UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

INFORME

INTERFAZ DE USUARIO PARA PROCESOS BPM CON INTALIO

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTA:

FILIBERTO RAMÍREZ BUSTAMANTE

DIRECTOR DE TRABAJO PROFESIONAL

M. C. ALEJANDRO VELÁZQUEZ MENA

CIUDAD UNIVERSITARIA 28 /Agosto/2014.

Índice

Introducción

El desarrollo de software actualmente tiene muchas vertientes, abarca una cantidad considerable de metodologías de desarrollo, un sin fin de enfoques, de lenguajes de programación, de patrones de diseño y de arquitecturas entre otras cosas. En este sentido, el hallar un conjunto de todos estos elementos y que convivan de manera armónica no es algo que deba tomarse a la ligera porque las decisiones se toman día a día impactan a lo largo de un proyecto.

En cuanto a lo que refiere a las metodologías la mayoría de ellas se centra en la combinación de los siguientes elementos:

- Investigación del mercado
- Obtención de los requerimientos para la solución propuesta
- Análisis del problema
- Planificación o diseño de la solución basada en software
- Implementación
- Prueba del software
- Despliegue, transición o implantación
- Mantenimiento y corrección de errores

Algunas empresas destinan más tiempo a ciertas actividades dependiendo de la experiencia adquirida y el enfoque que le dan a su negocio, de acuerdo al modelo que se lleva en la empresa estos tiempos son estimados y generalmente concuerdan con los tiempos reales.

¿Qué debería regir para un buen desarrollo de software?, es una pregunta que muchas metodologías de desarrollo se han hecho y han intentado resolver, así dependiendo de los adeptos que tengan, estas metodologías se van posicionando en el mercado. Las más populares son RUP (Rational Unified Process) y SCRUM.

Pero, ¿qué hay acerca de las otras metodologías?, ¿son malas?, ¿no sirven?, ¿no garantizan el éxito de un proyecto?, en realidad ninguna es mala o buena, todas son útiles, y el éxito de un proyecto no depende de la metodología que se use para llevarlo a cabo, a pesar de tener un impacto importante en este.

Lo más importante al llevar un desarrollo de software no es escoger las mejores herramientas de software, o la mejor metodología, es sentirse cómodo y confiado de que lo que se está eligiendo es adecuado para lo que se está haciendo, es decir, la mejor metodología es la que nos funciona así como las herramientas que usamos y conocemos.

No obstante si se cuenta con tiempo y si la solución actual no está funcionando, no está de más revisar nuevas tendencias, nuevas herramientas y nuevos paradigmas, de esta manera la solución construida estará al día dando la seguridad de que lo que se hace es de la manera en la que se quiere, proporciona la calidad y grado de satisfacción requerido.

A menudo las soluciones diseñadas cumplen explícitamente con los requisitos funcionales, visuales y de rendimiento, sin embargo lo que hace que esto funcione como el cliente esperaba es algo que pocas veces se toma a consideración o que se destina poco tiempo en diseñar, siendo así muchas veces la peor de las formas de realizar una solución a pesar de que cumpla con las expectativas del cliente.

Es aquí donde el rol de arquitecto de software enfatiza que la solución realmente sea una buena solución, una solución como muchos dicen "360", y como todo, ser un buen arquitecto se logra con el paso del tiempo, cometiendo errores y aprendiendo de ellos. Es por ello que para llegar a una propuesta de solución desde el diseño, construcción y desempeño, un factor importante es la investigación y el aprovechar los recursos materiales como humanos para lograr establecer una solución.

Finalmente, una solución no solamente es la que define un arquitecto de software, ni tampoco los patrones de diseño o la manera en que se concibe la integración de los componentes, sino el conjunto de individuos que lo hacen posible a través de sus ideas y que las expresan en forma verbal, escrita, en blogs, en foros, incluso en listas de correo y que muchas veces no reciben el crédito que se merecen.

El presente informe detalla la concepción de una aplicación web que sirve de interfaz entre el sistema BPMS Intalio y otros sistemas web, aprovechando infraestructura de base de datos y servicios web ya existentes.

La intención de crear un sistema de este tipo nace de la necesidad de mostrar el BPMS Intalio como una alternativa viable open source para grandes empresas, así como para el gobierno e instar al uso de tecnologías tanto open source como free source.

A través del informe se muestra el proceso de desarrollo a nivel personal de quien escribe este informe, como el desarrollo de la interfaz mencionada, desde su concepción, la necesidad de crearla, el alcance así como parte del análisis, diseño, construcción, pruebas e implantación. Asimismo la metodología empleada para llevar a cabo el desarrollo.

Capítulo 1. La organización

Una organización es un conjunto de elementos, compuesto principalmente por personas, que actúan e interactúan entre sí bajo una estructura pensada y diseñada para que los recursos humanos, financieros, físicos, de información y otros, de forma coordinada, ordenada y regulada por un conjunto de normas, logren determinados fines... (Thompson, 2007).

1.1 SaitoSoft

SaitoSoft es una empresa fundada en noviembre del 1996 con 15 años de experiencia en la Industria de las Tecnologías de Información en el segmento de desarrollo de Software y ha participado en proyectos para clientes del sector público y privado.

Un distintivo de SaitoSoft en la industria de las Tecnologías de Información es que en el año 2006 lanzo una iniciativa de mejora de sus procesos de desarrollo, permitiéndole controlar su operación y poder adaptarse al constante cambio tecnológico.

En 2008 SaitoSoft a través de la operación de procesos estructurados y ordenados consigue obtener una acreditación satisfactoria del modelo de madurez de capacidades para el desarrollo de software CMMI Nivel 2 (por sus siglas en inglés de Capability Maturity Model Integration) SaitoSoft evoluciona su oferta de servicios orientándola a la automatización de procesos de negocio para operar como una fábrica de procesos a partir del modelo de fábrica de Software con el que originalmente inicia. (Saitosoft, 2007)

Enfocado a dar soluciones integrales en el desarrollo de sistemas y consultoría en informática de acuerdo a las necesidades y requerimientos de cada cliente, tomando siempre en cuenta las posibilidades tecnológicas disponibles de cada uno de ellos (Saitosoft, 2007).

Debido al modelo CMMI2 el desarrollo de software y demás artefactos que se elaboran, deben de cumplir con un estándar, como procesos, dispositivos electrónicos, etc., lo que resulta en una buena administración de proyectos y que la documentación de los mismos sea de buena calidad de acuerdo a los estándares establecidos.

Como organización, la importancia que tiene que los miembros de ella desempeñen funciones diversas determina el éxito en la realización de un proyecto, es por ello que

SaitoSoft mediante las áreas que lo componen coordina de manera eficiente la integración de los individuos.

SaitoSoft está compuesta de varias áreas de tecnologías y otras tantas administrativas, en cuanto a lo que nos concierne, que son las áreas de tecnología, se encuentra un área de desarrollo de software, área de análisis, área de pruebas, área de procesos y un área de infraestructura y servicios.

Hoy en día SaitoSoft cuenta con clientes en distintos sectores, desde gobierno, educación, salud, hasta instituciones privadas de la industria farmacéutica, empresas de energía y construcción.

1.2 Área de análisis

El área de análisis es el área de campo, los analistas acuden frecuentemente con el cliente, a través de una serie de entrevistas acordadas, para recabar información que ayude al área de desarrollo a diseñar y construir la solución. Muy a menudo es necesario que el área de análisis reciba retroalimentación por parte del área de desarrollo para no comprometer el proyecto o partes del mismo mejorando la toma de decisiones, la integridad, y el alcance.

De acuerdo a estas actividades se determinan los módulos, procesos, y sistemas a construir o modificar (en caso de que sea un re trabajo), requerimientos funcionales, requerimientos no funcionales, reglas de negocio, entre otros. Es importante destacar que las competencias que tienen los analistas así como la realimentación que se tenga con el área de desarrollo influye en que el trabajo final de satisfacción al cliente, y permite evitar los retrasos en las fechas de entrega.

De acuerdo a la solución que se pretenda realizar, ya sea una solución a la medida, un ERP, un servicio de consultoría o la automatización de un proceso, el analista realiza documentos ad hoc, realizando casos de uso o documentos que especifiquen el flujo, interacciones, y demás de un proceso. Estos documentos son estandarizados y revisados por el área de calidad.

El análisis es un elemento fundamental para el desarrollo ya que proporciona los artefactos necesarios para la siguiente etapa del proyecto, que es el desarrollo.

1.3 Área de pruebas

El área de pruebas planifica, construye, ejecuta y recaba la información mediante mecanismos automatizados, programación u otras herramientas que faciliten esta labor. Dependiendo del modelo empleado, esta fase puede ir al final de proyecto o puede verse involucrada a través de todas las fases.

Esta área está Íntimamente relacionada con el área de desarrollo y con el área de análisis, ya que los analistas conocen los requerimientos más a fondo (pruebas de aceptación) y el área de desarrollo sabe lo que puede fallar, de esta manera se conciben las pruebas necesarias a realizar y el punto de vista de atender a los requerimientos es diferente del de un analista y del de un desarrollador, proporcionando una solución más robusta.

En esta área se realizan todo tipo de pruebas, de aceptación, pruebas de estrés, pruebas de concurrencia, pruebas unitarias, pruebas de sistema, pruebas de integración y se documentan.

1.4 Área de procesos

El área de procesos concierne a todo lo relacionado con soluciones BPM (Business Process Management), para esta labor SaitoSoft tiene como opción principal Intalio, además de ser socio nivel plata.

En el área de procesos existen roles equivalentes a los del área de desarrollo, hay un analista de procesos, arquitecto de procesos, ingeniero de procesos, estos son los encargados de llevar la automatización e integración de un proceso en BPMN (Business Process Management Notation) desde la concepción hasta la transición del mismo.

En ocasiones se necesitan desarrollar componentes de software que el BPMS (Business Process Management System) debe involucrar en el flujo del proceso, de esta manera existe una relación muy estrecha entre el área de procesos y el área de desarrollo.

Realizar un proceso en Intalio involucra emplear diversas tecnologías no solamente BPMN, se hace uso de xpath, xml schema, ws-*, sql y xslt. Siendo así muchos de estos elementos son construidos por el área de desarrollo no por el área de procesos, ya que los conocimientos necesarios no los posee el área de procesos.

1.5 Área de infraestructura

El área de infraestructura y servicios está encargada del despliegue de las soluciones, se instalan, configuran y se ponen a punto servidores, servicios y demás necesidades de la solución para que su funcionamiento sea óptimo.

Desde la concepción de la solución esta área debe estar involucrada ya que prepara el ambiente de pruebas como el de producción, provee las herramientas necesarias para el desarrollo, así como para el control de versiones, automatización de construcciones de software y comunicación entre las otras áreas de la empresa.

1.6 Área de desarrollo

Siendo SaitoSoft una empresa dedicada al software, esta área es el núcleo de un éxito en la realización de cualquier proyecto. En esta área se deposita todo lo recabado por los analistas, se conjuntan los servicios proporcionados por el área de infraestructura y se crean las soluciones, los estándares, y lo más importante, las ideas que fundamentan y hacen que la empresa destaque en el desarrollo, de no ser por esta área y los elementos que la componen, una compañía de software es difícil que se mantenga a flote.

El área de desarrollo está dispuesta en varias especialidades de acuerdo al lenguaje que sea necesario para la construcción de una solución, así se divide en .NET, Delphi y Java de manera general.

De acuerdo a la organización hay roles que se deben desempeñar en cada una de estas áreas de desarrollo, existe un líder técnico, un arquitecto de software, un ingeniero de software y un ingeniero de pruebas como mínimo para cada subdivisión del área de desarrollo de software.

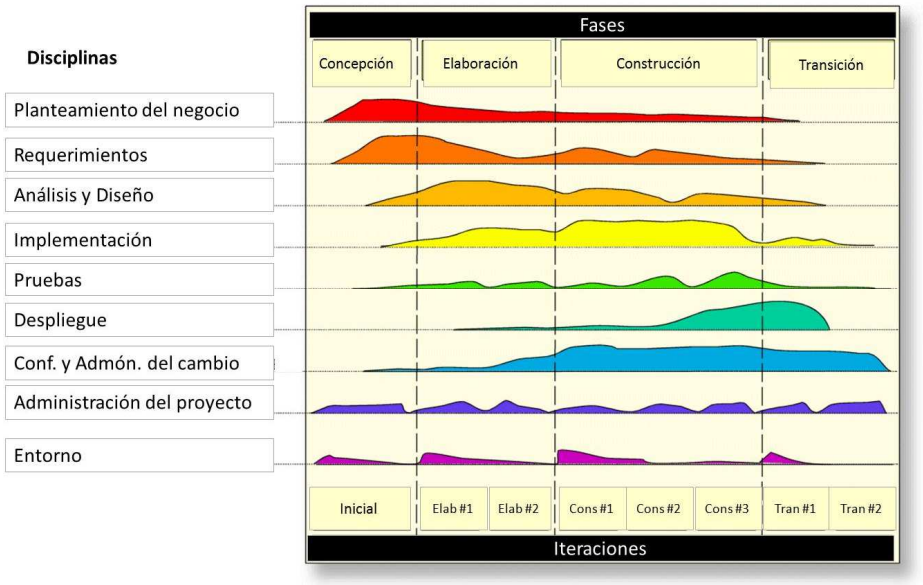
El área de desarrollo se centra en planear, diseñar, construir, integrar y probar la solución que se pretende implementar para algún cliente, o bien para generar servicios internos para facilitar el mismo desarrollo de software o alguna otra área de la empresa, como calidad.

Para realizar esto, los encargados del análisis interactúan con el arquitecto de software el cual diseña la solución y los ingenieros de software construyen. Ocasionalmente el

arquitecto de software junto con el líder técnico funge como guías y proveen ayuda a los ingenieros de software en caso de que exista algún problema con la implementación.

De manera general, para la mayoría de desarrollos se utiliza la metodología RUP, sin embargo a partir de mi incorporación como arquitecto de software, y debido a que la empresa es pequeña se incorporó la metodología de desarrollo XP.

En la Fig. 1.1 se esquematiza RUP para mostrar el proceso de desarrollo, esta metodología es bien conocida, y consta de 4 fases, concepción, elaboración, construcción y transición.



Metodología RUP Fig. 1.1

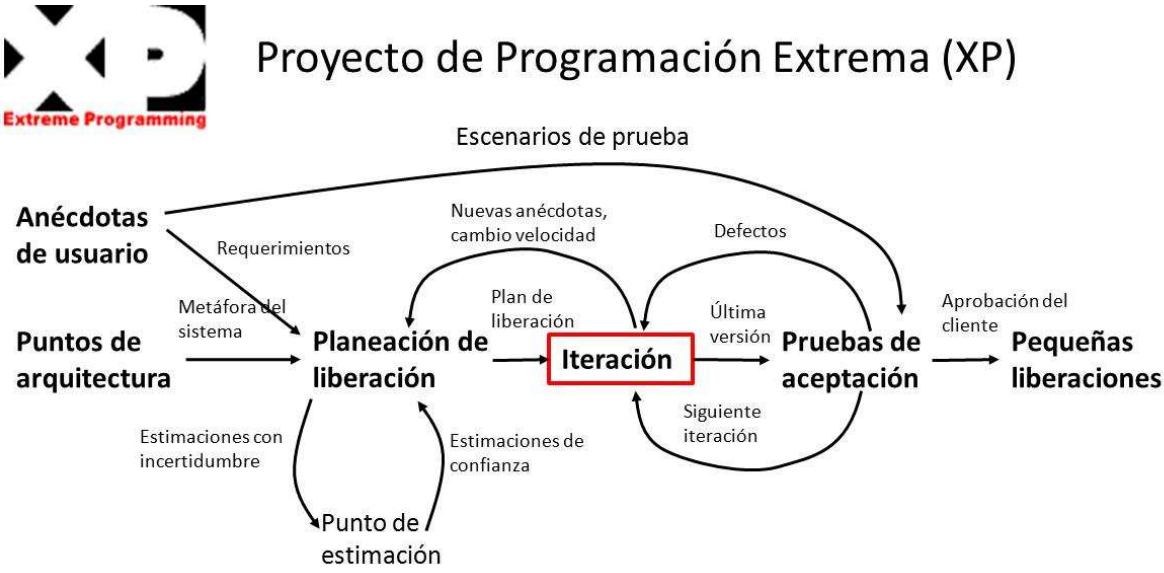
De acuerdo a la fase distintos roles están involucrados, en la fase de concepción el líder técnico, el arquitecto de software y el analista participan mayormente ya que en esta fase se plantea el qué y el cómo de la solución, en esta fase es necesario generar un calendario para estimar el esfuerzo necesario que se requerirá de cada elemento que participará en el proyecto y la cantidad de elementos necesarios para terminar en tiempo y forma.

En las otras fases se involucra mayormente el arquitecto de software, el líder técnico y los ingenieros de software ya que se trabaja en la elaboración y construcción. Para ello es indispensable contar con estándares de todo tipo, codificación, de documentación, de

diseño, más adelante en este documento se comenta acerca de CMMi3 y la colaboración que tuve.

Finalmente, para la fase de transición el área de desarrollo no es involucrada salvo por problemas de último momento que no hayan sido concebidos en la etapa de pruebas y ajustes generales como puesta a punto de los servidores, configuración del aplicativo y gestión de seguridad.

