir con los compromisos.
   - Redimensionar si fuera necesario el equipo del proyecto.

2. Identificar en cuáles de los proyectos existentes se podía comenzar el trabajo de la arquitectura de referencia.

   En este caso también era imprescindible:

   - Identificar los activos que pueden ser obtenidos en el proyecto en cuestión.
   - Identificar activos que pueden ser incorporados como objetivos a obtener dentro del proyecto.
   - Identificar si el proyecto necesita de activos construidos en otros, y eliminar de su plan todo lo relativo a la construcción de esa funcionalidad: en vez de eso, se planificarían actividades de integración frecuente del activo «externo».
   - Definir un plan de iteraciones y valorar el balance de la fuerza de trabajo.

3. Identificar qué proyectos se necesita crear para desarrollar activos necesarios para otros proyectos en desarrollo.

4. Identificar qué proyectos se necesita crear para el desarrollo de activos que sean necesarios para el establecimiento gradual de la línea PATDSI.

   En estos dos últimos pasos se necesita también definir un plan de iteraciones para centrar los esfuerzos del equipo y priorizar las características fundamentales de todos los activos que se construyan en el proyecto.

   En todos los casos se recomienda que *no* se cree un proyecto para construir cada activo por separado, a menos, que este sea suficientemente genérico. Lo recomendable es que crear agrupaciones

de activos según un criterio dado, el cual debería estar orientado hacia facetas del dominio de la LPS. En el caso de PATDSI esta agrupación está basada en las categorías dadas en la sección 2.5.

5. Establecer un plan de seguimiento para todos los planes de iteraciones, de modo que se asegure la necesaria coordinación entre los diferentes proyectos.

   Este plan de seguimiento incluía que:

   - Se mantuviera un registro continuo del avance en la implantación de la arquitectura, siguiendo los parámetros de cumplimiento de hitos en los proyectos que se crearon para materializar esta arquitectura. Esta medición se hizo de forma semanal.
   - Se registraran todas las entregas de productos realizadas de conjunto con el tiempo y el nivel de cumplimiento de los compromisos. En particular, se registró la cantidad de activos que fueron terminados y entregados como componentes a la red de centros de la UCI.

   Todos estos registros permiten calcular (1) el nivel de reutilización de los activos y también (2) el nivel de esfuerzo para integrar estos componentes en otros productos. De forma adicional se obtiene también una medida de la versatilidad de esta arquitectura para integrarse en productos diversos de la UCI.

Mediante la realización de las acciones indicadas en el listado anterior se tomaron varias decisiones: Por lo general, se mantuvieron varios de los equipos ya formados alrededor de productos comprometidos y tampoco se modificó el organigrama del departamento de forma substancial. Sin embargo, sí se alteró fundamentalmente el balance de proyectos.

3.2.1. Paso 1: Identificación de proyectos comprometidos

Antes de comenzar con la implantación de la arquitectura existían en el departamento seis proyectos:

- Uno de ellos estaba orientado a la creación de un sistema (no sus componentes) de análisis de datos y se pretendía que se reutilizara en varios proyectos. Sin embargo, al analizar los resultados parciales de ese proyecto, se pudo constatar que no capturaba las necesidades del resto de los productos de PATDSI; y tampoco se estaba realizando un trabajo arquitectónico que asegurara la integración.
- Otro, cuya versión 1.5 sí se estaba reutilizando, era el Generador Dinámico de Reportes. Al comienzo de este trabajo es versión acababa de ser liberada, y el proyecto estaba en mantenimiento. No se había realizado un plan para incluir otras funcionalidades.
- El resto de los proyectos no contribuían activos reutilizables a PATDSI, sino que eran básicamente sistemas hechos a la medida del cliente.

De estos proyectos se identificaron cuatro en los cuales no se podía, por su contenido o nivel de compromiso, introducir la arquitectura propuesta:

- El Sistema de Gestión Estadística para la ONE
- La personalización de PATDSI para el MENPET
- La personalización de PATDSI para el MINCI
- La personalización de PATDSI para el MIJ.

3.2.2. Paso 2: Identificación de proyectos factibles

El proyecto para el GDR, que si bien tiene un impacto importante en varios de los otros proyectos, no se encontraba en desarrollo activos, se seleccionó para comenzar a introducir elementos de la arquitectura de referencia en su próxima versión. Es decir, se cerró finalmente el proyecto Generador de Reportes 1.5 y se comenzó el proyecto Generador Dinámico de Reportes 1.6.1.

El plan de iteración para el GDR 1.6.1 fue cuidadosamente revisado y se centró en asegurar cualidades para el producto que se consideraban imprescindibles para que el sistema fuera aceptado por los clientes en cuatro contratos que se habían sido aprobados en la IX Comisión Mixta Cuba-Venezuela.

Dentro de los objetivos de este proyecto no se podía incluir la refactorización del sistema para desglosarlo en los activos básicos propuestos en la sección 2.5 para este caso. Comenzar una refactorización del sistema en ese contexto sería muy arriesgado para cumplir los compromisos contraídos. Por esta razón, la refactorización de este sistema se pospuso.

La otra acción que se realizó al revisar todos los proyectos fue el cierre del proyecto de análisis de datos.

3.2.3. Paso 3: Creación de proyectos para desarrollar activos

Para dar soporte a la nueva versión de GDR 1.6.1 y los futuros productos PATDSI se crearon proyectos orientados a la construcción de varios de los activos de PATDSI. Esto se corresponde con el segundo nivel propuesto por (Krueger, 2006, 2007), que básicamente orienta la producción en función de los activos. Sin embargo, es notable precisar que existen diferencias, porque, en primer lugar, no se contaba con una producción automática de los productos; sino que la composición de activos para crear productos todavía es una labor manual y la responsabilidad de algún equipo de trabajo.

El criterio utilizado para la creación de estos proyectos y la asignación de su plan de trabajo fue, en muchos casos, un resultado directo de buscar respuestas a las necesidades de los productos en construcción, y también a necesidades globales de la línea. Los siguientes proyectos fueron creados para resolver parte de este problema:

- El ChartServer. En este caso, se trataba de un activo de visualización de información en forma de gráficos estáticos, que se utilizaría tanto en sistemas de reportes, y en otros sistemas donde se pudiera incluir una imagen estática en la interfaz del usuario.
- El R-Server. Este proyecto sustituyó al de análisis de datos. Todos los miembros del antiguo proyecto pasaron al nuevo, y se cambió la dirección del mismo. Además se estableció un nuevo plan de iteraciones

Además de estos proyectos, se creó un proyecto llamado "Sistemas de Captura de Datos" con el objetivo de refactorizar en lo posible el actual producto SIGE y construir, a partir de este, los activos básicos de captura de datos. No obstante, como SIGE tenía entregas próximas, se mantuvo SIGE también como proyecto. Ambos proyectos coexistirían, pero tendrían equipos y misiones diferentes.

El proyecto de captura de datos tenía dos hitos fundamentales:

- La creación de aquellos componentes de captura de datos que permitieran satisfacer los requisitos del módulo de encuestas demográficas de SIGE.
- La creación de componentes de captura de datos que permitieran reconstruir completamente el sistema SIGE, importar los datos capturados por versiones anteriores de SIGE, y también la creación de otros productos con necesidades de captura de datos mediante formularios web.

Para la primera etapa de este proyecto, se les dio la tarea de construir además activos tecnológicos, como los componentes de trabajo desconectado en la Web.

3.2.4. Paso 4: Creación de proyectos para activos tecnológicos

Finalmente se crearon dos proyectos con vistas a las perspectivas futuras de la línea y también para asegurar la integración de diversos componentes en aplicaciones complejas:

- La plataforma y el IDE Caxtor. La plataforma Caxtor permitía la construcción de componentes de interfaz de usuario de forma independiente y luego integrarlos con los mecanismos previstos en esta tesis, incluyendo los *puntos de entrada* para representar la variabilidad de los productos.

  El IDE Caxtor es una facilidad para diseñar más rápidamente las interfaces de usuario, de modo que agilizara el proceso de desarrollo de los componentes básicos.
- Un proyecto para la creación de componentes de comunicación bidireccional en la Web, y la actualización en tiempo de real de gráficas con indicadores.

  El componente de comunicación en tiempo real permitirá que las aplicaciones de captura de datos envíen notificaciones en tiempo real a los componentes de visualización, de modo que se agilice la toma de decisiones en contextos donde la información cambia frecuentemente.

Los componentes de gráficos vivos son una aplicación directa de este componente en los Cuadros de Mando Integrales, donde un indicador, que se muestre en un gráfico, puede estar siendo constantemente actualizado.

3.2.5. Paso 5: Seguimiento y control

Para poder llevar un registro continuo de lo que sucedía en los proyectos se estableció un rol específico dentro de cada grupo del departamento para registrar semanalmente lo previsto en el paso 5 de la estrategia. Los registros se mantuvieron de forma semanal y se chequeaban con la misma periodicidad.

Adicionalmente el autor de este documento, se encontraba diariamente con alguno de los equipos como asesor técnico y revisaba también los principales problemas para lograr los objetivos planteados en los planes de iteración de cada proyecto.

3.3. Resultados del trabajo

3.3.1. Impacto inicial de la aplicación de la estrategia

La introducción de los cambios presentados en los párrafos anteriores tuvo un profundo impacto en el balance del contenido de trabajo de la línea PATDSI.

En primer lugar, implicó que la relación de proyectos que construyen activos pasó de representar solamente el 20 % de los proyectos de la línea a cubrir aproximadamente el 57,14 % del total de estos. Este primer resultado se ilustra en la figura 3.1.

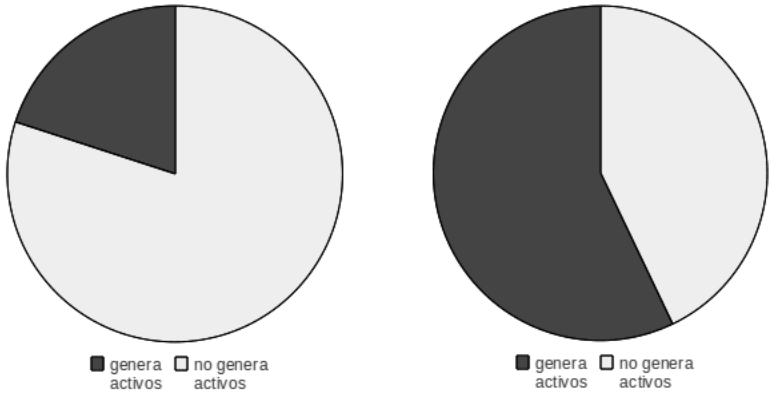

Figura 3.1: Cambio del balance de proyectos que generaban activos antes de implantar la estrategia presentada (figura de la izquierda), y después de implantada (figura de la derecha).

En segundo lugar, este cambio en el balance de proyectos permitió ir construyendo e implantando progresivamente la arquitectura de referencia para PATDSI. La importancia de este resultado es vital para la creación de una LPS, debido a que no se pudiera hablar de activos si no se dedican recursos a su construcción y mantenimiento (Crnkovic y Larsson, 2002).

En los siguientes epígrafes se podrá observar cómo los proyectos que estaban más orientados a la construcción de los componentes reutilizables tuvieron un avance sostenido durante su desarrollo, y también cómo la integración de estos componentes en otros productos fue mucho más eficiente.

3.3.2. Análisis de resultado respecto a la variable dependiente

En la sección anterior se comentó sobre el estado del balance de los proyectos que contribuían activos a la línea PATDSI contra aquellos que no lo hacían. En esta sección se brindan los detalles sobre este y otros indicadores relacionados con la productividad de la línea.

Al comienzo de este trabajo, solamente se reutilizaba como subsistema, el Generador Dinámico de Reportes, en su versión 1.6.1. También se reutilizaba en todos los productos el marco ExtJS, pero este no es un activo mantenido por el propio DATEC sino que se descarga desde el sitio de la empresa Sencha Inc.

Este nivel de reutilización en función de la cantidad de proyectos de la línea era de un 20 %. El resto de los proyectos estaban orientados a la creación de SIGE, el proyecto de integración de para el MIJ, el sistema de gestión de proyectos para el MINCI[17], y la personalización del sistema de reportes para el MENPET.