Hoy en día los proyectos activos son principalmente BPM y los aplicativos son desarrollados en la plataforma java. El proceso de trabajo utilizado por XP es mostrado en la Fig. 1.2. Los clientes disfrutan de ser parte del proceso de desarrollo, los desarrolladores contribuyen activamente independientemente del nivel de experiencia, y los líderes de proyecto se pueden concentrar en comunicación y relaciones interpersonales. Las actividades no productivas han sido removidas para reducir costos y frustración de cada uno de los involucrados.



Proyecto de programación extrema Fig. 1.2

La programación extrema (XP) es uno de los procesos agiles más populares. Ha sido probado múltiples ocasiones por muchas compañías de todos tamaños a lo largo del mundo y es una de las metodologías agiles más empleadas actualmente.

1.7 Líder técnico

El líder técnico es el rol que estuve desempeñando en SaitoSoft antes de mi salida. Para poder desempeñarlo es necesario tener vasto conocimiento horizontal, ya que entre las actividades que se desempeñan se encuentra el auxiliar, coordinar, y delegar actividades a los miembros del área de desarrollo y procesos.

Como líder técnico no solamente auxilié en cuanto a dudas sobre algún tema en particular de tecnología, apoyé en el desarrollo de soluciones y detecté posibles fallas en el diseño de las mismas. En ciertas circunstancias tuve que participar en el diseño, desarrollo e implementación de la solución.

Fue importante revisar cómo se llevaba a cabo el proyecto a través de todas sus fases, al momento de diseñar una solución evaluar el sistema operativo y las restricciones o beneficios que tiene este, los manejadores de la base de datos, y el uso de herramientas para el desarrollo de las soluciones así como el seguimiento de estándares de desarrollo.

Dentro de las actividades más relevantes que realicé como líder técnico se encuentran:

1. Realizar el análisis técnico de las necesidades del cliente
2. Diseñar la solución de software
3. Desarrollar los requerimientos técnicos
4. Diseñar y administrar la base de datos
5. Diseñar y administrar los ambientes para el plan de pruebas
6. Administración de la configuración y control de versiones, infraestructura y ambiente
7. Administrar los recursos de hardware y software
8. Realizar la integración entre sistemas

Estas actividades constituyeron el quehacer del día a día mientras mantuve ese rol, de esta manera aprendí entre otras cosas a tener una mayor abstracción del entorno de desarrollo tanto a nivel de software como en el interactuar con las personas en los distintos niveles.

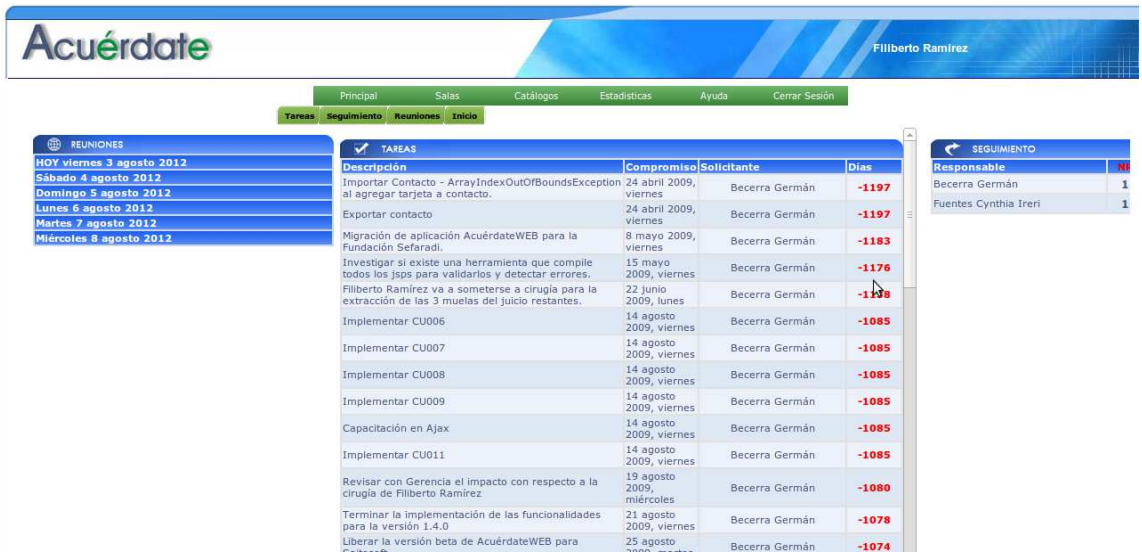
Capítulo 2. Participación en proyectos

Mi participación en proyectos ha sido heterogénea debido a los roles que he desempeñado. La colaboración con diferentes personas y diferentes soluciones han propiciado que mi desarrollo profesional así como el desempeño de las responsabilidades a las cuales estoy asignado, mejoren a través del tiempo, al mismo tiempo que mis habilidades se perfeccionen o adquiera nuevas.

2.1 AcuérdateWeb

AcuérdateWeb es una aplicación web implementada con la plataforma J2EE empleando el patrón MVC, consiste en gestionar actividades, asignaciones, reuniones, y administración de lugares de reunión. Esta gestión va desde crear una tarea, delegarla y darle seguimiento, crear una reunión, invitar asistentes, y convocarla vía correo, seleccionar un lugar para tener una reunión, establecer hora de encuentro, y duración, así como los asistentes, por lo cual la gestión de lugares de reunión puede estar relacionada con la gestión de reuniones, aunque no es un requisito.

La figura 2.1 muestra la pantalla principal del sistema, como parte de la información que proporciona, se encuentran las reuniones próximas, las tareas pendientes y las personas con las que se tienen actividades pendientes.



Acuérdate Fig. 2.1

Al ingresar a SaitoSoft como primer actividad apoyé en la mejora de la arquitectura de este y otro proyecto. Entre las actividades que realicé, destaca la prueba de concepto de multitud de frameworks, librerías javascript y entendimiento de metodologías de desarrollo, construí muchos demos de estas tecnologías las que mostraba a su vez al arquitecto que estaba a cargo de mi para que el decidiera las herramientas utilizar. Realizar este tipo de actividades permitió aprender y conocer tecnologías.

La forma de trabajar era desgastante, se trabajaba con un IDE (Eclipse) sin embargo solo se usaba para poder escribir el código de forma ágil. La forma de trabajar era la siguiente:

1) Buscar el archivo en el cual se trabajara, como un archivo .java, .jsp u otro.
2) Abrirlo con ayuda de Eclipse
3) Editar el archivo
4) Compilarlo con eclipse
5) Copiar los archivos compilados a bytecode al contenedor de aplicaciones
6) Reiniciar el contenedor de aplicaciones

Por este motivo corregir algún problema o desarrollar una nueva funcionalidad tomaba largos periodos de tiempo, lo primero que realice fue la migración de la aplicación a un proyecto web que pudiera ser compatible con eclipse y netbeans para poder tener el control completo de la aplicación desde el IDE, agregar un contenedor de servlets (tomcat) a eclipse para poder desplegar la aplicación desde el IDE al contenedor, con estas modificaciones se mejoró considerablemente el tiempo de desarrollo.

El proyecto necesitaba urgentemente una reingeniería dado que tenía muchas malas prácticas. Idealmente las aplicaciones web que son creadas con java se rigen bajo la especificación J2EE. Para el momento en que intervine en este proyecto, la especificación J2EE estaba en la versión 1.4.

Entre las ventajas que proporciona el tener una aplicación basada en la especificación J2EE son:

1) Una arquitectura simple basada en componentes bien definidos permitiendo así la portabilidad.
2) Interacción con servicios existentes, como manejadores de bases de datos, colas de mensajes, e-mail, entre otros.

3) Escalabilidad, permitiendo distribuir la aplicación a través de múltiples contenedores de aplicaciones, así como balanceo de carga.
4) Un modelo de seguridad flexible para el soporte de sesiones y cookies, integración con sistemas de seguridad existentes.

Entre los componentes de los que consta esta especificación se encuentran los EJB Enterprise java beans (ejecutados por un contenedor que soporte EJB), Servlets y JSP Java Server Pages (ejecutados por un contenedor web).

Entre otras tantas características de esta especificación lo descrito previamente permite dar el entendimiento a las actividades realizadas.

La aplicación no contaba con estos requisitos, era un intento mal logrado. Entre los principales problemas estaba el excesivo uso de JSPs para escribir código java, estaban llenos de líneas de código y scriptlets, esto cambió y solo se dejaron para hacer el despliegue de la página web y no cargar transacciones ni procesamiento de datos, esto último se dejó a cargo a controladores en forma de servlets.

Se implementó un ORM (Object Relational Mapping) para realizar las transacciones en la base de datos, todo estaba basado en servlets y jsp's (Java Server Pages), el ORM utilizado fue Ibatis, que actualmente es un proyecto de Apache Software Fundation y ha cambiado su nombre a MyIbatis. El uso de un ORM no desarrollado en casa (que era el que se tenía originalmente) mejoró el rendimiento, ya que se podían configurar pools de conexiones, administrar mucho mejor las consultas SQL, la gestión de las entidades de negocio entre otras tantas bondades.

Mayormente se mejoró la implementación de fondo y muy poco del front-end. Mucho de la construcción realizada en jsps como persistencia, subir o bajar archivos, lógica del negocio fue implementado en servlets con apoyo de patrones de diseño que no obstruyeran la visión del arquitecto ya que fungía como ingeniero de software. En resumen se pasó la aplicación a ser MVC (Model View Controller), introduciendo JSTL (Java Standar Tag Library) y un framework de javascript (Ext-JS y prototype).

Después de retocar el proyecto, pudimos desarrollar nueva funcionalidad la cual consistió en llevar a cabo un módulo de gestión de salas para reunión, en el cual se tiene un calendario, se selecciona un día y posteriormente se llena información de la reunión, como horario, integrantes y asuntos a discutir, se pueden dar de alta nuevas salas, así como la

capacidad de asistentes, en el calendario se visualiza los días disponibles así como los horarios.

Entre otras mejoras que se realizaron al proyecto se encuentra el soporte del estándar iCalendar para añadir las reuniones automáticamente al cliente de correo, por ejemplo Microsoft Outlook o Thunderbird, tareas periódicas, gestión de salas, mejoras en el rendimiento de la interfaz realizando llamadas Ajax y eliminando en la medida de lo posible ventanas emergentes con un framework llamado DWR.

Tiempo después comencé con el desarrollo de una nueva versión de AcuérdateWeb que incorporaba soporte de temas, múltiples idiomas, múltiples motores de bases de datos, anteriormente solo era Microsoft SQLServer 2000, así como el soporte multiplataforma, inicialmente solo era soportado Windows.

Esta aplicación actualmente es usada por más de diez empresas, aunque ya no está en desarrollo, solamente se realiza mantenimiento y correcciones de bugs.

Cabe mencionar que aún hay mucho trabajo por realizar en esta aplicación ya que como fue concebida ya no encaja en las aplicaciones de hoy en día, que son sencillas, no muy vistosas, amigables en cuanto a la interfaz y usuario, muy robustas, además de que se pueden conectar con múltiples sistemas, soportar distintos tipos y niveles de seguridad, sin mencionar la plataforma móvil tiene un gran auge en nuestros días.

2.2 CumpléWeb

CumpléWeb es un sistema que permite llevar el control de actividades que son generadas manual o automáticamente de acuerdo a un marco jurídico para así cumplir con los reglamentos que marca una ley o una norma. Algo muy similar a AcuérdateWeb, solo que es para llevar un control rígido de actividades que puede rescatarnos de un problema legal.

Las modificaciones hechas a AcuérdateWeb respecto a frameworks y demás fueron hechas también para CumpléWeb. Adicionalmente cree un módulo extra para control documental usando un framework (Lius basado en Lucene) para realizar búsquedas por contenido o bien por existencia.

Para llevar a cabo el módulo de control documental fue necesario determinar en qué puntos se realizaba ingesta de archivos hacia el servidor, que tipos de archivo y el control de quien podía acceder a ellos. De esta manera cada vez que se subía un archivo a través de la aplicación se indexaba y se creaba un registro en base de datos con la información de los metadatos del archivo indexado así como el autor del archivo.

Lius emplea entre otras cosas Apache POI que permite realizar ciertas operaciones sobre archivos de Microsoft Office, como búsquedas en el contenido reemplazo entre otras cosas.

Entre otras cosas el módulo de control documental realiza búsquedas e incorpora extractos del contenido donde encontró coincidencias, y resaltado con respecto al texto buscado.

La implementación de este control documental, me ayudo a mejorar mi entendimiento de casos de uso, y el desarrollo desde cero de un módulo de una aplicación, la aplicación de las habilidades adquiridas, y la capacidad de resolver problemas.

En el desarrollo de CumpléWeb para el front-end se empleó ExtJS que es un framework para construir aplicaciones web enriquecidas, en otras palabras aplicaciones web visualmente agradables y de fácil interacción con el usuario. Para construir con ExtJS se emplean widgets y se vinculan componentes java para una interacción simple; la mayoría de las llamadas realizadas al servidor funcionan mediante ajax, esto permite que la página web no sea actualizada del todo y el usuario sienta que el sistema responde ágilmente.

Previo a la selección de ExtJS se realizaron pruebas de concepto con GWT, Prototype y Script.aculo.us, ninguno de ellos permitió la facilidad de construcción y de interacción como ExtJS.

2.3 CONACULTA

El proyecto con CONACULTA fue la automatización de procesos en notación BPMN con Intalio como herramienta. Inicialmente me incorporé como ingeniero de software para apoyar con el desarrollo, desafortunadamente la arquitecto no estuvo en condiciones de llevar a cabo el proyecto, así que termine siendo promovido a arquitecto de software y diseñando un portal que permitiera interactuar con Intalio de manera amigable.

Siendo mi primer acercamiento al diseño de arquitectura para una aplicación, tuve que leer y realizar una gran cantidad de pruebas de concepto para entender el funcionamiento de frameworks y librerías determinando la comunicación entre ellas y el propósito de acuerdo a la capa en la cual se incorpora.

Intalio, no era una herramienta conocida, y con el poco tiempo que se tenía para desarrollar o aprender se optó por simular el comportamiento de un BPM a través de una maquia de estados, de esta manera se diseñó el primer portal con el que se interactuaba para tener tareas, notificaciones y procesos a desplegar. En un principio esta aplicación tenía el propósito de mostrar una interfaz amigable que interactuara con el BPM, sin embargo no se pudo lograr de inmediato por la falta de madurez y de conocimientos del equipo, no obstante fungió como base para un desarrollo de una interfaz más robusta, PUI.

El desarrollo estuvo basado en MVC, usé spring para manejar la inyección de dependencias e hibernate como ORM, el modelo de spring para manejar el MVC para simplificar las tareas de control, procesamiento de formularios y páginas web.

Se implementó el control de acceso mediante spring security autenticando contra una base de datos que se diseñó con roles y permisos, esto fue de lo más sencillo ya que solo basto crear la estructura ad-hoc y configurar un archivo. Entre otras cosas se implementaron reportes utilizando el api de iText.

Se consumieron algunos servicios web de Intalio para realizar la interacción con él. Esto derivo en una profunda investigación, ya que no se conocía en lo absoluto de servicios web, desde que son, como usarlos, como consumirlos, como crearlos.

Este acercamiento a Intalio sirvió como base para el proyecto que realizaría posteriormente y dado que fue realizado con mucha premura no hubo tiempo de diseño lo cual resulto en una implementación poco elegante sin el uso de ningún patrón de diseño y con anti patrones como spaghetti code.

2.4 Sin asignación de proyecto

En el tiempo que estuve desasignado SaitoSoft se estaba pensando en pasar a un nivel superior de CMMi, pasar del nivel 2 al 3, en esta transición (que no fue concretada) se

involucraba mucho el desarrollo y los estándares a nivel técnico, el qué, el cómo, y el con qué. Por ello tuve que ahondar en todo lo relacionado con estas actividades.

Debido a la investigación realizada, como resultado se pusieron en marcha herramientas de integración continua para agilizar el desarrollo sobre máquinas virtuales en un servidor con citrix. Se investigaron otras alternativas para usar como metodología de desarrollo, de ahí el uso de XP.

2.4.1 Hudson

Hudson es un sistema de integración continua que monitorea la ejecución de tareas que son repetitivas, como construir un proyecto de software o trabajos que se rigen por un cron. Entre estas y otras cosas, actualmente Hudson se enfoca en los siguientes puntos:

1. Construir/probar proyectos de software de manera continua, justo como CruiseControl o DamageControl. En una palabra, Hudson provee un sistema de integración continua fácil de usar, de fácil integración a los cambios en un proyecto, y de fácil acceso para un usuario al obtener la construcción del software. Esto incrementa la productividad.

2. Monitorear ejecuciones de trabajaos externos, justo como las tareas de un cron y procmail, incluso aquellos que son ejecutados en una maquina remota.

2.4.2 Archiva

Apache Archiva™: El gestor de repositorios y artefactos, es un software de administración de repositorios extensible que ayuda a proteger los artefactos construidos tanto personales como empresariales. Es un perfecto acompañante para herramientas de construcción como maven, continuum y ant. (Apache Software Foundation, 2013)

Archiva cuenta bastantes facilidades, como configurar repositorios remotos a través de proxies, administración de acceso seguro, almacenamiento de los artefactos, entrega, búsqueda, e indexado de los mismos, entre otras cosas.