El proyecto SIGE se estaba gestionando no como productor de activos reutilizables, sino un proyecto que resolviera las necesidades de la Oficina Nacional de Estadísticas. No se podían utilizar sus partes como componentes porque no se habían diseñado en función de reutilizarlo en otros sistemas. Los otros dos proyectos son proyectos que, básicamente, integrarían al GDR 1.6.1 (que se comenzaba a desarrollar) pero no aportaban activos a la línea.

El GDR 1.5 se estaba utilizando activamente:

- El ERP cubano Cedrux desarrollado por el Centro CEIGE. El GDR 1.5 es el sistema que se utiliza para la generación de todos los reportes del ERP.

  Sin embargo, el GDR 1.5 carecía de características importantes, como la integración con el sistema de seguridad del ERP y otras.

- Este sistema se estaba usando además integrado al sistema Redmine que usaba DATEC para la gestión de sus proyectos.

  Sin embargo, esta versión no permitía la interfaz de integración con el Redmine fuera amigable.

[17] La gestión de este proyecto luego pasó a otro departamento, aunque uno de sus entregables era el GDR 1.6.1 que se encontraba en desarrollo en la línea PATDSI.

La tabla siguiente resume estado de la variable dependiente «Productividad» antes de comenzar con la implantación de la arquitectura y luego de implantados varios de sus principios en los proyectos de la línea.

Cuadro 3.1: Mediciones sobre la variable dependiente «Productividad de la línea» antes y luego de implantar la arquitectura.

Dimensión	**Indicador**	**Antes de implantar la solución.**	**Luego de implantar la solución.**
Reutilización	Nivel de reutilización media por componente.	13,89 %	23,46 %
Integración de productos y componentes.	Esfuerzo medio de integración (en horas- hombres).	54,65	44,29

Las fuentes utilizadas para extraer los datos mostrados son las bases de datos de los registros del centro DATEC desde inicios del año 2009 hasta finales del 2010. El esfuerzo de integración se ha estimado a partir del registro de tiempo dedicado a las tareas durante los periodos donde se estaban realizando las integraciones entre varios productos.

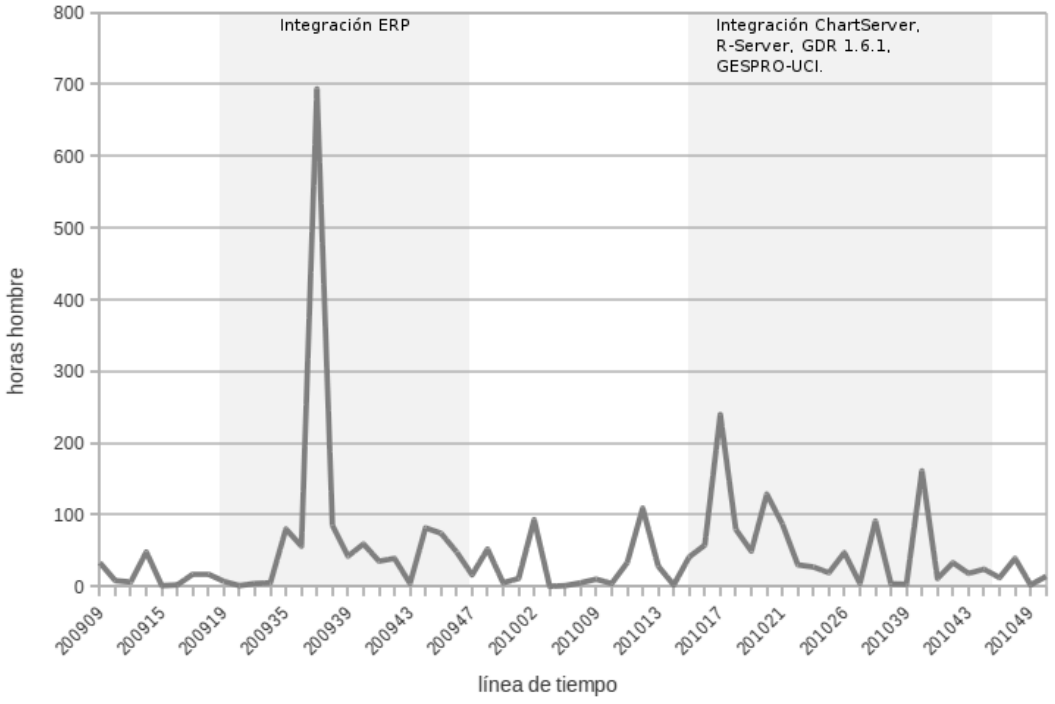

Figura 3.2: Línea del tiempo del esfuerzo empleado en la línea desde el 2009 hasta el 2010.

La gráfica 3.2 muestra el esfuerzo total de la línea durante ese tiempo. En la misma figura se señalan los momentos en que hubo mayor carga de trabajo en relación a las actividades de integración dentro de la línea. No obstante se debe señalar que parte de la carga de trabajo que sobresale en el periodo de integración con el ERP se debe al cierre de una fase del proyecto SIGE. Al inicio de implantar una LPS se puede esperar este tipo de elevación en el esfuerzo, debido a que es necesario refactorizar o rediseñar componentes para adecuarlos al nuevo contexto. No obstante, es notable que luego de aplicadas algunos principios el esfuerzo redujo en casi 10 horas hombres como promedio.

3.3.3. Análisis de la variable independiente

Los resultados mostrados con anterioridad se han obtenido a partir de la aplicación de solo una parte de lo propuesto en este trabajo. En particular, se han aplicado los siguientes componentes:

- El ChartServer
- El R-Server
- El GDR 1.6.1

De los principios arquitectónicos que se plantean el capítulo 2, se ha comenzado a aplicar (aunque de forma) parcial los siguientes:

- Los conectores RPC como mecanismo fundamental de comunicación entre el cliente y el servidor.
- Componer las aplicaciones servidoras como una red de servicios desconectados. En este caso, solamente se han logrado dos de estos servicios. Sin embargo, estos han sido exitosamente integrados al resto de los productos.
- Componer las aplicaciones clientes de forma ascendente. En este caso, solamente se ha aplicado en partes del GDR 1.6.1 que han sido modificadas durante el 2010. El resto del sistema aún permanece con el esquema anterior.

En este sentido la variable «Arquitectura de referencia» se encuentra en el estado que se detalla a continuación:

Cuadro 3.2: Estado de la variable independiente «Arquitectura de referencia» al finalizar la investigación.

Dimensión	**Indicador**	**Medida**
Conceptual	Definición	Sí
	Implementación	75 %
Organizativa	Nivel de introducción a nivel de departamento.	33.34 %
	Nivel de introducción en la muestra del experimento.	100 %
Activos	Activos definidos	11
	Por ciento de implementación	36.36 %

La tabla implica que solo una tercera parte de lo que conceptualmente ha sido definido en la arquitectura de referencia está siendo explotado para obtener los resultados que se mostraron con anterioridad. No obstante, el avance de los proyectos que se encargan de la implementación de los activos tecnológicos que incorporan todos los principios conceptuales definidos ya se encuentra por encima de la mitad. En particular:

- El proyecto Caxtor ya cuenta con un mecanismo de composición ascendente para las aplicaciones clientes y se basa siempre en el principio de un sistema cliente-servidor. De hecho, el IDE Caxtor funciona casi completamente en el navegador y solo en dos casos de uso necesita comunicarse con el servidor para realizar alguna acción.
- También en Caxtor se pueden definir los *puntos de entrada* para representar e implementar la variabilidad de los productos.
- En el proyecto de capturas de datos ya se tiene implementado el mecanismo de conexión por RPC. Esta implementación aún no utiliza el activo Apache Thrift; no obstante, como principio arquitectónico ya esta implementación puede comenzar a explotarse.
- Tanto el ChartServer como R-Server tienen versiones en explotación, e integradas a otros productos.
- En el proyecto para la creación de tableros digitales ya se tiene una implementación del componente de comunicación en tiempo real, y varios de los gráficos vivos funcionan a nivel de prototipos.

3.3.4. Generalización de los resultados en otros escenarios en la UCI

La arquitectura presentada ha contribuido un grupo de activos, los cuales se puede generalizar en varios productos de la UCI más allá del propio DATEC. La facilidad para generalizar estos productos, aunque en sí misma no constituye objetivo de este trabajo, sí constituye un aporte adicional del trabajo realizado.

En esta sección se brindan datos sobre la integración de varios de los activos que forman la arquitectura de referencia en otros productos de la UCI.

La forma en que se han hecho varios de los componentes PATDSI hace factible su utilización en diversos proyectos fuera del marco del centro DATEC. De hecho, es un objetivo de la universidad que sus productos, sistemas y componentes puedan reutilizarse en varios de proyectos y que los centros puedan especializarse en estos componentes.

Tres activos de la arquitectura propuesta se han desplegado en la UCI y se ha podido comprobar que se están utilizando activamente.

El Generador Dinámico de Reportes se ha integrado al sistema de gestión de proyectos GESPRO que se ha desplegado en todos los centros de la UCI, incluyendo los que radican fuera de la cede central.

Según los datos que conserva la Dirección Técnica de la UCI, para realizar la primera versión de GESPRO solo se necesitaron cuarenta y cinco días con un equipo de diez personas. En otras versiones, ya el esfuerzo fue mucho menor. La gráfica 3.3 muestra cómo, para distintas versiones del GESPRO ha disminuido el esfuerzo necesario.

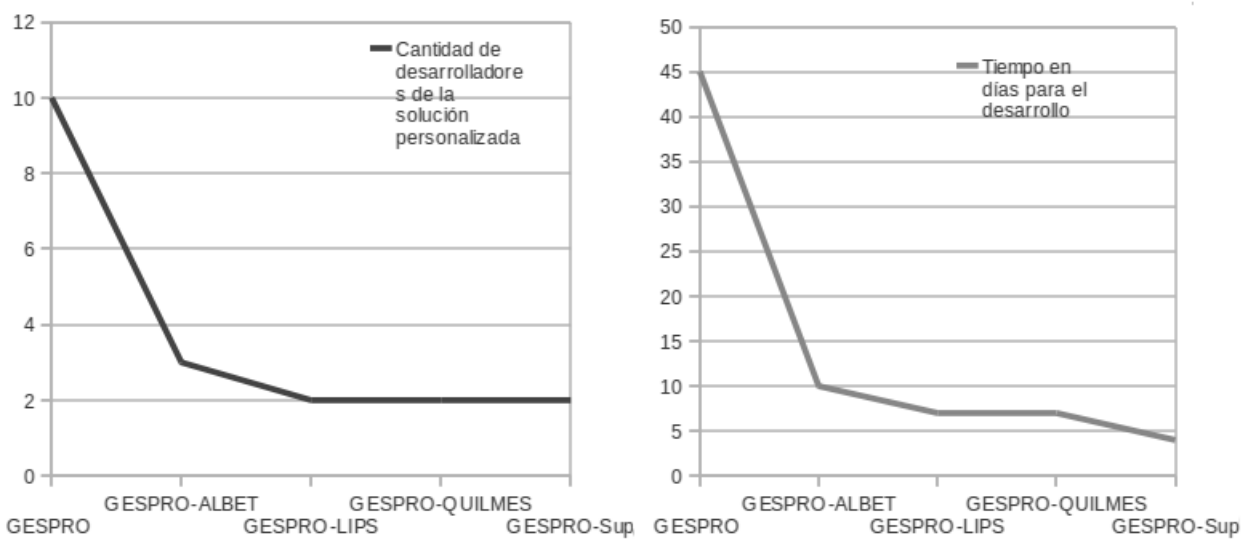

Figura 3.3: Reducción del esfuerzo para implantar las versiones de GESPRO.

Independientemente de esta disminución se puede notar también que el esfuerzo empleado inicialmente, dado volumen de sistemas a integrar, y la cantidad de centros en cuales se desplegó la solución, el esfuerzo no fue

superior a las 285,71 horas-hombres por centro desplegado. La integración del GDR a este resultado se puede considerar una prueba positiva sobre el principio de integración a nivel de servicios, aunque aún subsista la integración de datos en este caso.