2.4.3 Sonar

Ahora llamado Sonarqube, es una plataforma abierta para administrar la calidad del código, Sonarqube cubre 7 ejes de la calidad del código:

1) Comentarios
2) Reglas del código

3) Arquitectura y diseño
4) Bugs potenciales
5) Complejidad
6) Duplicación
7) Pruebas unitarias

Sonarqube se encarga de analizar el código generado de forma continua y crea métricas que permiten establecer la calidad del código desde distintos puntos de vista, como puede ser a nivel del proyecto hasta un método. (Sonarqube)

2.4.4 Checkstyle

Checkstyle es una herramienta de desarrollo para ayudar a los programadores a escribir código Java de acuerdo a un estándar de codificación, mediante esta herramienta se automatiza el proceso de revisar el código Java por otros programadores. (Sourceforge, 2013)

Esta herramienta es altamente configurable y adaptable a casi cualquier entorno de desarrollo existente.

2.4.6 Trac

Trac es un sistema de wiki y de seguimiento de issues para proyectos de desarrollo de software. Provee una interfaz de integración con subversion o git. (Trac, 2013)

Entre otras bondades, Trac permite vincular comentarios en issues hacia artículos o fragmentos en la wiki, crear ligas y referencias entre bugs, tareas, listas de cambios, archivos y páginas de wiki; Cuenta con una línea de tiempo para poder llevar a cabo el rastreo de los problemas a través de la evolución del proyecto.

Capítulo 3. Interfaz de Usuarios para Procesos (PUI)

3.1 Objetivo

Diseñar y construir una interfaz web capaz de procesar todas las acciones que involucra un BPMS como Intalio, permitiendo la integración hibrida con funcionalidades ya establecidas, de una manera simple, usable, rápida, segura, intuitiva y robusta.

3.2 Antecedentes

3.2.1 Business Process Management System (BPMS)

Un BPMS constituye un sistema capaz de llevar interpretar notación BPMN y ejecutar con ello un proceso de negocio que tiene por objetivo mejorar la eficiencia y la ejecución sistemática de las actividades dentro de una organización. Estos procesos de negocio deben ser diseñados, modelados, automatizados y optimizados de forma continua.

Mediante el modelado de las actividades se va conformando un flujo de trabajo el cual se denomina proceso y mediante este se puede tener una comprensión de cómo funciona el negocio, generalmente al realizar dicho modelo se observan las áreas de oportunidad para mejorar el proceso, las actividades y por ende la organización.

El automatizar procesos reduce gradualmente el número de errores que existen en el negocio, ya que se identifican problemas en las actividades, con ello se mejoran y depuran para asegurar que se ejecuten eficientemente, se detectan cuellos de botella, actividades sin relevancia o que se pueden sustituir por un sistema, de esta forma la información obtenida a partir de ellos pueda ser usada posteriormente para su mejora así como para realizar estadísticas del proceso.

Para poder realizar esta administración o gestión es necesario contar con un conjunto de herramientas que proporcionen la facilidad de maleabilidad del modelado, cambios en la automatización, visualización del proceso y sus actividades, interacción con él, así como el cumplimiento del ciclo de vida del mismo. Estas herramientas comúnmente son llamadas BPMS y son poderosas soluciones empleadas para construir aplicaciones tipo BPM, una de ellas es Intalio.

3.2.2 Intalio

Intalio se describe hoy en día como "El líder BPMS de código abierto". Es un sistema de administración de procesos de negocio (BPMS), el cual cuenta con una aplicación para diseñar procesos, la cual está basada en Eclipse y emplea notación BPMN (Business Process Management Notation) en la versión 2.0 para modelar los procesos y sus actividades. Como BPMS, Intalio es capaz de administrar, orquestar, y ejecutar las tareas de un proceso.

La arquitectura del BPMS Intalio está basada en SOA (Service Oriented Architecture), de esta manera la integración con sistemas es generalmente mediante servicios web. El motor encargado de ejecutar el código BPEL es Apache ODE (Orchestration Director Engine), ODE se encarga de comunicarse con los servicios web enviando y recibiendo mensajes, manipulando datos y errores tal y como se describa en la definición del proceso.

Adicionalmente cuenta con un API de servicios denominado Tempo que tiene la función de soportar procesos, e interacciones con usuarios.

BPMN 2.0 además de representar el proceso de negocio, también es usado para su ejecución, para ello Intalio está construido de cierta forma que no es necesario realizar una traducción de BPMN a BPEL ya que Intalio es capaz de ejecutar el código BPMN de forma nativa.

En la Fig.3.1 se muestra la arquitectura actual de Intalio en la que esta implementado Intalio, esta arquitectura es bastante parecida a la arquitectura original, solo difiere en la versión de BPMN, la primera que se implemento fue la versión 1.2.

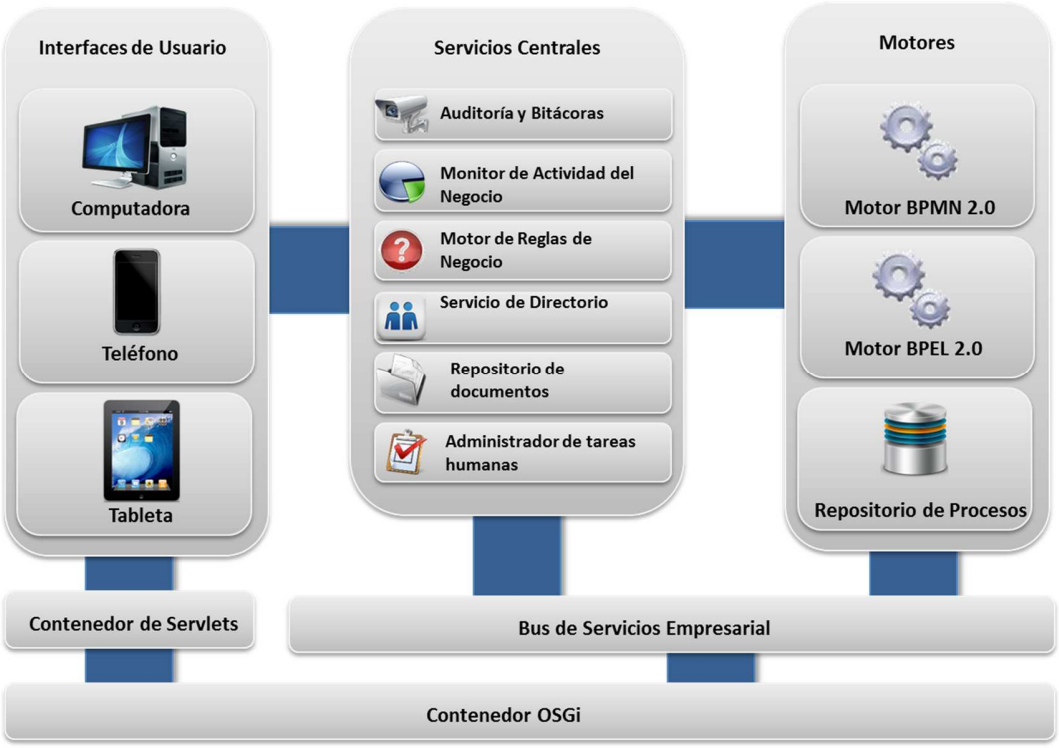
Esencialmente al idealizar un proceso se cuenta con las herramientas mencionadas enmarcadas como servicios centrales, todos ellos nos ayudan a crear los elementos que conformaran el proceso, la mayoría de ellas son opcionales y bien pueden ser implementadas por separado o mediante otras tecnologías.

En el apartado de motores, están los ya mencionados, uno que soporta BPEL, BPMN y un repositorio de base de datos que sirve para el motor completo de Intalio Server.

Los clientes pueden interactuar con el proceso a través de la interfaz de usuario denominada ui-fw, en ella se muestran los procesos que existen y a los cuales se tienen

permiso de acceder, ya sea para monitorear o para ejecutar, también se muestran las tareas generadas por el proceso y por último las notificaciones.

De manera ideal debería haber un ESB pero puede no haberlo, esta característica también es opcional, cabe mencionar que un ESB mejora la estabilidad y la escalabilidad de un proceso, ya que maneja conectividad con colas de mensajes como medio de interacción entre servicios y un conjunto bastante amplio de patrones para simplificar las tareas de comunicación.



Arquitectura Conceptual Intalio Fig. 3.1

Una de las mejores formas de interactuar con Intalio es mediante servicios web, sin embargo se proporcionan formas web para interactuar con un usuario, estas son xforms y ajax forms, xforms es un estándar de la W3C (World Wide Web Consortium) para crear interfaces de usuario basadas en XML, las Ajax forms pretenden ser interfaces de usuario enriquecidas basándose en el concepto del web 2.0 y utilizan una framework llamado General Interface proporcionado por TIBCO – JSX3.

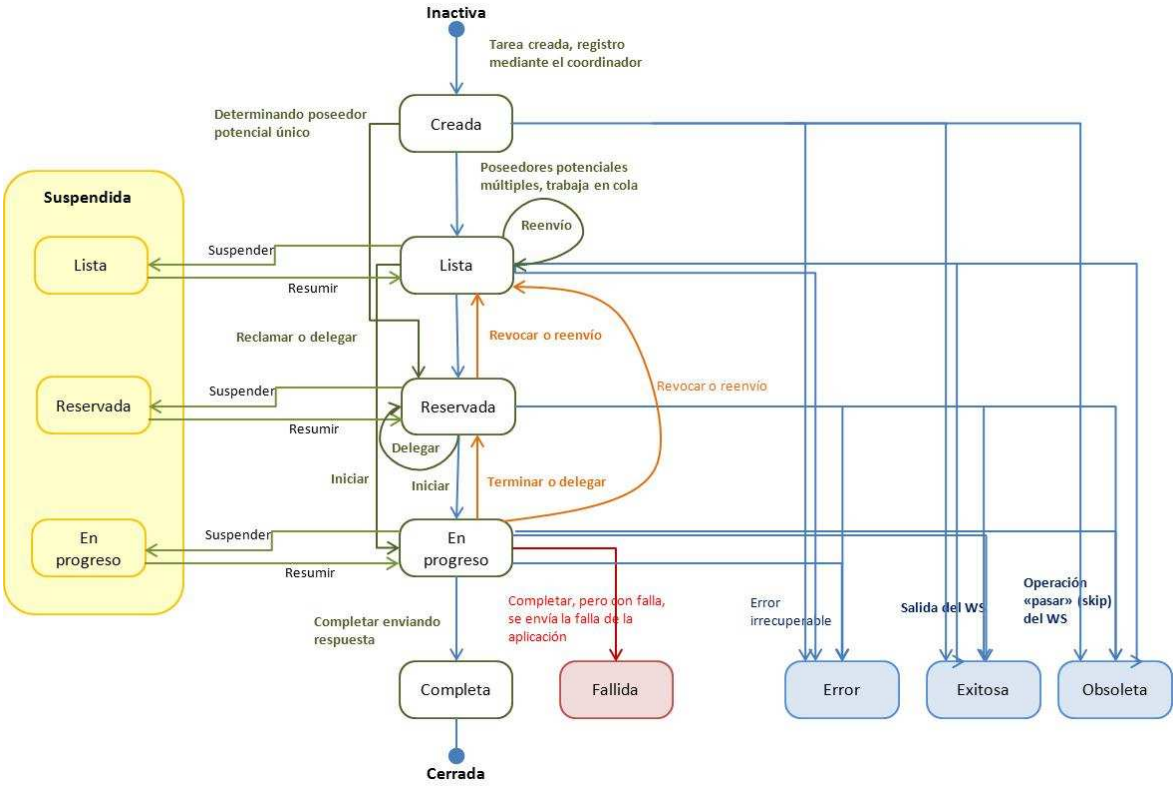
De esta forma, para soportar los complejos flujos de los procesos, Intalio incluye un API de servicios, uno de ellos y el principal es denominado Tempo creado con servicios web de tipo SOAP y asociado a tareas humanas. Estos servicios implementan el ciclo de una

tarea que tiene interacción con un humano, y puede ser fácilmente modificado para soportar fases personalizadas y transiciones, mientras se toma control de reglas de negocio mediante BRE (Business Rules Engine) que es un servicio web para evaluar reglas de tipo escalar.

El diagrama que se muestra en la Fig. 3.2 representa el flujo de una tarea, es importante conocerlo ya que muchas de las funcionalidades que contempla el desarrollo realizado, que se describe en este trabajo, están estrechamente relacionados con estados particulares que se muestran en la figura.

Los estados más importantes para el desarrollo que realice es cuando una tarea es creada y está lista para llevarse a cabo, cuando ha sido reservada y cuando ha sido completada, una tarea es creada cada vez que una tarea previa ha sido realizada, de esta manera las tareas no existen si no ha existido una tarea previa o un inicio de proceso.

Una tarea es reservada de forma automática (mediante modelado del proceso) o manualmente. Que una tarea se encuentre reservada significa que nadie más excepto el que tiene la reservación pueda completar esta tarea, esto involucra problemas importantes de concurrencia cuando no se hace de manera reservada el control de las tareas.



Flujo de una Tarea Fig. 3.2

Cuando una tarea es completada suceden cosas importantes, previamente se mencionó que se crean las tareas consiguientes, además de esto se realizan cambios en la base de datos, la tarea que ha sido completada debe desaparecer de la vista pero no de la base de datos. Es importante entonces considerar el cuándo deberían desaparecer estas tareas y ello involucra una planeación de archivamiento de tareas o mantenimiento de la base de datos y en ocasiones resulta complicado hacer ver que estas tareas no deben ser de información vital para una explotación de información del proceso por parte del cliente .En este desarrollo no contemple otros estados, sin embargo resulta interesante la creación de tareas. Ya que actualmente la manera en que se realiza esto en SaitoSoft es mediante el modelado de procesos, pero nada impide que se creen tareas en demanda sin más que algunos requisitos mínimos, como conocimiento de la instancia del proceso y de los XML Schemas existentes en él.

3.2.3 Procesos, Actividades y Tareas

Un proceso es una representación de cómo interactúan diversos actores para llevar a cabo un flujo del negocio, éstos interactúan ejecutando procesos, atendiendo o generando actividades y tareas.

Una actividad es un quehacer, una función a llevar a cabo y no necesariamente realizable por un humano, es aquí donde intervienen otros sistemas, otras interfaces u otros procesos.

A diferencia, una tarea siempre será algo con lo que el usuario deberá interactuar para alimentar al proceso, ya sea mediante una solicitud llenando un formulario de datos, proporcionar información para realizar un cálculo, introducir valores para el BRE, etc.

Estas tareas con las que tiene que interactuar un humano, necesitan ser presentadas de alguna manera, para ello Intalio provee de una interfaz en la cual se categorizan de la siguiente manera:

- Tareas que inician un proceso
- Tareas que notifican de algún suceso en el flujo de un proceso
- Tareas en las que se tiene que intervenir en alguna actividad del proceso

A pesar de todo lo que nos proporciona Intalio, es poco flexible en cuanto a modificaciones de la interfaz que proporciona al usuario final, esta interfaz de usuario final tiene muchas restricciones ya que no es de código abierto, no es configurable, no es adaptable a las necesidades del cliente, y los filtros que proporciona para las diferentes tareas a realizar son poco eficientes. Es por ello que surge la necesidad de realizar una interfaz general para el usuario donde pueda tener organizadas las actividades que tiene que llevar a cabo con la cual se sienta cómodo, además de que se adaptable, configurable ante nuevos cambios y que sea fácil de integrar con Intalio.

Cada actividad, proceso y tarea se encuentra asignado a un usuario o rol por reino (realm), toda esta configuración se realiza en un árbol LDAP, no obstante Intalio cuenta con otros mecanismos de configuración entre los cuales está el basarse en un archivo XML, o bien es posible implementar el propio extendiendo una interfaz.

3.2.4 Design Driven Development (D3)

3.2.4.1 Manifiesto y Filosofía

Cada sistema existente el día de hoy está construido en algunos prerrequisitos, estos funcionan correctamente solo si estos prerrequisitos son conocidos, de la misma manera, D3 funciona solo para las organizaciones y proyectos que creen en el desarrollo ágil y lo que este involucra, innovación y diseño como el diferenciador estratégico. (D3 Foundation, 2007)

Los siguientes son los principios clave en los que se basa D3:

- Diseño pleno y total realización del sueño son el nuevo mantra para un negocio exitoso.
- El diseño es un accidente que se inicia en el momento de la concepción, maximizando las oportunidades de hacer que los accidentes sucedan es la llave de la innovación.
- Ningún proceso puede garantizar un mejor diseño. Crear un entorno adecuado con un conjunto de gente adecuada es la única forma de crear innovación.

- Los más poderosos diseños son siempre el resultado de un proceso continuo de simplificación y refinamiento.

- Clientes y usuarios a menudo necesitan ayuda para entender, verbalizar, visualizar y organizar sus requerimientos.

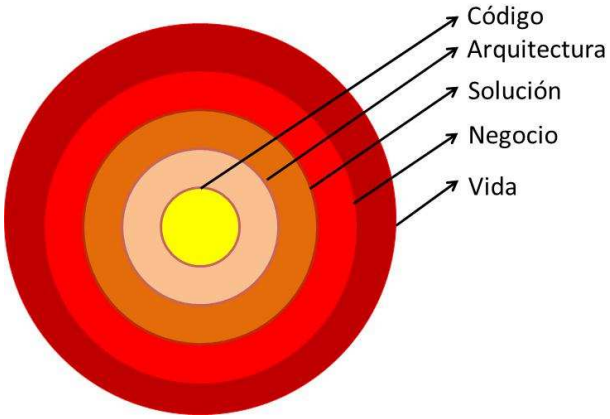
D3 está diseñado para para brindar agilidad al diseño de una solución e innovación al mismo tiempo, al lado del desarrollo de software.

3.2.4.2 Definición de Diseñar la Solución

Diseño es el arte de crear hermosas, elegantes e innovadoras soluciones las cuales funcionan en el contexto del cliente y el usuario. Alan Cooper tiene una clara definición para el diseño, "Todo es acerca del diseño, planeación de los artefactos, la forma, el contenido, comportamiento usable, inusable, deseable, así como económicamente viable y técnicamente realizable". (D3 Foundation, 2007)

Un buen diseño debería resultar en un sistema, que elimina tareas existentes (donde quiera que sea posible), acelerando las tareas eficientemente, ahorrando dinero, tiempo, y haciendo a las personas enamorarse.

D3 hace un llamado para un cambio de perspectiva y comprensión un paso adelante hacia el diseño técnico y el código; centralizándose en cómo podemos diseñar una mejor experiencia de vida para los usuarios.



Capas de abstracción en diseño Fig. 3.3

Un buen diseño consiste en cuatro elementos fundamentales, innovación, interacción, información e inteligencia.

- La innovación es el más amplio nivel de avance en la solución del problema
- La interacción es acerca de cómo el software/producto se comporta con los usuarios
- La información es la forma de organizar los distintos elementos en la pantalla
- La inteligencia se centra en las pequeñas cosas que pueden cambiar la usabilidad de una aplicación.

3.3 Definición del problema

El UI-FW es la interfaz que proporciona Intalio para acceder a las formas xforms y Ajax-forms, proporciona una vista para acceder a las tareas alusivas a iniciar un proceso, acceder a las notificaciones y finalmente acceder a las tareas que implican una interacción con el proceso. Esta interfaz es la que se sustituirá.

3.3.1 Problemas del UI-FW:

- No es posible filtrar tareas de acuerdo a un rol cuando se tienen múltiples roles.
- No era posible visualizar más columnas que las que se muestran por defecto para cada tarea.
- No es posible agregar más vistas o funcionalidades adicionales garantizando mantenimiento
- No hay manera de mostrar formularios con otras tecnologías que no sean Ajax Forms o XForms.
- La personalización o agregación de funcionalidades sobre las tecnologías Ajax Forms o XForms requieren de mayor tiempo, inclusive puede ser que no se puedan realizar.

La interfaz de usuario para procesos (PUI) resolvió estos problemas, entre otros, como de desempeño, modularización, estandarización y conectividad.

3.3.2 Conexión con otros problemas

Comenzando por la autenticación, realice una implementación de un AuthenticationProvider para poder utilizar cualquiera de las autenticaciones provistas por Intalio mediante spring. Intalio posee una autenticación basada en un servicio web denominado RBACQueryService, este servicio tiene dos implementaciones una basada en un archivo XML y otra basada en LDAP, gracias a este AuthenticationProvider no importa la implementación que se tenga en Intalio ya que se hace mediante el servicio mencionado pero adaptado a esta interfaz que utiliza spring-security.

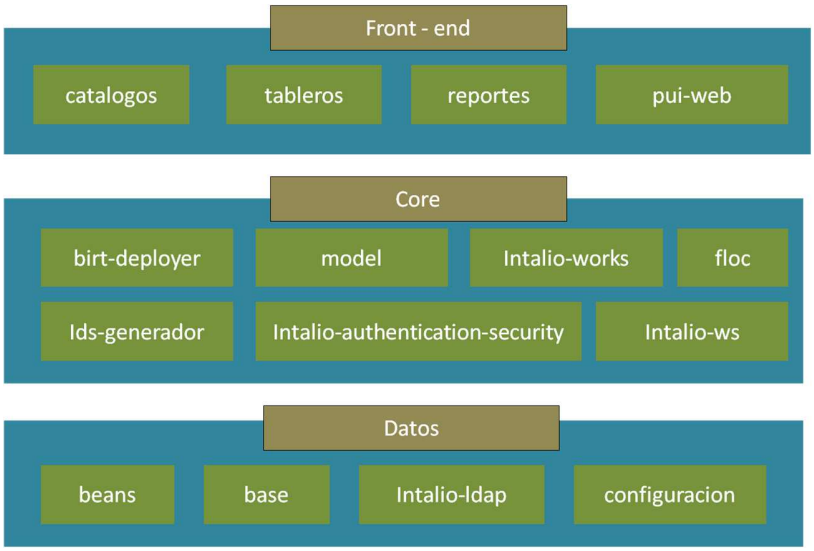
La interconexión de aplicaciones web entre sí para garantizar seguridad en el envío de peticiones tipo POST mediante AJAX, lo solucione usando un contexto compartido, esto representa un problema, ya que las aplicaciones deben convivir en el mismo servidor, de no ser así habría que garantizar otro método para asegurar el paso de parámetros entre aplicaciones de manera segura y eficiente.

La manera de presentar las tareas, notificaciones, solicitudes y reportes es en un grid hecho con un plugin denominado jqgrid que utiliza jQuery como núcleo, me representó un problema dado que los servicios regresan en su mayoría XML, pero no un XML adecuado para el grid elegido, por lo que se recurrió a utilizar XSLT para transformar el mensaje, y en otras ocasiones utilice un formato JSON. Esto representó crear clases que pudieran convertir una petición cualquiera a estos MIME-TYPES, en un contexto de spring.

Otros problemas que me surgieron y que surgen en toda aplicación web, fueron los relacionados a configuración e integración de múltiples frameworks con spring, así como soporte de codificaciones de caracteres e internacionalización.

A pesar de ser Intalio un BPMS de código abierto, fue necesario realizar Ingeniería inversa para determinar cómo se comporta Intalio a nivel de base de datos, ya que es un aspecto no documentado, de esta forma fue necesario interceptar peticiones vía wireshark y revisar como afectaban a nivel base de datos para poder conformar una interfaz intermedia que interactuara directamente con Intalio a nivel de base de datos.

En la figura 3.4, se muestran los componentes que se tuvieron que construir para poder tener a PUI funcionando con Intalio.



Componentes de PUI, Fig.3.4

A continuación se describen brevemente cada uno de estos componentes, ya que este trabajo se centra solo en PUI la parte web que es solo un componente de los mostrados.

Catálogos: Este componente es encargado de gestionar y poner toda la base para crear catálogos casi al vuelo, ya que se crean a base de configuración en la base de datos de PUI, en ella se dan de alta los catálogos como registros, se especifican las columnas que contiene, los tipos de dato, si será o no editable y el origen de datos para popularlo.

Tableros: Los tableros son conformados por gráficas y tablas de datos que reflejan estadística del proceso, número de procesos iniciados, el estado en el que se encuentran, etc. Este tablero interactúa directamente con una librería javascript llamada highcharts, para la cual se creó el soporte de lado de java para ser integrable con PUI.

Reportes: Este proyecto consiste en un conjunto de servicios web que leen información de configuración desde un lugar en específico en el sistema de archivos de manera remota, es decir, existe un servidor (Birt-deployer) el cual sirve archivos de configuración de cada reporte posible a desplegar, una vez obtenido este archivo de configuración en PUI se muestra un registro que representa al reporte y que contiene la URL destino a la cual se accederá al mostrar el reporte, entre otros parámetros.

PUI-Web: Es la interfaz principal, la cual es abordada a lo largo de este trabajo, es un eje central para el consumo, transformación, y representación de la información.

Birt-deployer: Es un servidor que emplea SocketServer (java.net) para escuchar por peticiones, mediante configuración sirve de archivos XML de configuración de acuerdo a las peticiones recibidas.

Model: En este proyecto se alojaron principalmente interfaces y clases utilitarias para reutilizar en otros proyectos.

Intalio-works: Es un proyecto que implementa acciones básicas para interactuar con servicios de Intalio a nivel de tareas y notificaciones, está construido con base en el patrón command.

Floc: Form Locator o localizador de formas, es un proyecto que sirve como capa intermedia para poder configurar orígenes de formularios web, de esta forma es posible emplear no solo ajax forms o x-forms.

Ids-generator: Cada actividad que se genera en Intalio es necesario identificarla, para realizar correlación y poder llevar el seguimiento a través de las distintas fases en el proceso. Por ello para cada proceso generado se necesita generar un identificador único, este proyecto tenía esa función, era simplemente un servicio web que generaba números únicos.

Intalio-authentication-security: Spring security tiene muchas características y entre ellas está la de soportar múltiples modos de autenticación y soporte de plataformas de seguridad, a pesar de ello fue necesario implementar un tipo de autenticación más para que se amoldara a Intalio y a PUI. De esta forma surge el proyecto de Intalio-authentication-security que sirve de puente entre el servicio de autenticación que expone Intalio el cual funciona a través de tokens y la interfaz web de PUI a través de sesiones web, todo en un esquema de seguridad RBAC.

Intalio-ws: Es un proyecto que sirve para consumir los servicios de Intalio, para tareas, notificaciones y procesos. Contrario a Intalio-works que opera sobre las actividades de cada tarea, notificación y proceso, como aceptar, delegar o declinar por ejemplo.

Beans: Aquí se conjuntaron todos los POJOS para todos los proyectos, evitando repetirlos.

Base: Sirve como molde para generación de catálogos, transformaciones e interacción con LDAP.

Intalio-ldap: Funciona como interfaz para interacción con LDAP para realizar operaciones de búsqueda, inserción, eliminación y actualización.

Configuracion: Es el proyecto principal de configuración para PUI, en él se especifican los origines de datos, es decir, de donde se consumirán los servicios de Intalio, donde está configurado LDAP, las credenciales, mensajes para consumo de servicios web, etc.

3.4 Alcance

El alcance contempla las siguientes funcionalidades:

a) Realizar el diseño y construcción de la arquitectura hibrida basada en MVC, SOA y SOUI de PUI con las siguientes funcionalidades y mediante el apoyo de spring framework:

   a. Filtrar tareas, notificaciones, solicitudes y reportes de acuerdo a un rol seleccionado
   b. Mostrar Tareas
      i. Añadir o quitar columnas (vía configuración)
      ii. Realizar búsqueda sobre una columna
      iii. Paginar tareas
      iv. Ocultar/mostrar columnas extras
      v. Mostrar Tarea
         1. Completar Tarea
         2. Guardar cambios en Tarea
         3. Realizar otras funciones
      vi. Reclamar tarea
      vii. Revocar tarea
      viii. Reasignar tareas
   c. Mostrar Notificaciones
      i. Realizar búsqueda sobre una columna
      ii. Paginar notificaciones
      iii. Mostrar Notificación
      iv. Ocultar notificación
   d. Mostrar Solicitudes
      i. Realizar búsqueda sobre una columna
      ii. Paginar solicitudes
      iii. Mostrar solicitud
         1. Iniciar proceso

e. Mostrar Reportes
    i. Mostrar reporte
    ii. Realizar búsqueda sobre una columna
    iii. Paginar reportes

b) Construir otros componentes y conectarlos mediante la tecnología de cross-context, se realizó el diseño de un Controlador Localizador de Formas (FLOC)
    a. Configurar rutas hacia las formas de captura
    b. Desplegar las rutas de acuerdo a una ruta asociada en el proceso particular

c) Implementar una interfaz de autenticación que cumpliera con spring-security y que usara la autenticación de Intalio.

3.5 Metodología

La metodología empleada fue una combinación de XP (eXtreme Programming), D3 (Design Driven Development) y TDD (Test Development Driven).

a) Design Driven Development (D3); Se basa principalmente en el diseño, enfatizando la usabilidad, de esta forma se garantiza que lo que el cliente requiere este lo más apegado a lo que se está desarrollando, el diseño no solo en cuanto a visualización sino a arquitectura, mantener las cosas simples y concisas.

b) eXtreme Programming; Permite cambios a lo largo del desarrollo, por ello el diseño está hecho de tal forma que se puede adaptar rápidamente a un cambio. Todas las partes están involucradas en el desarrollo de software, de esta manera un error no es particular, sino que se afronta desde todas las perspectivas.

c) Test Development Driven; Se enfoca principalmente en que antes de realizar la implementación mediante código primero se creen las pruebas unitarias correspondientes a lo que se desea implementar.

Adicionalmente, tome el principio de que el código sea consistente con lo que hace para que sea autodocumentado y así cualquier miembro del equipo pueda entender, modificar o corregir algún error en el funcionamiento o diseño.

Otra práctica que tome en cuenta es la colaboración en conjunto, es decir, más de una persona enfocada en lo mismo, con el fin de obtener los mejores resultados, esto no necesariamente es perder el tiempo o disminuir la producción ya que se mejora la calidad del código y se toman distintos puntos de vista permitiendo que la implementación sea más robusta, asimismo la revisión entre colegas es innecesaria porque ya se está realizando.

La programación por parejas es una práctica que se da comúnmente en metodologías ágiles como SCRUM y XP, para tener una efectividad en esta programación por parejas es necesario intercambiar roles, es decir, el que programa no siempre debe ser el mismo, de igual forma el que observa y hace comentarios, otro aspecto a considerar es el tiempo personal, a pesar de estar enfocados en resolver lo mismo se necesita de un tiempo a solas para hacer cualquier otra cosa y así evitar el hostigamiento.

En la Fig. 3.5 se muestra el ciclo de desarrollo XP en una versión simplificada. Se puede observar que todo está interrelacionado y que lo más importante, lo que está en el núcleo de este ciclo de desarrollo, es la comunicación diaria y el mejoramiento de las personas, siendo así, si las personas no contribuyen a esta metodología de trabajo muy probablemente el proyecto fracasará al emplear esta metodología.

Ciclo de desarrollo XP Fig. 3.5

3.6 Planeación

3.6.1 Anécdotas de Usuarios

Las anécdotas de usuarios sirven para el mismo propósito que los casos de uso, no obstante no son lo mismo. Estas anécdotas de usuario son realizadas para describir las necesidades que tienen el usuario y que el sistema pretende resolver.

Para el caso de PUI estas anécdotas fueron documentadas a través del tiempo de las variadas implementaciones de procesos, las necesidades que iban surgiendo y los modelos actuales que se manejaban, las limitaciones, etc.

Una vez recopiladas estas anécdotas fue sencillo vislumbrar cuales eran los requerimientos necesarios para realizar el diseño y la construcción obteniendo solamente lo más relevante para las necesidades del usuario sin tomar en cuenta requerimientos no funcionales como el manejador de base de datos, el sistema operativo o detalles de interfaz gráfica.

3.6.2 Reunión para la planeación de liberación

Esta reunión es una de las más importantes ya que en ella se construye el plan de la liberación el cual rige a lo largo del desarrollo. Algo significativo en este tipo de reuniones es permitir tomar las decisiones a quienes les corresponde, gente técnica toma decisiones técnicas y gente con conocimiento del negocio toma decisiones acerca del mismo.
La reunión para la planeación de la liberación involucró a la mayoría de los involucrados en el proyecto, dos ingenieros de software y un arquitecto de software. En esta reunión se estimaron cada una de las anécdotas, y se determinó cuantas anécdotas podrían realizarse simultáneamente plasmándolo en una hoja de Project, donde el líder de proyecto va dando seguimiento lo más puntual posible.

3.6.3 Reuniones de planeación de las iteraciones

Cada planeación de iteraciones fue realizada al inicio de cada iteración, contando con ocho reuniones en total, dos reuniones cada mes.
En estas reuniones se discutía acerca de que anécdotas deberían realizarse primero de acuerdo a las necesidades del cliente y que desviaciones se estaban presentando con base en lo ya programado. De esta manera el desarrollo de PUI mediante XP se vio acelerado ya que era posible ir realizando pequeñas liberaciones y así las pruebas de

aceptación por parte del usuario resultaron agiles. Si una iteración resultaba pequeña en estimación por semanas para las anécdotas contempladas fue posible aceptar en algunas ocasiones alguna otra anécdota como mejora al sistema.

En estas reuniones participa todo el equipo ya que es necesario que todos se encuentren enterados de las decisiones y como afectan a su plan de trabajo personal. Como resultado de las reuniones se modifica el plan general.

3.6.4 Investigación

Debido a la falta de documentación técnica de cómo funciona en su interior Intalio, tuve que determinar mediante ingeniería inversa como funciona tempo (parte importante de Intalio que controla todo el flujo y estado de una tarea) y los servicios que ofrece, en un primer acercamiento se utilice una herramienta llamada SOAP-UI que permite realizar llamadas a servicios, con esta herramienta fue posible determinar cómo se puede llevar a cabo todo el ciclo de vida de un proceso y como es que intercambian información UI-FW y tempo.

Por otra parte realice un análisis a UI-FW, desde ver el código fuente, hasta revisar las llamadas que realiza en cada acción o mediante cada funcionalidad, por medio de un analizador de protocolos llamado wireshark.