Otro de los componentes que se ha desplegado en la UCI ha sido el ChartServer, el cual se encuentra incorporado a las herramientas de GESPRO y, por tanto, es accesible desde el portal de cada centro; y también se puede acceder por la URL http://graficos.prod.uci.cu/.

Al analizar el nivel de uso de este componente durante un periodo de tiempo se puede notar que está siendo constantemente utilizado. El sistema de monitorización de los servicios de la Dirección Técnica[18], puede ser utilizado para extraer información sobre el uso de esta herramienta.

Se puede observar que durante todo el tiempo en que la herramienta de monitorización lleva funcionando la herramienta ha tenido un elevado nivel de uso.

La gráfica 3.4 muestra la cantidad de accesos diarios que ha tenido la herramienta desde noviembre de 2010 hasta enero de 2011. Es notable un descenso luego período vacacional de invierno (en cual se señala con un fondo más oscuro), pero en términos absolutos, siguen siendo más de doscientas peticiones diarias como promedio.

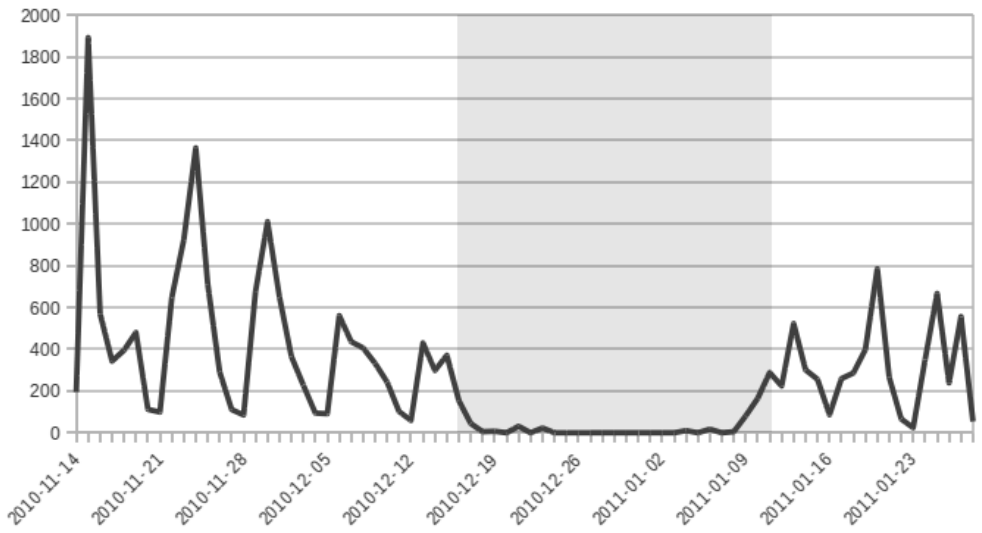

Figura 3.4: Nivel de utilización de la herramienta ChartServer desplegada en la UCI.

Otro indicador que ofrece resultados interesante es el modo de acceso que tiene la herramienta. Para esto se analizaron los registros del servidor Apache donde está desplegada la herramienta y se pudo obtener un estimado del origen desde donde se realizaban los accesos. La tabla siguiente muestra los orígenes desde los cuales los usuarios han accedido al ChartServer.

[18] Esta herramienta, para el caso del uso del ChartServer, puede consultarse en la dirección http://awstats.dt.prod.uci.cu/hosting_chartserver/.

Cuadro 3.3: Resumen del nivel de uso por orígenes que tiene el servidor de gráficos, basado en el activo Chart Server

Orígenes	**Accesos**
graficos.prod.uci.cu	2.774
facultad3.uci.cu	180
Desconocido	172
localhost	91
portal.cenia.prod.uci.cu	48
portal.fortes.prod.uci.cu	42
portal.ceige.prod.uci.cu	41
primavera.uci.cu	37
portal.cedin.prod.uci.cu	34
portal.isec.prod.uci.cu	34
portal.datec.prod.uci.cu	34
portal.cesim.prod.uci.cu	32
portal.geysed.prod.uci.cu	28
portal.cegel.prod.uci.cu	17
portal.geitel.prod.uci.cu	15
portal.sitel.prod.uci.cu	11
portal.cised.prod.uci.cu	7
portal.cdae.prod.uci.cu	7
portal.calisoft.prod.uci.cu	5
portal.dt.prod.uci.cu	2
portal.prod.vcl.uci.cu	2
portal.curso.prod.uci.cu	1
portal.albet.prod.uci.cu	1
portal.cice.prod.uci.cu	1
orion.prod.uci.cu	1

Si bien la mayoría de los accesos se originan desde el mismo sitio del ChartServer, se puede notar que se han originado peticiones desde todos los despliegues de GESPRO en los centros de la cede central de la UCI, y también en otros departamentos y centros regionales. Llama la atención también el cuarto renglón de la tabla, cuyo origen es «localhost». Una inspección minuciosa de estos registros reveló existían desarrolladores que estaban utilizando este servicio para sus proyectos.

3.4. Discusión de los resultados y conclusiones parciales

Los resultados plasmados en las secciones anteriores muestran que la arquitectura propuesta en este trabajo, de conjunto con su implantación, han logrado aumentar la productividad de la línea en las dos dimensiones planteadas al inicio de este trabajo.

Si bien es mucho más notable el aumento en el nivel de reutilización, tampoco se puede restar importancia al resultado de la reducción del esfuerzo. Este resultado, aunque menor, es consecuencia directa de que todavía la arquitectura propuesta no ha sido completamente implementada y muchos menos introducida en todos los proyectos de la línea.

La evidencia sugiere que la dimensión organizativa de la variable «Arquitectura de Referencia» es la que domina (por encima de las dimensiones conceptual y la de los activos) en la influencia de los resultados finales del componente de esfuerzo en la variable «Productividad». Los datos con los que se cuentan no permiten, todavía, realizar una prueba conclusiva al respecto por lo que se deja esta cuestión como una pregunta para responder en futuras investigaciones. Por otra parte, la reutilización en sí misma elimina actividades que antes eran necesarias y esto influye también en la poca variación observada en este indicador.

La hipótesis planteada al inicio del trabajo requería que la arquitectura fuera *definida* y *establecida*. Al concluir esta investigación ya se tiene el 75 % de la arquitectura definida e implementada. Esta fue establecida en el 100 % de los proyectos seleccionados para la muestra, pero sólo se ha establecido en menos del 35 % de los proyectos de todo el departamento. Independientemente de esto, ya produce resultados positivos en la productividad. Es factible pensar que la *definición* de la arquitectura es una condición necesaria (aunque no suficiente) para obtener mejoras en la productividad de la línea.

La estrategia planteada en la sección 3.2 es un aporte secundario de este trabajo, que complementa la definición de la arquitectura para lograr su implantación progresiva dentro de la línea de productos.

Finalmente, varios de los activos definidos por la arquitectura dada han probado ser reutilizable incluso fuera del marco de DATEC. Estos activos se han podido generalizar a proyectos del resto de la UCI, como el GESPRO. También el ChartServer se ha estado utilizando de alguna manera por muchos de los centros de la UCI.

Conclusiones

Este trabajo expone una arquitectura de referencia para la creación de productos en la línea de productos de software «Paquete de Herramientas para la Toma de Decisiones y Sistemas Inteligentes» (PATDSI).

La arquitectura propuesta ha sido parcialmente implementada y se han obtenido los activos fundamentales siguientes:

- ChartServer
- R-Server
- Generador Dinámico de Reportes 1.6.1
- Plataforma Caxtor versión 1.0.1 beta.

La arquitectura propuesta trabaja en el nivel más bajo del modelo propuesto por (Krueger, 2007). Al contemplar un mecanismo explícito para establecer colaboraciones de primer nivel (sección 2.6.2), las cuales representan puntos de variabilidad en los productos, se promueve un mecanismo de variabilidad que puede ser manejable, y no es diferente para cada producto.

En este sentido aún se puede mejorar el mecanismo propuesto para incluir la variabilidad en los componentes servidores de la arquitectura de referencia. También se puede adoptar completamente el modelo ortogonal de variabilidad propuesto por (Pohl y otros, 2005) y explotado exitosamente en (Pérez Lamancha y Polo Usaola, 2009).

Dentro de las definiciones concretadas en esta tesis, la descripción de muchos de los activos de PATDSI es un paso de avance en la definición de la LPS en sí misma. Se ha comenzado por la detección de subsistemas completos para la visualización de información, en el cual los componentes para la generación de reportes han sido radicalmente reformulados; la concepción de usar representaciones de modelos de datos como el mecanismo viabilizador de la integridad de los productos; la identificación de componentes concretos como el ChartServer y el R-Server, y otros que, si bien están identificados, aún no se ha podido comenzar su definición concreta.

Para la realización de la propuesta fue necesario formular una estrategia que permitiera la aplicación progresiva de la arquitectura dentro de la línea sin afectar los compromisos establecidos con los clientes. Esta estrategia permitió implementar el 75 % de la arquitectura en el 100 % de los proyectos seleccionados para la muestra. Solamente se estableció en menos del 35 % de los proyectos del departamento, puesto que abarcar más suponía un riesgo intolerable.

Por otra parte, parte de los activos creados están siendo explotados efectiva y continuamente por usuarios de toda la UCI. Lo cual es un indicador de la baja curva de aprendizaje de los mismos para su integración.

El resultado fundamental que se observa en esta tesis es un *aumento de la productividad de la línea*, expresado por el nivel de reutilización de los componentes, y la disminución del esfuerzo.

De forma adicional, se han brindado un grupo de acciones que se debieron realizar para poder introducir los cambios necesarios en la LPS sin interrumpir los compromisos contraídos con los clientes.

Es también importante señalar que se ha podido generalizar el uso de varios de los activos definidos en este trabajo. El GDR 1.6.1 ha sido desplegado para todos los centros de la UCI en el sistema GESPRO. Este mismo sistema también incorpora al ChartServer, y mediante esto, se ha podido comprobar un elevado nivel del uso de esta herramienta.

Recomendaciones

Al concluir este trabajo es importante detallar varios elementos en los cuales el autor considera que se puede trabajar en el futuro:

- DATEC debe mantener un trabajo permanente en el tema de las líneas de productos de software y trabajar intensivamente porque sus directivos apliquen conocimientos de este tema.
- Se debe mantener el sistema de seguimiento de los proyectos que construyen activos de modo que se complete el 100 % de la arquitectura definida.
- Se deben comenzar investigaciones para incluir los formalismos para representar la variabilidad y también para automatizar la derivación de productos a partir de los activos.
- Se deben seguir publicando periódicamente los avances y actualizaciones de los activos para que la UCI pueda beneficiarse de las mejoras introducidas en futuros desarrollos.
- No se debe dejar de prestar atención al control y seguimiento de los proyectos involucrados con la introducción de cambios a la arquitectura de referencia; puesto que es notable que la reducción del esfuerzo general empleado puede depender mucho del nivel de introducción de estos cambios en toda la línea.

Bibliografía

Security Assertion Markup Language, versión 2. Documento electrónico, 2005. URL http://saml.xml.org/saml-specifications.

Joe Armstrong. *Making reliable distributed systems in the presence of software errors*. Tesis doctoral, The Royal Institute of Technology Stockholm, Sweden, Diciembre 2003.

Len Bass, Paul Clements, Sholom Cohen, Linda Northrop, y James Withey. Product line practice workshop report. Reporte técnico CMU/SEI-97-TR-003, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, Junio 1997.

Len Bass, Paul Clements, y Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2nd edición, Abril 2003. ISBN 0-321-15495-9.

John Bergey, Paul Clements, Sholom Cohen, Pat Donohoe, Larry Jones, Bob Krut, Linda Northrop, Scott Tilley, Dennis Smith, y James Withey. DoD product line practice workshop report. Reporte técnico CMU/SEI-98-TR-007, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890, Mayo 1998.

John Bergey, Grady Campbell, Paul Clements, Sholom Cohen, Lawrence Jones, Robert Krut, Linda Northrop, y Dennis Smith. Second DoD product line practice workshop report. Reporte técnico CMU/SEI-99-TR-015, ESC-TR-99-015, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Octubre 1999.