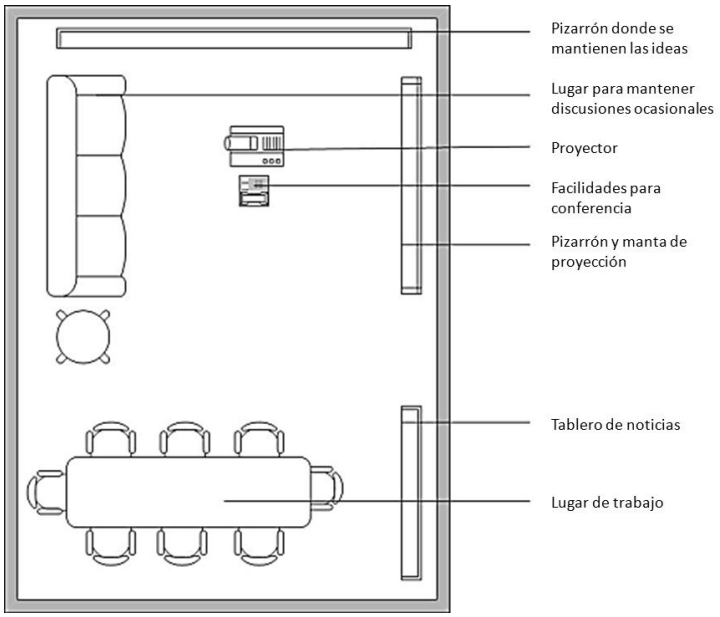
Al tiempo que se realizaban estos análisis observe el comportamiento en las tablas de la base de datos, lo cual permitió determinar las alteraciones que sufre la base de datos cada vez que se realiza algún cambio en tempo.

3.7 Administración

3.7.1 Espacio de trabajo abierto

La comunicación es lo más importante cuando se está realizando un desarrollo basado en la metodología XP. D3 nos muestra un ambiente de trabajo el cual promueve que las ideas surjan y que la comunicación sea efectiva.
Lamentablemente no pude lograr que se diera esta forma de trabajo, lo más que logré fue que se quitaran las barreras entre cubículos. La figura 3.6 muestra el espacio ideal recomendado por la metodología, en él se muestra un espacio abierto lo que permite que

la comunicación fluye ágilmente y se disponen de las herramientas necesarias para realizar o mostrar cualquier idea que surja.



Espacio de trabajo propuesto por D3 Fig. 3.6

Según D3 el entorno ideal para llevar acabo el desarrollo y que las ideas surjan de manera fluida, así como que la comunicación sea efectiva, involucra un lugar iluminado, espacioso, arreglado para entablar discusiones casuales, y equipado con suficientes pizarrones.

A pesar de no poder contar con el entorno propuesto, la comunicación siempre fue efectiva ya que se procuró dar tiempo para todo, manteniendo espacios personales, y cordialidad entre los miembros del equipo, comunicándonos a diario, en cada momento de inquietud, propiciando confianza entre los integrantes.

3.7.2 Reuniones diarias

Las reuniones diarias debido al reducido número de miembros involucrados en el proyecto pudieron darse eficazmente, sirvieron para corregir, detectar problemas al momento y no dejar pasar mucho tiempo intentando resolver lo mismo.

Estas reuniones diarias ocasionalmente eran grupales, casi siempre fue individualmente para recabar el avance y no hacer perder tiempo a demás integrantes, la duración no era más de diez minutos.

3.8 Diseño

3.8.1 El diseño como accidente

Inicialmente partí de las necesidades del cliente, de las anécdotas recopiladas con base en eso los controles fueron apareciendo y el diseño en si resulto simple, pero no fácil de concretar, es decir, a pesar de que visualmente el diseño parezca simple, el encontrar los componentes adecuados y la disposición de los mismos no fue sencilla.

Teniendo el conocimiento de cómo funciona tempo a nivel de base de datos, llevé a cabo el diseño de los servicios que permitirían a PUI interactuar con tempo y con los procesos. El resultado de este diseño se plasma en los diagramas de arquitectura a un nivel general.
Antes de empezar con este diseño no contaba con conocimiento acerca de tecnologías como hadoop, ETL, o ESB, es por ello que el consumo se realizó de manera directa, sin embargo evolucionar la arquitectura para hacerla más robusta no tendría problema porque hay muy bajo acoplamiento entre los servicios que conforman el sistema.

3.8.2 Innovación

Como innovación PUI incorpora un grid basado en jQuery que es alimentado vía un controlador (servlet) el cual consume un servicio de acuerdo a la opción seleccionada en un menú. El grid idealmente está hecho para trabajar con XML, php y/o .NET, spring en ninguna de sus vistas retorna XML, así que diseñé un view personalizado para poder retornar XML y poder alimentar al grid, y la implementación realizada para retornar los datos en forma de XML está hecha con java, con anotaciones y diversas APIs.

3.8.3 Interacción

Algo que es muy importante cuando se realiza software, no importando si será una GUI o web, es que sea intuitivo para el usuario además de agradable a la vista. El diseño en

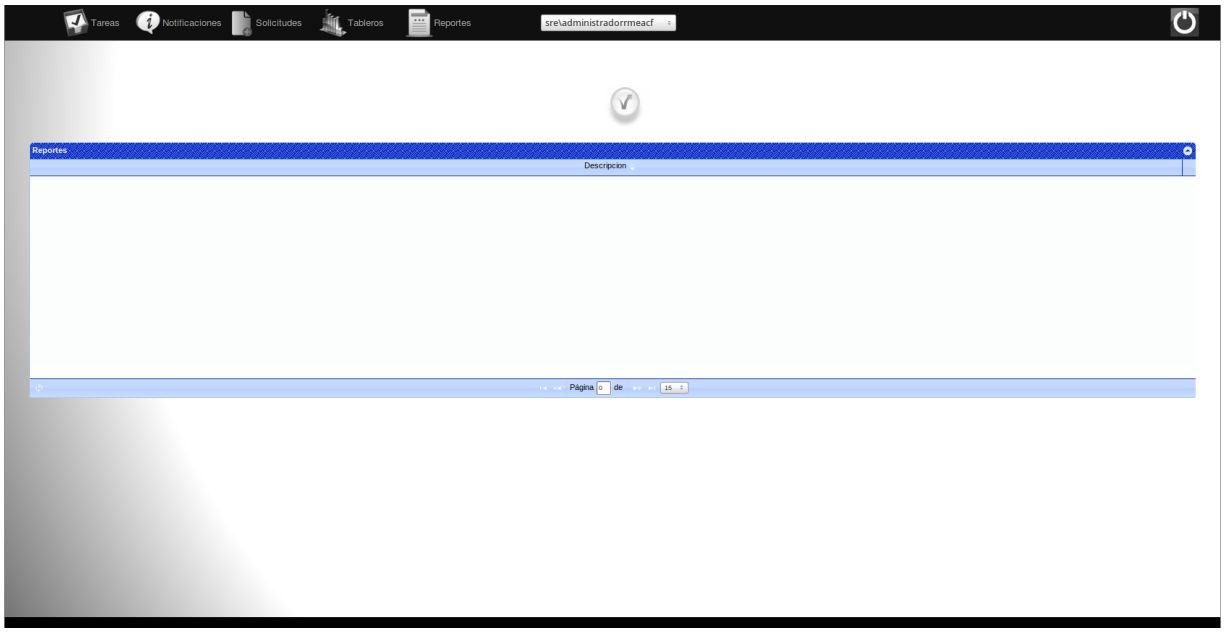
esta parte jugo un papel muy importante ya que tuve que pensar muy bien la manera en que sería natural para el usuario consultar la información a través de PUI.

Como previamente comenté en innovación, la interfaz es simple, consta de un menú no estático, es deslizable, un combo de selección y un botón para poder salir de la aplicación. Esto es más que suficiente para realizar la mayoría de las interacciones, cosas adicionales se le delegaron al grid, por ejemplo el desplegar una tarea, ordenación de tareas, etc.

En la figura 3.7 se muestra la primera versión de PUI, donde se cargaba el contenido a petición a partir de AJAX y se tenían opciones básicas con un diseño muy colorido.
Para la versión 1.5 (Fig. 3.8), se cambia el diseño tanto del administrador como del portal principal y se evitan abrir diálogos en ventanas externas, es decir, los formularios, reportes y demás elementos gráficos se despliegan en la misma ventana con ayuda de un Iframe.

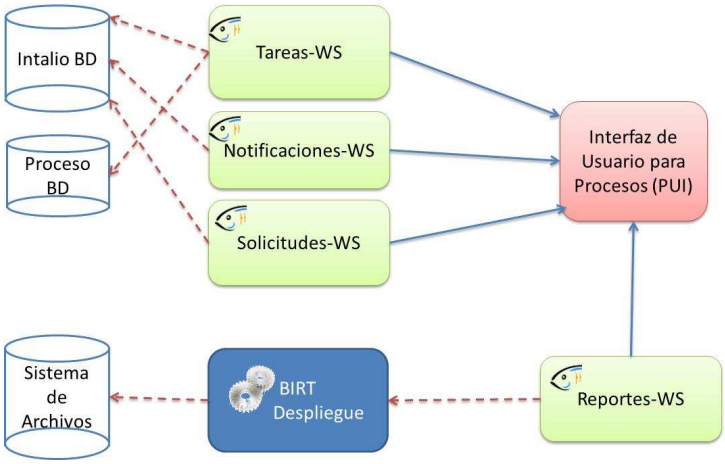
Interfaz de Usuario para Procesos v1.0 (PUI) Fig. 3.7

Se le denominó PUI 1.5, ya que los cambios fueron mínimos y fueron principalmente a la interfaz gráfica, se tenía pensado cambiar más aspectos tanto gráficos como de backend y generar una versión 2.0 mucho más robusta, debido a mi salida de Saitosoft esto ya no se concretó.



Interfaz de Usuario para Procesos v1.5 (PUI) Fig. 3.8

Cabe mencionar que PUI es la única solución en toda la empresa que tiene este tipo de arquitectura. En el diagrama de la Fig.3.9 se muestra como es la interacción para obtener datos y mostrarlos en el front-end, para cada uno de los apartados disponibles en PUI.



Interacción de componentes, conceptual Fig. 3.9

Cuando se solicitan notificaciones o solicitudes, mediante el componente jqgrid se envían parámetros de paginación, ordenación, búsqueda, así como datos del usuario que se encuentra en sesión, el servicio web los procesa y realiza el consumo de datos de la fuente correspondiente, regresa los datos en forma de respuesta xml usando el protocolo SOAP y PUI mediante XSLT realiza la transformación para que el grid sea alimentado.

El diagrama de la Fig. 3.10 muestra cómo se despliega un grid de acuerdo a un elemento seleccionado en el menú, esta es la implementación general, las clases que realizan el trabajo de acuerdo al ítem del menú seleccionado extienden de estas clases mostradas.

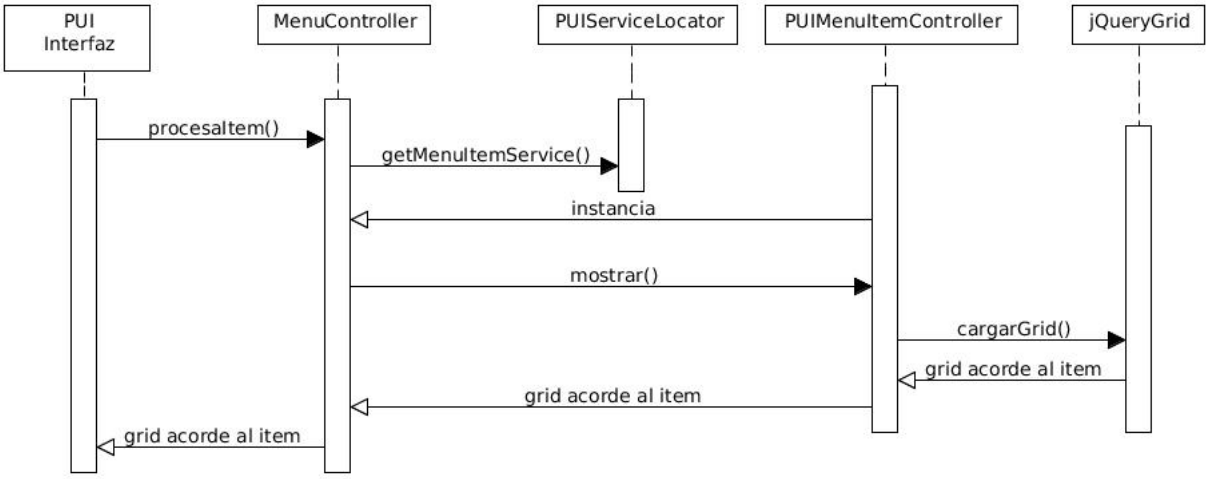


Diagrama de secuencia para desplegar un grid Fig. 3.10

En el caso particular del servicio web de tareas, cuando se hace una invocación, se consumen dos fuentes de datos (si el proceso en cuestión tiene una base de datos asociada) para poder recuperar información tanto de la entidad en cuestión (la tarea) como de datos asociados a esa tarea en otra base de datos.

El grid de tareas también es particular, es construido a partir de un XML que define la configuración, esta configuración incluye los parámetros para realizar el consumo de una fuente de datos adicional. La ventaja de esto, es que esa fuente de datos puede estar en cualquier RDBMS y la configuración puede ser tan completa y compleja como lo permita la imaginación y el SQL.

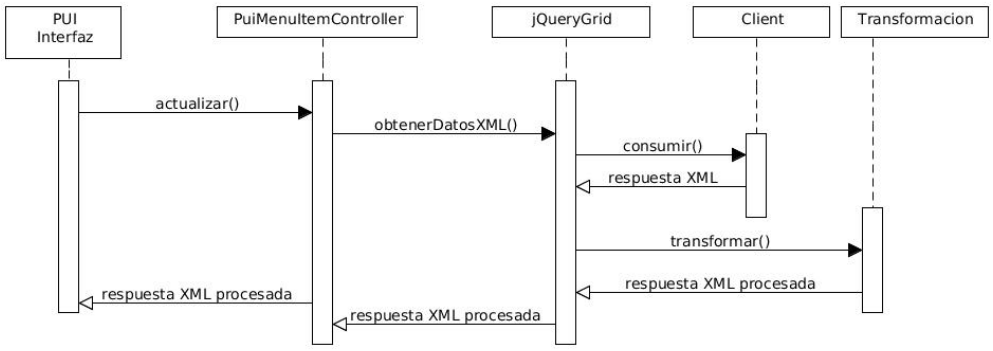


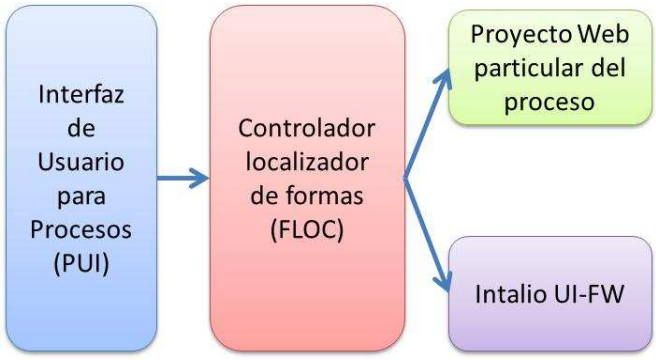
Diagrama de secuencia, consumo de un servicio y transformación Fig. 3.11

Por último, el servicio de reportes, es auxiliado de un servidor desarrollado para múltiples fines relacionados con reportes hechos en BIRT, uno de estos fines es realizar búsquedas sobre el sistema de archivos para reportes en particular, este servidor es independiente de la plataforma y se integra como servicio al sistema operativo.

3.8.4 Información

Siguiendo algunas pautas de disposición de elementos en la pantalla, se ordenaron los controles de PUI, el menú en la parte superior izquierda junto con el combo, el salir de sesión en la esquina derecha superior, y controles internos del grid justo en medio.
De acuerdo a la utilización que se le ha venido dando, ha resultado bastante bien, no se han requerido disponer los controles de otra forma, lo único que ha cambiado son los colores por políticas propias de los lugares donde se ha instaurado PUI.

En el diagrama mostrado a continuación (Fig.3.12) se observa cómo se obtienen a las interfaces de usuario (necesariamente web) para mostrar solicitudes, tareas, o notificaciones a partir de diferentes orígenes.



Interacción de componentes, conceptual F 3.12

A partir del grid construido previamente, es posible determinar la interfaz correspondiente a esa tarea, solicitud o notificación, siendo así FLOC, busca si tiene conocimiento de esa tarea, solicitud o notificación, es decir, busca si existe un proyecto web que mapee a esa ruta e intenta desplegar la versión alternativa, si ocurre un error o no existe tal mapeo se lanza la interfaz default de Intalio ui-fw. FLOC agrega un sin fin de posibilidades, ya que es posible mostrar en vez de las interfaces de default cualquier cosa, un reporte, una interfaz enriquecida, alguna conexión con un sistema, además de utilizar expresiones regulares para la identificación de las mismas.

3.8.5 Interfaz

Usabilidad es un término que describe la aceptación del usuario y es algo de lo más difícil de lograr, ya que dependiendo el impacto que se quiera tener, se deben elegir los colores, la disposición, la forma de interacción con los elementos que componen a la página web.

De aquí deriva el término de diseño de interfaz, para PUI se optó por usar algunas cosas de CSS3 como son los gradientes, este pequeño detalle da un matiz de color a la interfaz que en navegadores como Internet Explorer no se nota por problemas de compatibilidad, pero en otros como Mozilla Firefox o Google Chrome le da al usuario una sensación de iluminación y proporciona una pantalla moderna.

Detalles como la luz, el matiz de color, el deslizado de los elementos en el menú o el mostrar las tareas, solicitudes, reportes y notificaciones en un Iframe pero mediante una animación y oscurecimiento del fondo de pantalla proveen al usuario una experiencia de web 2.0.
Quizás en un futuro sea posible incorporar la web 3.0, sin embargo este objetivo requiere de mucha investigación y maduración de este concepto.

3.9 Codificación

3.9.1 Seguimiento de estándares

Es importante que cuando se realiza un desarrollo se sigan estándares sin ellos cada persona tiene su forma de hacer las cosas y la revisión o auditoria del código se vuelve un caos.

En PUI usamos los estándares establecidos para Java ya sean los de las convenciones de Oracle (antes SUN Microsystems) o bien los establecidos en la empresa.

Entre los estándares establecidos se encuentran convenciones de nombres, disposición de los elementos de una clase (primero atributos, luego constructor, métodos accesores y mutadores, y/o métodos adicionales). Estandarización de donde deben ir cada uno de los elementos de la aplicación, jsp's, javascript, CSS, XML de configuración, WSDL e identación del código.

3.9.2 Programación por parejas

Este concepto es de los más interesantes de XP, el trabajar dos personas en un mismo objetivo para los directivos sería como desperdiciar recursos, sin embargo de acuerdo a la práctica que obtuve, los resultados son bastante satisfactorios. El nivel de calidad de software se incrementa y es mejor la implementación al considerar todos los puntos de vista se atacan más puntos y se depura mejor el código.

Esta práctica no resulta sencilla ya que debe haber empatía entre las personas que trabajan juntas, de no ser así los resultados son totalmente opuestos a lo que se busca, y es muy común en esta carrera es de saberse que el tener la razón es una discusión que se tiene frecuentemente, debido a esto es importante reconocer nuestros defectos, carencias de conocimiento, pero también hacer ver a los demás cuando tenemos razón.

PUI al menos en la parte de servicios fue hecho con programación en parejas, dos personas realizaban el mismo trabajo y se retroalimentaban el uno con el otro, a pesar de esto en ocasiones no pudieron dar solución a algún problema y fue necesaria mi intervención. Dicho esto, la programación por parejas no garantiza que se solucionen todos los problemas y en ocasiones hay que acudir con un tercero, no obstante es una práctica que resulta satisfactoria la mayoría de las veces.

3.9.3 Integración del código al momento

Una práctica importante en XP es la integración al momento, luego de terminar algún modulo, o sección de código, rutina, algoritmo etc. Es necesario integrarlo a la solución para determinar si existe algún problema o si afecta de alguna forma la solución.

PUI fue hecho mediante consumo de servicios web, una vez terminado un servicio web fue casi natural el integrarlo y probar la funcionalidad del mismo desde PUI, en ocasiones

esto llevó más tiempo de lo esperado por consideraciones de negocio no vislumbradas en un principio y por algunos conflictos de compatibilidad con manejadores de base de datos.

Un problema en particular que surgió fue el soporte de integración con bases de datos cuyas llaves primarias fueran compuestas, lo cual retrasó el proyecto en al menos dos semanas. Prácticamente se tuvo que rediseñar e implementar el servicio de tareas.

3.9.4 Entorno de integración

Siempre debería existir un entorno de integración para desarrollo el cual mantenga a salvo la estabilidad de la solución en cuestión. Pese a todas las pruebas unitarias o de estrés, la integración resulta ser de lo más complicado y laborioso cuando se desarrolla.

Este entorno de integración debe ser idéntico al entorno de producción o al menos lo más parecido, así se pueden determinar posibles fallos a causa de no tomar consideraciones con respecto al sistema operativo, base de datos, reglas en el firewall o problemas con la red.

3.10 Pruebas

El modelo TDD dicta que primero se construyan las pruebas unitarias antes de construir lo que se desea probar, esto facilita la implementación ya que se considera los test fallidos y los test exitosos lo que deriva en una implementación más robusta al considerar los casos posibles de falla, éxito y no solo avocarse a la mera funcionalidad.

Otra bondad de realizar primero los test y no el código es que a medida que se avanza en el desarrollo de los componentes se pueden seguir ejecutando las pruebas y así se reducen el número de problemas de integración entre componentes, así como que no se haya roto una funcionalidad por modificaciones a otros productos.

Esta fase del desarrollo no debe consumir mucho tiempo ya que solo se prueban escenarios conocidos o esperados de falla, lo recomendable es que alguien distinto al programador pruebe lo programado.

3.11 El fondo de la implementación

3.11.1 Patrones implementados

Es interesante el conocer los patrones de diseño, y el realizar la implementación resulta sencillo, lo interesante de ellos es cuando, como y donde usarlos.

El libro Design Patterns: Elements of Reusable Object-Oriented Software es un eje principal para el entendimiento de los patrones, en él se describen los 4 tipos principales así como la implementación, lamentablemente conocí el libro hasta después de haber realizado todo el diseño, casualmente muchas ideas que tuve encajaron con lo que ya había sido pensado por otros y plasmado en este libro.

Algunos de los patrones usados en PUI son los siguientes:

- Service Locator: Mediante la ayuda del session facade es posible determinar el servicio que se requiere para procesar una petición.

Ejemplo de implementación en PUI:

```
public PuiMenuItemController getMenuItemService(MenuItems menuItem,
          Configuracion conf){

    switch(menuItem){
          case notificaciones:
              return nc;
          case reportes:
              return rc;
          case solicitudes:
              return sc;
          case tareas:
              return tc;
          default:
              throw new UnsupportedOperationException("Operación no soportada");

     }
}
```

En este caso dependiendo del ítem seleccionado se retorna un objeto que implemente la interfaz PuiMenuItemController, este objeto realiza acciones predefinidas que son comunes a todos los demás manejadores de ítems.

- Singleton: Crea una sola instancia de una clase, ideal para conexiones a base de datos.

Ejemplo de implementación en PUI:

```
public class HibernateUtil {

    private static SessionFactory sessionFactory;
    private static HibernateUtil instanceDB;

    /**
     * Configuración en hibernate.hbm.xml
     */
    private HibernateUtil() {
       sessionFactory = new Configuration().configure().buildSessionFactory();
    }

    public static HibernateUtil getInstance() {
       if (instanceView == null) {
           instanceView = new HibernateUtil();
       }
       return instanceView;
    }

    public SessionFactory getSessionFactory() throws Exception {
       if(sessionFactory1 !=null && active){
            return sessionFactory1;
        }
    }
}
```

Este es un singleton para obtener una conexión a una base de datos configurada en un archivo hbm de hibernate.

- DAO (Data Access Object): Es un patrón que permite el acceso a datos y proporciona la respuesta a través de POJOs (Plain Old Java Object) generalmente.

En PUI los daos básicamente fueron utilizados para consumir los datos a partir de vistas creadas en el DBMS, como ya explique previamente una vista de tareas, una vista de notificaciones y una vista de solicitudes, existiendo así un objeto que las representa a cada una y un objeto de acceso a datos que provee la interacción con las mismas.

- Front Controller: Centraliza las peticiones que serán distribuidas de acuerdo al session facade y el service locator.

La parte central de PUI está en un controlador que involucra además de este patrón, el business delegate, el session facade y el service locator al mismo tiempo, es notable la

reducción de código que esto conlleva en vez de tener controladores con código interminable, prácticamente dos líneas de implementación.

```
@RequestMapping(value = "/{menuItem}/{rol}")
public ModelAndView procesaItem(@PathVariable MenuItems menuItem,
          @PathVariable String rol,
          HttpServletRequest request,
          HttpServletResponse response) throws PUIException {

    PuiMenuItemController pmic = psl.getMenuItemService(menuItem, conf);
    return pmic.mostrar(rol, menuItem);
}
```

- Business Delegate: Permite determinar a qué objeto de negocio debe delegarse una acción solicitada mediante la interfaz.

- Session Facade: Reduce el acoplamiento y generaliza el comportamiento de los objetos ocultando la implementación de los mismos.

Al realizar las implementaciones vía contrato a través de interfaces, la implementación de este patrón en PUI resulta simple y es implementado por el business delegate, que a su vez invoca al service locator y obtiene los componentes.

```
@Service
public abstract class PuiMenuItemController {

    @Autowired
    protected PuiServiceLocator psl;
    protected Properties propiedades;
    protected jQueryGrid servicio;
    protected Filtro filtro;
    protected Configurable configuracion;

    public ModelAndView mostrar(String rol,MenuItems menuItem) throws PUIException{
       filtro.setRole(rol);
       servicio=psl.getService(menuItem);
       return servicio.cargarGrid(configuracion, filtro);
    }

    public abstract ModelAndView actualizar(HttpServletRequest request,Filtro filtro)
    throws PUIException;
}
```

Este es un ejemplo de contrato, en este caso no es precisamente una interfaz ya que de manera general el método mostrar era idéntico para todos los ítems, de tal suerte que opte por una clase abstracta.

3.11.2 La solución

La idea de generar una interfaz de procesos mediante la tecnología java la pude haber abordado de muchas formas, usar todo lo que J2EE proporciona o algunos otros stacks como los de JBOSS, o spring puro, sin embargo la manera de usar frameworks diferentes y APIs coadyuvaron a tener una solución bastante estable y funcional, aparte de fácil configuración y desarrollo.

El uso de patrones de diseño simplifico de forma importante las líneas de código y todo esto no puede darse sin conocimiento previo, conocimiento que se adquiere mediante la práctica, mediante probar y equivocarse, es acerca de tomar decisiones adecuadas no por intuición sino por experiencias previas.

PUI ha resultado ser una herramienta importante para la empresa, y en manos correctas el paso del tiempo hará de él lo que a los vinos, mejorarlo.

3.11.3 De los request, service y transformation

Para facilitar el consumo de los servicios web de tipo SOAP así como las transformaciones, se implementaron interfaces y clases abstractas, lo cual reduce el consumo de servicios a crear cuatro clases, bastante parecido a la implementación de los EJB (Enterprise JavaBeans) antes de su versión 3.

Para consumo de servicios de tipo REST solo se optó por emplear el objeto RestTemplate provisto por Spring 3.0.

A continuación se muestran estas interfaces con el fin de mostrar la generalización.

Clase client:

```
public abstract class Client {

    /**
      * Realiza el consumo del servicio, crea todo lo necesario para realizar el
consumo y transformaciones si     es que son requeridas, retorna un string con la
respuesta obtenida
     * @return
     * @throws PUIException
     */
```

```
    public String consumir() throws PUIException{
       String respuesta = "";
       try {
           if(filtro==null || configuracion==null)
              throw new PUIException("Se debe asignar la configuración y el filtro
antes de poder consumir el servicio");

           Dispatch<SOAPMessage> dispatch = service.createDispatch(port,
           SOAPMessage.class, javax.xml.ws.Service.Mode.MESSAGE);
           SOAPMessage response = dispatch.invoke(request.construyeMensaje());
           SOAPPart sp = response.getSOAPPart();
           Source src = sp.getContent();
           StreamResult result = new StreamResult(new ByteArrayOutputStream());
           Transformer trans = TransformerFactory.newInstance().newTransformer();
           trans.transform(src, result);
           ByteArrayOutputStream baos = (ByteArrayOutputStream)
result.getOutputStream();
           respuesta = new String(baos.toByteArray());
       } catch (TransformerException ex) {
           throw new PUIException("Error al transformar");
       } catch (SOAPException ex) {
           throw new PUIException("Soap Exception");
       }
       return respuesta;
    }

}
```

Esta clase cliente es necesaria para realizar el consumo de un servicio basado en SOAP.

Interfaz Transformation:

```
public interface Transformacion  {

    /**
     * Ejecuta la transformación
     * @param f El archivo xslt
     * @return Una cadena con la transformación realizada
     * @throws PUIException En caso de falló en la transformación
     */
    public String transformar(File f) throws PUIException;

}
```

La interfaz Transformation lee la configuración del archivo XML para realizar la transformación del mensaje XML mediante xslt.

Interface Request:

```
public interface Request {
    /**
     * Se debe implementar este método que construye un mensaje soap
     * @return el mensaje soap a enviar
     * @throws PUIException En caso de que falle la construcción
     * @see SOAPMessage
     */
    public SOAPMessage construyeMensaje() throws PUIException;
}
```

De acuerdo a la clase que implemente esta interfaz el método `construyeMensaje()` hará justamente eso, devolver el mensaje en forma de `SOAPMessage`.

Así con estas interfaces y clases el consumo de un servicio, transformación de la respuesta para alimentación del grid resulta en las siguientes líneas de código, esto es un caso particular de la implementación para la obtención de notificaciones.

```
public String obtenerDatosXML(Configurable configuracion, Filtro filtro)
          throws PUIException {

   NotificacionesClient nr=NotificacionesClient.getInstance(configuracion);
   nr.setFiltro(filtro);
   nr.setRequest(new NotificacionesRequest(filtro));
   NotificacionesTransformation st;
   st = new NotificacionesTransformation(nr.consumir());
   return st.transformar();
}
```

En un futuro espero sea posible mejorar esta implementación y trasladarlo a anotaciones ya que parecer ser la evolución natural.

3.12 Participación Profesional

La participación profesional que dejó este desarrollo servirá como una propuesta de un todo, es decir, un conjunto de frameworks y herramientas de desarrollo que se llevan bien entre sí, es un hibrido ya que la mayoría de las soluciones de este tipo optan por usar solo un conjunto de éstas, las más comúnmente utilizadas son las de java.net, las de Spring, las de JBoss, las de Apache, y las de Sun (actualmente Oracle); Sin embargo la arquitectura diseñada toma un poco de todas estas tecnologías y las logra hacer convivir sin mayores inconvenientes.

Así mismo el aprendizaje y conocimiento depositado en cada uno de los integrantes que apoyaron a realizar este desarrollo, el uso real de patrones de diseño, y no solo teórico de lo que representan, y la participación armónica de estos individuos aun participando en una misma tarea.

La herramienta construida tiene mucho involucrado y proporciona las bases para que otros ingenieros puedan hacerla evolucionar convirtiéndola en algo más poderoso y rentable.

Capítulo 4. Resultados y Conclusiones

4.1 Resultados

La generación de esta solución para la interacción con procesos automatizados en Intalio constituye algo nuevo, ya que en el mercado pocas empresas realizan desarrollo para esta plataforma, siendo así ésta una interfaz agradable y con una arquitectura SOA que permite integración con muchos otros sistemas de una manera sencilla, además de ser única. Es un desarrollo que ha tenido mejoras, que se está usando y se ha vendido, aporta valor agregado a las soluciones que provee Intalio, a la empresa por tener un producto, al uso de procesos y la mejora de interacción con estos.

La realización de este proyecto de software no resultó simple, ya que a pesar de ser Intalio un producto en su mayoría open source, la documentación de los mismos para el desarrollo es limitada o nula, por lo que la investigación e ingeniería inversa para conocer cómo funciona resultó complicada.

La visión que se tenía en un principio era la de poder llegar a utilizar un ESB (Enterprise Service Bus), sin embargo el tiempo y diversos factores que intervinieron no permitieron llegar a esta meta. No obstante se espera que este desarrollo siga mejorando y creciendo, representando un beneficio de aprendizaje y ejemplo de cómo se pueden realizar estas integraciones y como pueden convivir armoniosamente las personas en un desarrollo de software sin tener que llevar metodologías más estrictas.

Es importante considerar alternativas open-source y probar nuevas metodologías de desarrollo, más que llevarnos fracasos o retrasos en tiempos de entrega quizás nos podríamos llevar una agradable sorpresa y comenzar a cambiar la idea que principalmente en instituciones gubernamentales en nuestro país se tiene sobre el software libre y la plataforma Linux, que por falta de información o bien por la falta de tiempo se opta por sistemas ya hechos que solo es de configurar, sistemas estilo Microsoft, IBM, Oracle o de código cerrado y no free-source u open-source.

4.2 Conclusiones

Desarrollar un sistema es algo relativamente sencillo siempre y cuando la planeación y el diseño abarquen la mayor parte de los posibles escenarios, es decir, se considere

dificultad en la implementación de nuevas tecnologías, experiencia de los ingenieros, asignaciones en proyectos, esto en cuanto a planeación; En lo que a diseño se refiere es de vital importancia tener en cuenta todos los requerimientos, y documentarlos ya sea en anécdotas o casos de uso según la metodología a emplear, e ir plasmando todo el diseño en algún documento o sistema como Enterprise Architect, ya que los ingenieros de esta forma son capaces de seguir las actividades que tienen asignadas y desarrollarlas de acuerdo a un "esquema" o modelo plasmado permitiendo concluir el sistema en tiempo y forma.

PUI constituye solo una parte de un conjunto de proyectos, si madura lo suficiente sería muy bueno que SaitoSoft lo pudiera donar para hacerlo de código abierto y que crezca por sí mismo. Como primer diseño de arquitectura realizada y llevada a cabo, me siento satisfecho gracias a las enseñanzas de la carrera pude adquirir la capacidad de autoaprendizaje lo cual hasta este momento me ha servido de sobremanera, sin ello simplemente no sería nada.