Grady Booch. *Object-Oriented Analysis and Design with applicatios*. Addison-Wesley, 2nd edición, 1998. ISBN 0-8053-5340-2.

Jan Bosch y Judith A. Stafford. Architecting component-based systems. En Crnkovic y Larsson (2002). ISBN 1-58053-327-2.

Frederick P. Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20: 10–19, 1987.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, y Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, volumen 1 of *Software Design Patterns*. John Wiley & Sons, Chichester, England, 1996. ISBN 0-471-95869-7.

Yoannia Castillo, Raidel Arenas, y Joel Armada. Implementación de algoritmos de recomendaciones de imágenes para textos noticiosos. Trabajo de diploma TD-2394-09, Universidad de las Ciencias Informáticas, 2009.

Gary J. Chastek, Patrick Donohoe, y John D. McGregor. Formulation of a production strategy for a software product line. Nota técnica CMU/SEI-2009-TN-025, Software Engineering Institute, Carnegie Mellon University, Agosto 2009.

Javier Chirino, Adrián Alberto Mesa, y Ailyn Alfonso. Implementación de algoritmos de reconocimiento de imágenes duplicadas. Trabajo de diploma TD-2395-09, Universidad de las Ciencias Informáticas, 2009.

Paul Clements y Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, 2001. ISBN 0-201-70332-7.

Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, y Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002. ISBN 0-201-70372-6.

Paul C. Clements, Lawrence G. Jones, Linda M. Northrop, y John D. McGregor. Project management in a software product line organization. *IEEE Softw.*, 22(5):54–62, 2005. ISSN 0740-7459. doi: 10.1109/MS.2005.133.

Ivica Crnkovic y Magnus Larsson, editores. *Building Reliable Component-Based Software Systems*. Artech House, Norwood, MA, 2002. ISBN 1-58053-327-2.

Ivica Crnkovic, Brahim Hnich, Torsten Jonsson, y Zeynep Kiziltan. Basic concepts in CBSE. En Crnkovic y Larsson (2002), capítulo 1, páginas 3–22. ISBN 1-58053-327-2.

O. J. Dahl, E. W. Dijkstra, y C. A. R. Hoare. *Structured Programming*. A.P.I.C. Studies in Data Processing, No. 8. Academic Press, 1972. ISBN 0-12-200550-3.

César de la Torre Llorente, Unai Zorrilla Castro, Miguel Ángel Ramos Barros, y Javier Calvarro Nelson. *Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0*. Krasis Press, Marzo 2010. ISBN 978-84-936696-3-8. Borrador.

Sijbren Keimpe Deelstra y Marco Sinnema. *Managing the Complexity of Variability in Software Product Families*. Tesis doctoral, Rijksuniversiteit Groningen, 2008.

ECMA International. ECMAScript language specification, 2009.

Facebook. Hiphop for php. Página Web, 2010. URL http://github.com/facebook/hiphop-php.

Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Tesis doctoral, University of California, Irvine, 2000. URL http://www.ics.uci.edu/fielding/pubs/dissertation/top.htm. Sitio web consultado el 2009-08-15.

David Garlan y Mary Shaw. An introduction to software architecture. Reporte técnico CMU-CS-94-166, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1994.

David Garlan, Robert Monroe, y David Wile. Acme: an architecture description interchange language. En *CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, página 7. IBM Press, Noviembre 1997.

L. Hotz, K. Wolter, T. Krebs, S. Deelstra, M. Sinnema, J. Nijhuism, y J. MacGregor. *Configuration in Industrial Product Families. The ConIPF Methodology.* IOS Press, 2006. ISBN 978-1-58603-641-6.

Andrew Hunt y David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. The Pragmatic Bookshelf. Addison-Wesley, 1999. ISBN 0-201-61622-X.

Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, y John Irwin. Aspect-oriented programming. En *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. Springer-Verlag, 1997.

John Klein y David Weiss. What is architecture? En Diomidis Spinellis y Georgios Gousios, editores, *Beautiful Architecture*. O'Reilly, 1 edición, Enero 2009. ISBN 978-0-596-51798-4.

Donald Knuth. Literate programming (1984). En *Literate Programming*, capítulo 4, página 99. Stanford, California: Center for the Study of Language and Information, 1992. URL http://www-cs-faculty.stanford.edu/~uno/lp.html.

Charles W. Krueger. New methods in software product line practice. *Communications of the ACM*, 49(12): 37–40, 2006. ISSN 0001-0782. doi: 10.1145/1183236.1183262.

Charles W. Krueger. The 3-tiered methodology: Pragmatic insights from new generation software product lines. *Software Product Line Conference, International*, 0:97–106, 2007. doi: 10.1109/SPLINE.2007.34.

Charles W. Krueger. Catalysts and inhibitors for momentum in the software product line industry. En *12th International Software Product Line Conference*, página 365, Los Alamitos, CA, USA, 2008a. IEEE Computer Society. ISBN 978-0-7695-3303-2. doi: 10.1109/SPLC.2008.48.

Charles W. Krueger. Pragmatic methods for commercial software product line engineering practice. En *12th International Software Product Line Conference*, volumen 0, página 376, Los Alamitos, CA, USA, 2008b. IEEE Computer Society. ISBN 978-0-7695-3303-2. doi: 10.1109/SPLC.2008.66.

Charles W. Krueger, Dale Churchett, y Ross Buhrdorf. Homeaway's transition to software product line practice: Engineering and business results in 60 days. En *Proceedings of the 12th International Software Product Line Conference*, páginas 297–306, 2008.

Craig Larman. *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Prentice Hall, 2002.

Kim Lauenroth y Klaus Pohl. Principles of variability. En Pohl y otros (2005), capítulo 4, páginas 57–88. ISBN 3-540-24372-0.

Marco Lormans y Arie van Deursen. Can LSI help reconstructing requirements traceability in design and test? En *Proc. of the 10th European Conf. on Software Maintenance and Reengineering*, páginas 47–56. IEEE Computer Society, 2006.

Frank Lüders, Kung-Kiu Lau, y Shui-Ming Ho. Specification of software components. En Crnkovic y Larsson (2002), capítulo 2, páginas 23–38. ISBN 1-58053-327-2.