Durante mi estancia en Saitosoft aprendí muchas cosas, principalmente administrativas y de gestión de personal, se debe entender de alguna forma a los compañeros de trabajo ya que no todos los días son 100% productivos, hay que recordar que a fin de cuentas somos humanos, que requerimos tiempo personal para divagar, para lidiar con problemas personales, para pagar cuentas del banco, inclusive hay días en que no se tienen las ganas de trabajar y la productividad decae. Las personas que son totalmente administrativas o de gestión de recursos humanos, muchas veces no entienden esto y se aplican sanciones de todo tipo, desde por tener retardos hasta por no terminar con el trabajo a tiempo, lo que aprendí en Saitosoft es que si se tiene la suficiente confianza entre los miembros de un equipo todo este tipo de problemas no son obstáculo para lograr los objetivos, ya que cada uno se siente comprometido con lo que está haciendo, cree en lo que está haciendo y confía en que si falla alguien más se lo hará notar.

Al momento de escribir este informe me encuentro laborando para Quarksoft, otra consultora de software que es mucho más grande, he aprendido nuevas metodologías y nuevas tecnologías, tengo nuevos compañeros, trabajo para un proyecto de gobierno, y con tristeza veo como se malgastan los recursos en el gobierno, como mucha gente solo va como coloquialmente se dice "a calentar la silla", como no se sienten felices por su trabajo y solo esperan a que de la hora de la salida para poder irse a casa, no hay entrega, no hay eficiencia, pero así es el gobierno generalmente.

Espero algún día poder realizar algún cambio significativo por mi país, sé que trabajando para la iniciativa privada es poco probable que lo haga, pero mientras mantenga la esperanza espero que ese día pueda llegar.

"Algunas veces, es más importante tener el problema correcto que la mejor solución"
Bjarne Stroustup

5. Anexos

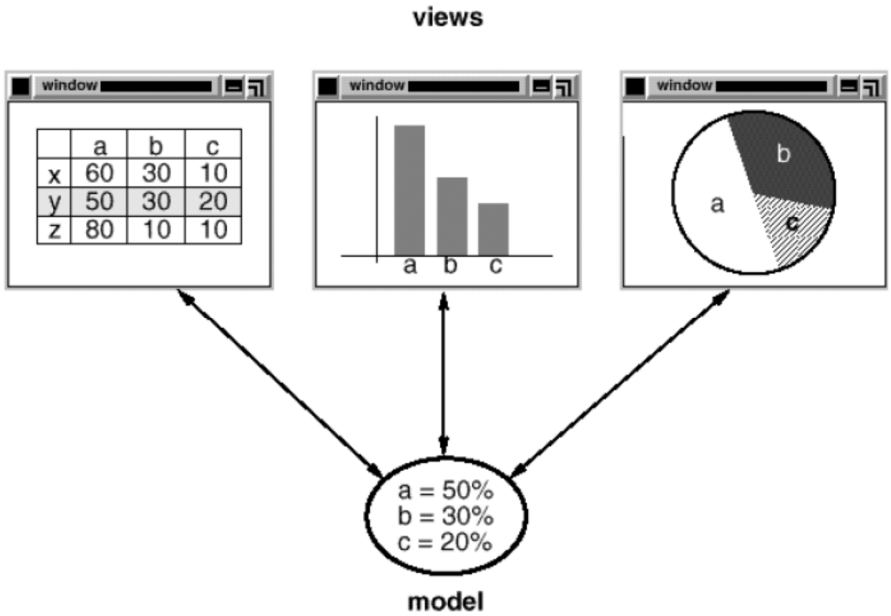
5.1 Design Patterns in Smalltalk MVC (Erich Gamma, 1994)

The Model/View/Controller (MVC) triad of classes [KP88] is used to build user interfaces in Smalltalk-80. Looking at the design patterns inside MVC should help you see what we mean by the term "pattern."

MVC consists of three kinds of objects. The Model is the application object, the View is its screen presentation, and the Controller defines the way the user interface reacts to user input. Before MVC, user interface designs tended to lump these objects together. MVC decouples them to increase flexibility and reuse.

MVC decouples views and models by establishing a subscribe/notify protocol between them. A view must ensure that its appearance reflects the state of the model. Whenever the model's data changes, the model notifies views that depend on it. In response, each view gets an opportunity to update itself. This approach lets you attach multiple views to a model to provide different presentations. You can also create new views for a model without rewriting it.

The following diagram shows a model and three views. (We've left out the controllers for simplicity.) The model contains some data values, and the views defining a spreadsheet, histogram, and pie chart display these data in various ways. The model communicates with its views when its values change, and the views communicate with the model to access these values.



Taken at face value, this example reflects a design that decouples views from models. But the design is applicable to a more general problem: decoupling objects so that changes to one can affect any number of others without requiring the changed object to know details of the others. This more general design is described by the Observer design pattern.

Another feature of MVC is that views can be nested. For example, a control panel of buttons might be implemented as a complex view containing nested button views. The user interface for an object inspector can consist of nested views that may be reused in a debugger. MVC supports nested views with the CompositeView class, a subclass of View. CompositeView objects act just like View objects; a composite view can be used wherever a view can be used, but it also contains and manages nested views.

Again, we could think of this as a design that lets us treat a composite view just like we treat one of its components. But the design is applicable to a more general problem, which occurs whenever we want to group objects and treat the group like an individual object. This more general design is described by the Composite design pattern. It lets you create a class hierarchy in which some subclasses define primitive objects (e.g., Button) and other classes define composite objects (CompositeView) that assemble the primitives into more complex objects.

MVC also lets you change the way a view responds to user input without changing its visual presentation. You might want to change the way it responds to the keyboard, for example, or have it use a pop-up menu instead of command keys. MVC encapsulates the response mechanism in a Controller object. There is a class hierarchy of controllers, making it easy to create a new controller as a variation on an existing one.

A view uses an instance of a Controller subclass to implement a particular response strategy; to implement a different strategy, simply replace the instance with a different kind of controller. It's even possible to change a view's controller at run-time to let the view change the way it responds to user input. For example, a view can be disabled so that it doesn't accept input simply by giving it a controller that ignores input events.

The View-Controller relationship is an example of the Strategy design pattern. A Strategy is an object that represents an algorithm. It's useful when you want to replace the algorithm either statically or dynamically, when you have a lot of variants of the algorithm, or when the algorithm has complex data structures that you want to encapsulate.

MVC uses other design patterns, such as Factory Method to specify the default controller class for a view and Decorator to add scrolling to a view. But the main relationships in MVC are given by the Observer, Composite, and Strategy design patterns.

5.2 Role Based Access Controls (David F. Ferraiolo, 1992)

ABSTRACT

While Mandatory Access Controls (MAC) is appropriate for multilevel secure military applications, Discretionary Access Controls (DAC) are often perceived as meeting the security processing needs of industry and civilian government. This paper argues that reliance on DAC as the principal method of access control is unfounded and inappropriate for many commercial and civilian government organizations. The paper describes a type of on-discretionary access control - role-based access control (RBAC) - that is more central to the secure processing needs of non-military systems than DAC.

1 Introduction

The U.S. government has been involved in developing security technology for computer and communications systems for some time. Although advances have been great, it is generally perceived that the current state of security technology has, to some extent failed to address the needs of all [1], [2]. This is especially true of organizations outside the Department of Defense (DoD). [3]

The current set of security criteria, criteria interpretations, and guidelines has grown out of research and development efforts on the part of the DoD over a period of twenty plus years.

Today the best known U.S. computer security standard is the Trusted Computer System Evaluation Criteria (TCSEC [4]). It contains security features and assurances, exclusively derived, engineered and rationalized based on DoD security policy, created to meet one major security objective - preventing the unauthorized observation of classified information. The result is a collection of security products that do not fully address security issues as they pertain to unclassified sensitive processing environments. Although existing security mechanisms have been partially successful in promoting security solutions outside of the DoD [2], in many instances these controls are less than perfect, and are used in lieu of a more appropriate set of controls.

The TCSEC specifies two types of access controls: Discretionary Access Controls (DAC) and Mandatory Access Controls (MAC). Since the TCSEC's appearance in December of 1983, DAC requirements have been perceived as being technically correct for commercial and civilian government security needs, as well as for single-level military systems. MAC is used for multi-level secure military systems, but its use in other applications is rare.

The premise of this paper is that there exists a control, referred to as Role-Based Access Control (RBAC), that can be more appropriate and central to the secure processing needs within industry and civilian government than that of DAC, although the need for DAC will continue to exist.

2 Aspects of Security Policies

Recently, considerable attention has been paid to researching and addressing the security needs of commercial and civilian government organizations. It is apparent that significant and broad sweeping security requirements exist outside the Department of Defense.[2], [5], [6] Civilian government and corporations also rely heavily on information processing systems to meet their individual operational, financial, and information technology requirements. The integrity, availability, and confidentiality of key software systems, databases, and data networks are major concerns throughout all sectors. The corruption, unauthorized disclosure, or theft of corporate resources could disrupt an organization's operations and have immediate, serious financial, legal, human safety, personal privacy and public confidence impact.

Like DoD agencies, civilian government and commercial firms are very much concerned with protecting the confidentiality of information. This includes the protection of personnel data, marketing plans, product announcements, formulas, manufacturing and development techniques. But many of these organizations have even greater concern for integrity. [1]

Within industry and civilian government, integrity deals with broader issues of security than confidentiality. Integrity is particularly relevant to such applications as funds transfer, clinical medicine, environmental research, air traffic control, and avionics. The importance of integrity concerns in defense systems has also been studied in recent years. [7], [8]

A wide gamut of security policies and needs exist within civilian government and private organizations. An organizational meaning of security cannot be presupposed. Each organization has unique security requirements, many of which are difficult to meet using traditional MAC and DAC controls.

As defined in the TCSEC and commonly implemented, DAC is an access control mechanism that permits system users to allow or disallow other user’s access to objects under their control:

A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control). [4]

DAC, as the name implies, permits the granting and revoking of access privileges to be left to the discretion of the individual users. A DAC mechanism allows users to grant or revoke access to any of the objects under their control without the intercession of a system administrator.

In many organizations, the end users do not ``own'' the information for which they are allowed access. For these organizations, the corporation or agency is the actual ``owner'' of system objects as well as the programs that process it. Control is often based on employee functions rather than data ownership.

Access control decisions are often determined by the roles individual users take on as part of an organization. This includes the specification of duties, responsibilities, and qualifications. For example, the roles an individual associated with a hospital can assume include doctor, nurse, clinician, and pharmacist. Roles in a bank include teller, loan officer, and accountant. Roles can also apply to military systems; for example, target analyst, situation analyst, and traffic analyst are common roles in tactical systems. A role based access control (RBAC) policy bases access control decisions on the functions a user is allowed to perform within an organization. The users cannot pass access permissions on to other users at their discretion. This is a fundamental difference between RBAC and DAC.

Security objectives often support a higher level organizational policy, such as maintaining and enforcing the ethics associated with a judge's chambers, or the laws and respect for privacy associated with the diagnosis of ailments, treatment of disease, and the administering of medicine with a hospital. To support such policies, a capability to centrally control and maintain access rights is required. The security administrator is responsible for enforcing policy and represents the organization.

The determination of membership and the allocation of transactions to a role is not so much in accordance with discretionary decisions on the part of a system administrator, but rather in compliance with organization-specific protection guidelines. These policies are derived from existing laws, ethics, regulations, or generally accepted practices. These policies are non-discretionary in the sense that they are unavoidably imposed on all users. For example, a doctor can be provided with the transaction to prescribe medicine, but does not possess the authority to pass that transaction on to a nurse.

RBAC is in fact a form of mandatory access control, but it is not based on multilevel security requirements. As defined in the TCSEC, MAC is a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e. clearance) of subjects to access information of such sensitivity.[4]

Role based access control, in many applications (e.g. [9], [10], [11]) is concerned more with access to functions and information than strictly with access to information.
The act of granting membership and specifying transactions for a role is loosely analogous to the process of clearing users (granting membership) and the labeling (associate operational sensitivities) of objects within the DoD. The military policy is with respect

to one type of capability: who can read what information. For these systems the unauthorized flow of information from a high level to a low level is the principal concern. As such, constraints on both reads and writes are in support of that rule. Within a role based system, the principal concern is protecting the integrity of information: “who can perform what acts on what information.”

A role can be thought of as a set of transactions that a user or set of users can perform within the context of an organization. Transactions are allocated to roles by a system administrator. Such transactions include the ability for a doctor to enter a diagnosis, prescribe medication, and add an entry to (not simply modify) a record of treatments performed on a patient. The role of a pharmacist includes the transactions to dispense but not prescribe prescription drugs.
Membership in a role is also granted and revoked by a system administrator.

Roles are group oriented. For each role, a set of transactions allocated the role is maintained. A transaction can be thought of as a transformation procedure [1] (a program or portion of a program) plus a set of associated data items. In addition, each role has an associated set of individual members. As a result, RBACs provide a means of naming and describing many-to-many relationships between individuals and rights. Figure 1 depicts the relationships between individual users, roles/groups, transformation procedures, and system objects.

The term transaction is used in this paper as a convenience to refer to a binding of transformation procedure and data storage access. This is not unlike conventional usage of the term in commercial systems. For example, a savings deposit transaction is a procedure that updates a savings database and transaction file. A transaction may also be quite general, e.g. “read savings file”. Note however, that “read” is not a transaction in the sense used here, because the read is not bound to a particular data item, as “read savings file” is.

The importance of control over transactions, as opposed to simple read and write access, can be seen by considering typical banking transactions. Tellers may execute a savings deposit transaction, requiring read and write access to specific fields within a savings file and a transaction log file. An accounting supervisor may be able to execute correction transactions, requiring exactly the same read and write access to the same files as the teller. The difference is the process executed and the values written to the transaction log file.

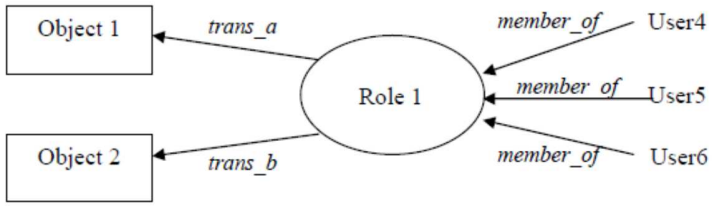


Figure 1: Role relationships

The applicability of RBAC to commercial systems is apparent from its widespread use. Baldwin [9] describes a database system using roles to control access. Nash and Poland [10] discuss the application of role based access control to cryptographic authentication devices commonly used in the banking industry. Working with industry groups, the National Institute of Standards and Technology has developed a proposed standard, “Security Requirements for Cryptographic Modules”' (Federal Information Processing Standard 140-1)[11] that will require support for access control and administration through roles. To date, these role based systems have been developed by a variety of organizations, with no commonly agreed upon definition or recognition in formal standards. *Role based access controls described in this paper address security primarily for application-level systems, as opposed to general purpose operating systems*.

3 Formal Description of RBAC

To clarify the notions presented in the previous section, we give a simple formal description, in terms of sets and relations, of role based access control. No particular implementation mechanism is implied.

For each subject, the active role is the one that the subject is currently using:

$$AR(s: subject) = \{the\ active\ role\ for\ subject\ s\}$$

Each subject may be authorized to perform one or more roles:

$$RA(s: subject) = \{authorized\ roles\ for\ subject\ s\}$$

Each role may be authorized to perform one or more transactions:

$$TA(r: role) = \{transactions\ authorized\ for\ role\ r\}$$

Subjects may execute transactions. The predicate $exec(s,t)$ is true if subject s can execute transaction t at the current time, otherwise it is false:

$$exec(s: subject, t: tran) = true\ iff\ subject\ s\ can\ execute\ transaction\ t.$$

Three basic rules are required:

1. Role assignment: A subject can execute a transaction only if the subject has selected or been assigned a role:

$$\forall s: subject, t: tran, \big(exec(s,t)\big) \Rightarrow AR(s) \neq \emptyset)$$

The identification and authentication process (e.g. login) is not considered a transaction. All other user activities on the system are conducted through transactions. Thus all active users are required to have some active role.

2. Role authorization: A subject's active role must be authorized for the subject:

$$\forall s: subject, (AR(s) \subseteq RA(s))$$

With (1) above, this rule ensures that users can take on only roles for which they are authorized.

3. Transaction authorization: A subject can execute a transaction only if the transaction is authorized for the subject's active role:

$$\forall\ s: subject, t: tran, (exec(s,t) \Rightarrow t\ \epsilon\ TA\big(AR(s)\big))$$

With (1) and (2), this rule ensures that users can execute only transactions for which they are authorized. Note that, because the conditional is “only if”, this rule allows the possibility that additional restrictions may be placed on transaction execution. That is, the rule does not guarantee a transaction to be executable just because it is in *TA(AR(s))*, the set of transactions potentially executable by the subject's active role. For example, a trainee for a supervisory role may be assigned the role of “Supervisor”, but have restrictions applied to his or her user role that limit accessible transactions to a subset of those normally allowed for the Supervisor role.

In the preceding discussion, a transaction has been defined as a transformation procedure, plus a set of data items accessed by the transformation procedure. Access control in the rules above does not require any checks on the user's right to access a data object, or on the transformation procedure's right to access a data item, since the data accesses are built into the transaction. Security issues are addressed by binding operations and data into a transaction at design time, such as when privacy issues are addressed in an insurance query transaction.

It is also possible to redefine the meaning of "transaction" in the above rules to refer only to the transformation procedure, without including a binding to objects. This would require a fourth rule to enforce control over the modes in which users can access objects through transaction programs. For example, a fourth rule such as

4. $\forall s: subject, t: tran, o: object, (exec(s,t)) \Rightarrow access(AR(s), t, o, x))$

could be defined using a transaction (redefined to transformation procedure) to object access function *access(r, i, o, x)* which indicates if it is permissible for a subject in role *r* to access object *o* in mode *x* using transaction *t*, where *x* is taken from some set of modes such as read, write, append. Note that the Clark-Wilson access control triple could be implemented by letting the modes *x* be the access modes required by transaction *t*, and having a one-to-one relationship between subjects and roles. RBAC, as presented in this paper, thus includes Clark and Wilson access control as a special case.