David Martínez Alarcón, Iliannis Pupo Leyva, Manuel Vázquez Acosta, Dioleisy Fontela González, y Michael Hernández Martínez. IDE Caxtor: Constructor de aplicaciones web. En *I Taller de Ciencias Informáticas aplicadas a la gestión empresarial. Memorias del evento UCIENCIA*, 2010.

Jonás A Montilva. Desarrollo de software basado en líneas de productos de software. Presentación, 2006.

Linda Northrop y Paul Clements. A framework for software product line practice, version 5.0. Sitio web, Julio 2007a. URL http://www.sei.cmu.edu/productlines/frame_report/. Consultado en enero de 2010.

Linda Northrop y Paul Clements. Core assets development. En *A Framework for Software Product Line Practice, Version 5.0* Northrop y Clements (2007a). URL http://www.sei.cmu.edu/productlines/frame_report/coreADA.htm. Consultado en agosto de 2010.

Linda Northrop y Paul Clements. All three together. En *A Framework for Software Product Line Practice, Version 5.0* Northrop y Clements (2007a). URL http://www.sei.cmu.edu/productlines/frame_report/all_three.htm. Consultado en agosto de 2010.

Linda M. Northrop y Lawrence G. Jones. Introduction to software product line adoption. En *SPLC '08: Proceedings of the 2008 12th International Software Product Line Conference*, páginas 371–372, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3303-2. doi: 10.1109/SPLC.2008.63.

Bashar Nuseibeh y Steve Easterbrook. Requirements engineering: a roadmap. En *Proceedings of the Conference on The Future of Software Engineering*, páginas 35–46. ACM Pres, Limerick, Ireland, 2000.

Jorge Octavio Ocharán Hernández y Karen Cortes Verdin. Documentando arquitecturas orientadas a aspectos para líneas de productos de software. Reporte técnico, Universidad Veracruzana, 2010. URL http://www.uv.mx/personal/kcortes/files/2010/08/OcharanCortes.pdf. Consultado en septiembre de 2010.

Dewayne E Perry y Alexander L Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, Octubre 1992.

Henrik Pestano y Pedro Piñero Pérez. Propuesta de modelo de desarrollo para líneas de productos de software en centros de producción. Tesis de maestría, Universidad de las Ciencias Informáticas, 2011. (Consultado el borrador en noviembre de 2010.).

Pedro Yobanis Piñero Pérez y Maykel Yelandi Leyva Vázquez. Conferencia no. 1, modelos de desarrollo de software. Presentación Digital en PowerPoint, 2010.

Klaus Pohl, Günter Böckle, y Frank var der Linden, editores. *Software Product Line Engineering. Foundations, Principles, and Techniques.* Springer Verlag, Berlín, Alemania, 2005. ISBN 3-540-24372-0.

Project Management Institute. *Guía de los Fundamentos de la Dirección de Proyectos (Guía del PMBoK)*. Project Management Institute, 3rd edición, 2004.

Project Management Institute. *A Guide to the Project Management Body of Knowledge*. PMI Press, 2008.

Beatriz Pérez Lamancha y Macario Polo Usaola. Pruebas basadas en modelos para líneas de producto software. *Revista Española de Innovación, Calidad e Ingeniería del Software*, 5(2):17–27, 2009.

Trygve Reenskaug. Thing-Model-View-Editor, an example from a planning system. Note, Xerox PARC, Mayo 1979. URL http://heim.ifi.uio.no/trygver/themes/mvc/mvc-index.html. Consultado el 2009-12-01.

Ravi Sandhu, David Ferraiolo, y Richard Kuhn. The NIST model for role-based access control: towards a unified standard. En *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, páginas 47–63, New York, NY, USA, 2000. ACM. ISBN 1-58113-259-X. doi: 10.1145/344287.344301. URL http://doi.acm.org/10.1145/344287.344301.

Marco Sinnema y Sybren Deelstra. Industrial validation of covamof. *Journal of Systems and Software*, 81(4): 584–600, 2008. ISSN 0164-1212. doi: 10.1016/j.jss.2007.06.002. URL http://www.sciencedirect.com/science/article/B6V0N-4NXXM95-2/2/1aff678f4d0ecc5272c527aa521b97f3. Selected papers from the 10th Conference on Software Maintenance and Reengineering (CSMR 2006).

Marco Sinnema, Onno De Graaf, y Jan Bosch. Tool support for COVAMOF. En *International Workshop on Software Variability Management for Product Derivation. Towards Tool Support*, 2004.

Sebastiano Vigna. ERW: Entities and relationships on the web. En *Proceedings of The 11th International World Wide Web Conference*, 2002. URL http://www2002.org/CDROM/poster/166/.

Sebastiano Vigna. Automatic generation of content management systems from EER-based specifications. En *International Conference on Automated Software Engineering*, páginas 1527–1366. IEEE Computer Society, 2003. doi: 10.1109/ASE.2003.1240316.

Manuel Vázquez Acosta. Desarrollo de una arquitectura rectora para el programa de informatización de la prensa y su impacto en la gestión de proyectos. En *Forum de Ciencia y Técnica*. Universidad de las Ciencias Informáticas, 2009.

Manuel Vázquez Acosta. Impacto del patrón MVC en los requisitos de calidad percibidos. En *I Taller de Ciencias Informáticas aplicadas a la gestión empresarial. Memorias del evento UCIENCIA*, 2010a.

Manuel Vázquez Acosta. Desarrollo de una arquitectura de referencia para el programa de informatización de la prensa. *Serie Científica de la Universidad de las Ciencias Informáticas*, 3(4), 2010b.

Jim Webber, Savas Parastatidis, y Ian Robinson. *REST in Practice*. O'Reilly Media, Inc., 2010. ISBN 978-0-596-80582-1.

Andrzej Zalewski y Szymon Kijas. Architecture decision-making in support of complexity control. En M. Ali Babar y I. Gorton, editores, *Lectures in Computer Sciences (LNCS) 6285*, páginas 501–504. Springer-Verlag, Berlin Heidelberg, 2010.

Dave Zubrow y Gary Chastek. Measures for software production lines. Nota técnica CMU/SEI-2003-TN-031, Software Engineering Institute, Carnegie Mellon University, Octubre 2003.