Use of this fourth rule might be appropriate, for example, in a hospital setting. A doctor could be provided with read/write access to a prescription file, while the hospital pharmacist might have only read access. (Recall that use of the first three rules alone requires binding the transaction program t and data objects that t can access, and only controls access to the transactions.) This alternative approach using the fourth rule might be helpful in enforcing confidentiality requirements.

Another use of RBAC is to support integrity. Integrity has been defined in a variety of ways, but one aspect [8] of integrity is a requirement that data and processes be modified only in authorized ways by authorized users. This seems to be a reasonable security objective for many real systems, and RBAC should be applicable to such systems.

In general, the problem of determining whether data have been modified only in authorized ways can be as complex as the transaction that did the modification. For this reason, the practical approach is for transactions to be certified and trusted. If transactions must be trusted then access control can be incorporated directly into each transaction. Requiring the system to control access of transaction programs to objects through the access function used in rule (4) might then be a useful form of redundancy, but it could involve significant overhead for a limited benefit in enforcing integrity requirements.

Therefore, inclusion of a transaction to object access control function in RBAC would be useful in some, but not all applications.

4 Centrally Administering Security Using RBAC

RBAC is flexible in that it can take on organizational characteristics in terms of policy and structure. One of RBAC's greatest virtues is the administrative capabilities it supports.

Once the transactions of a Role are established within a system, these transactions tend to remain relatively constant or change slowly over time. The administrative task consists of granting and revoking membership to the set of specified named roles within the system.

When a new person enters the organization, the administrator simply grants membership to an existing role. When a person's function changes within the organization, the user membership to his existing roles can be easily deleted and new ones granted. Finally, when a person leaves the organization, all memberships to all Roles are deleted. For an organization that experiences a large turnover of personnel, a role-based security policy is the only logical choice.

In addition, roles can be composed of roles. For example, a Healer within a hospital can be composed of the roles Healer, Intern, and Doctor. Figure 2 depicts an example of such a relationship.

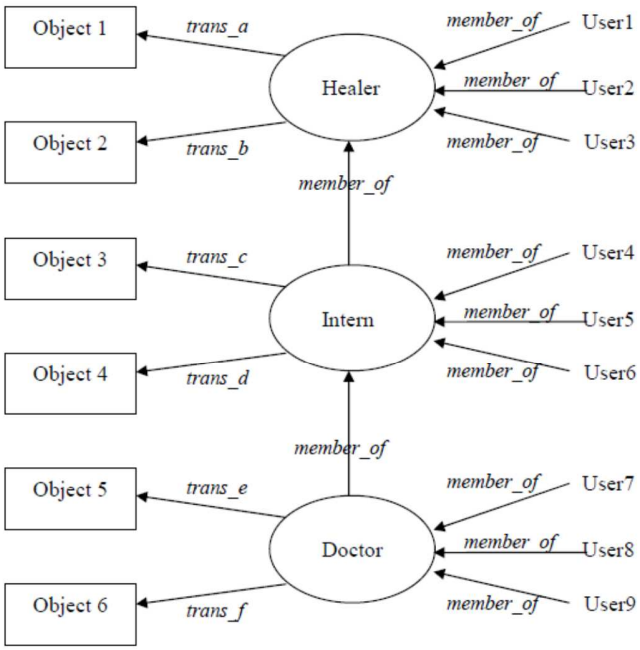


Figure 2: Multi-Role relationships

By granting membership to the Role Doctor, it implies access to all transactions defined by Intern and Healer, as well as those of a Doctor. On the other hand, by granting membership to the Intern role, this implies transactions of the Intern and Healer not the Doctor. However, by granting membership to the Healer role, this only allows access to those resources allowed under the role Healer.

5 Principle of Least Privilege

The principle of least privilege has been described as important for meeting integrity objectives. [8] The principle of least privilege requires that a user be given no more privilege than necessary to perform a job. Ensuring least privilege requires identifying what the user's job is, determining the minimum set of privileges required to perform that job, and restricting the user to a domain with those privileges and nothing more. By denying to subjects transactions that are not necessary for the performance of their duties, those denied privileges cannot be used to circumvent the organizational security policy.

Although the concept of least privilege currently exists within the context of the TCSEC, requirements restrict those privileges of the system administrator. Through the use of RBAC, enforced minimum privileges for general system users can be easily achieved.

6 Separation of Duties

RBAC mechanisms can be used by a system administrator in enforcing a policy of separation of duties. Separation of duties is considered valuable in deterring fraud since fraud can occur if an opportunity exists for collaboration between various job related capabilities. Separation of duty requires that for particular sets of transactions, no single individual be allowed to execute all transactions within the set. The most commonly used examples are the separate transactions needed to initiate a payment and to authorize a payment. No

single individual should be capable of executing both transactions. Separation of duty is an important consideration in real systems. [1], [12], [13], [14]

The sets in question will vary depending on the application. In real situations, only certain transactions need to be restricted under separation of duty requirements. For example, we would expect a transaction for “authorize payment” to be restricted, but a transaction “submit suggestion to administrator” would not be.

Separation of duty can be either static or dynamic. Compliance with static separation requirements can be determined simply by the assignment of individuals to roles and allocation of transactions to roles. The more difficult case is dynamic separation of duty where compliance with requirements can only be determined during system operation.

The objective behind dynamic separation of duty is to allow more flexibility in operations. Consider the case of initiating and authorizing payments. A static policy could require that no individual who can serve as payment initiator could also serve as payment authorizer. This could be implemented by ensuring that no one who can perform the initiator role could also perform the authorizer role. Such a policy may be too rigid for commercial use, making the cost of security greater than the loss that might be expected without the security. More flexibility could be allowed by a dynamic policy that allows the same individual to take on both initiator and authorizer roles, with the exception that no one could authorize payments that he or she had initiated. The static policy could be implemented by checking only roles of users; for the dynamic case, the system must use both role and user ID in checking access to transactions.

Separation of duty is necessarily determined by conditions external to the computer system. The Clark-Wilson [1] scheme includes the requirement that the system maintain the separation of duty requirement expressed in the access control triples. Enforcement is on a per-user basis, using the user ID from the access control triple. As discussed above, user functions can be conveniently separated by role, since many users in an organization typically perform the same function and have the same access rights on TPs and data. Allocating access rights according to role is also helpful in defining separation of duty in a way that can be enforced by the system.

7 Summary and Conclusions

In many organizations in industry and civilian government, the end users do not “own” the information for which they are allowed access. For these organizations, the corporation or agency is the actual “owner” of system objects, and discretionary access control may not be appropriate. Role-Based Access Control (RBAC) is a nondiscretionary access control mechanism which allows and promotes the central administration of an organizational specific security policy.

Access control decisions are often based on the roles individual users take on as part of an organization. A role specifies a set of transactions that a user or set of users can perform within the context of an organization. RBAC provide a means of naming and describing relationships between individuals and rights, providing a method of meeting the secure processing needs of many commercial and civilian government organizations. Various forms of role based access control have been described and some are used in commercial systems today, but there is no commonly accepted definition or formal standards encompassing RBAC. As such, evaluation and testing programs for these systems have not been established as they have for systems conforming to the Trusted Computer Security Evaluation Criteria. This paper proposed a definition of The requirements and access control rules for RBAC proposed in this paper could be used as the basis for a common definition of access controls based on user roles.

References

1. D.D. Clark and D.R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In IEEE Symposium on Computer Security and Privacy, April 1987.
2. Computers at Risk. National Research Council, National Academy Press, 1991.
3. Minimum Security Functionality Requirements for Multi-User Operating Systems (draft). Computer Systems Laboratory, NIST, January 27 1992.
4. Trusted Computer Security Evaluation Criteria, DOD 5200.28-STD. Department of Defense, 1985.
5. Z.G. Ruthberg and W.T. Polk, Editors. Report of the Invitational Workshop on Data Integrity. SP 500-168. Natl. Inst. of Stds. and Technology, 1989.

6. S.W. Katzke and Z.G. Ruthberg, Editors. Report of the Invitational Workshop on Integrity Policy in Computer Information Systems. SP 500-160. Natl. Inst. of Stds. and Technology, 1987.
7. J.E. Roskos, S.R. Welke, J.M. Boone, and T. Mayfield. Integrity in Tactical and Embedded Systems. Institute for Defense Analyses, HQ 89-034883/1, October 1989.

8. Integrity in Automated Information Systems. National Computer Security, Center, September 1991.
9. R.W. Baldwin. Naming and Grouping Privileges to Simplify Security Management in Large Databases. In IEEE Symposium on Computer Security and Privacy, 1990.
10. K.R. Poland M.J. Nash. Some Conundrums Concerning Separation of Duty. In IEEE Symposium on Computer Security and Privacy, 1990.
11. Security Requirements for Cryptographic Modules. Federal Information Processing Standard 140-1, National Institute of Standards and Technology, 1992.
12. W.R. Shockley. Implementing the Clark/Wilson Integrity Policy Using Current Technology. In Proceedings of 11th National Computer Security Conference, October 1988.
13. A R. Sandhu. Transaction Control Expressions for Separation of Duties. In Fourth Aerospace Computer Security Applications Conference, December 1988.
14. S. Wiseman P. Terry. A 'New' Security Policy Model. In IEEE Symposium on Computer Security and Privacy, May 1989.

5.3 BPM Meets SOA (Brooke, 2010)

Abstract

Service-Oriented Architecture (SOA) provides a framework for the design of business processes that manage shared capabilities. Shared capabilities can be engaged in multiple lines of business and achieve both economies of scale through consolidation and enterprise agility through the ability to configure new lines of business using existing capabilities. Capabilities are managed as service units that include the skills and resources to deliver well-defined services. In a transformed enterprise, service units engaged by each line of business become participants in a value chains that form the basis for optimization of operations and delivery of customer value.

1 Introduction

Service-Oriented Architecture (SOA) describes an approach to integration of systems where business capabilities are accessed across organizational boundaries. Though the development of SOA has been driven by the development of supporting technology, SOA should not be viewed as a technical discipline, but rather an approach to designing enterprises, including extended enterprises that involve multiple, collaborating companies, agencies, or institutions. SOA provides a framework for the design of business processes to promote consolidation of redundant operations and an improved ability to adapt to changing business needs. At the same time, it provides alignment of business processes with the organization structure, shared capabilities, and delivery of customer value.

SOA emerged from the ability to engage automated business services electronically, over the Internet. Technical standards and the Internet enable ad hoc interactions with systems implemented using diverse technologies. Loose coupling is achieved through message exchanges where the implementation of a service is hidden from the consumer and the service is designed to be used by a diverse community of consumers. The message exchanges are driven by the internal business processes of the service consumers and providers.

The full potential of SOA is realized when it is applied as an architecture for business design. The enterprise becomes a composition of capabilities that can be employed in a variety of business contexts. As such, SOA provides the basis for structuring and integrating business processes. The result of applying this architecture will be an enterprise that is more efficient and flexible – an agile enterprise. The agile enterprise is designed for change and optimization through specialization and sharing of capabilities.

Traditionally, the design of enterprises has been an art guided by experience, iterative improvements, and survival of the fittest. Conventional organization structures are a product of this evolution. Each line of business is typically developed as a separate organization, and business processes are optimized for each line of business. Optimization will depend on the current state of the ecosystem and technology, so the optimum will change over time. The entanglement of business processes and organizations delays needed changes as expensive and disruptive undertakings and results in suboptimal operations. Change is further encumbered by the embedding of business processes in computer applications. In today's rapidly changing world, an enterprise must be able to continuously adapt and optimize its operations.

This adaptation and optimization cannot be confined by traditional organizational silos, and optimization should be from an enterprise perspective to maximize economies of scale. SOA enables rapid reconfiguration and adaptation along with enterprise-level optimization.

In this chapter, we begin by examining the definition of SOA from a business perspective and its relationship to the design of business processes, including a case management business process model. Next, we consider value chain modeling as an important new perspective on enterprise design and optimization. Finally, we examine how SOA and BPM support additional enterprise optimization and agility, with a brief look at the challenge of enterprise transformation. The illustrations and many of the concepts presented here are from the book, Building the Agile Enterprise with SOA, BPM and MBM (Cummins 2009).

2 Definition of SOA

Based on the OASIS (Organization for Advancement of Structured Information Systems) SOA Reference Model (OASIS http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm), SOA involves the delivery of services from shared capabilities accessed across organizational boundaries. A simple example of such a service is access to information such as stock quotes from another organization through a request over the Internet. This is the typical web services model. However, the value delivered by a service may be more complex information or a tangible product, the interaction may be more complex than a simple request-response, and the requester may itself be an organization providing value as a service to yet another organization.

A SOA enterprise is designed as a composition of services. For example, a customer order is processed by an order processing service, which in turn uses an order fulfillment service to pack and package products and uses a transportation service to deliver the products to the original requester. Each service provider applies its capability in response to a request from another organization. Together, they satisfy the customer order. At the same time, because they provide services in a defined way, they may each respond to requests from a variety of service consumers.

A service provider must have a well-defined interface to receive requests and provide results so that it can serve a variety of service consumers representing different business contexts.

The transportation carrier can deliver goods for a variety of suppliers. Because it provides its capability to multiple suppliers, it can achieve economies of scale in the utilization of its resources and thus provide the service at a lower cost than each supplier could achieve on its own. It can also maintain a capacity that enables it to respond more quickly than a dedicated transportation capability. And by specializing, it can develop and maintain special skills and equipment that improve the quality of the service. These are the fundamental benefits of SOA: speed, lower cost, and quality.

Figure 1 illustrates the impact of SOA on a typical, large enterprise. Different lines of business operate in separate organizational silos, each optimized for its particular line of business as depicted in Fig. 1a. The boxes represent different capabilities needed to perform the line of business. The capabilities are typically tightly integrated so that the boundaries and relationships are not nearly as clear as suggested by the diagram.

When SOA principles are applied, similar capabilities from the different lines of business are combined as depicted in Fig. 1b. Each of the consolidated capabilities has an opportunity to achieve economies of scale that can improve speed, cost, and quality. I will refer to each of the organizations providing a shared capability as a service unit; this distinguishes it from its service, which is the thing of value it provides through application of its capability. In a fully transformed enterprise, all capabilities are implemented as service units that respond to requests from other service units or customers.

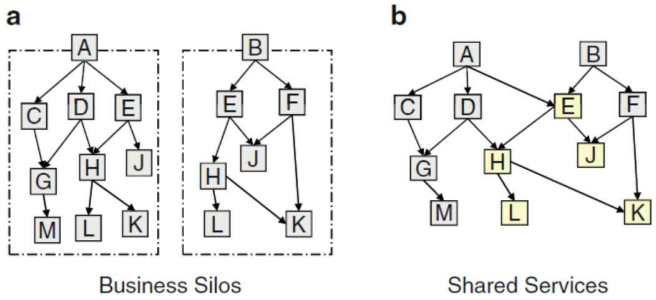


Fig. 1 From silos to services [figure originally published in Cummins (2009, p. 14 and p. 15)]

Of course, there is a trade-off. Each of the lines of business has less control over the shared service units. Each service unit potentially has multiple consumers to satisfy. The organization that owns a service unit must take responsibility for meeting the needs of all of its consumers. At the same time, economies of scale should enable the shared service unit to achieve better results than capabilities dedicated to each line of business.

Each line of business incorporates the services it needs to deliver value to its customers.

3 BPM in SOA

So where are the business processes in SOA? They are in the service units.
Many technical approaches to SOA position business processes above services, driving the use of services. While processes do use services, these approaches fail to comprehend that the business process that invokes a service is part of yet another service unit.

Figure 2 illustrates this relationship. Service unit A accepts two kinds of requests as indicated by the arrows entering from the top. Each of these invokes a business process – business processes X and Y, respectively. These business processes engage computer applications and people to apply the capability of the service unit. Business process X delegates some of its service responsibility to service unit B, which provides a different capability, potentially shared by other parts of the enterprise.

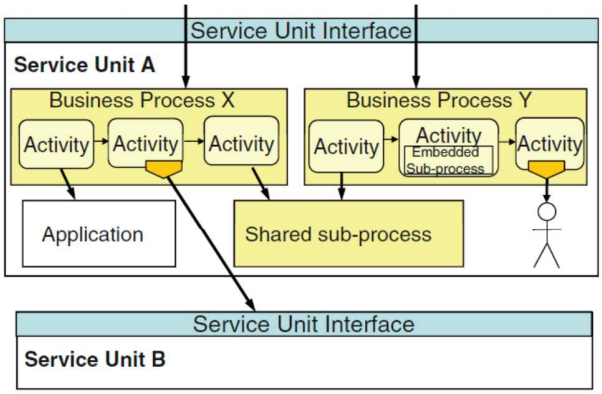


Fig. 2 Business processes within service units [figure originally published in Cummins (2009, p. 97)]

Each of these service units has capabilities provided by business processes, people, applications, facilities, intellectual capital, and other resources such as tools and materials. Each also has a responsibility to maintain, improve, and adapt the capability to changing business circumstances.

The business processes start and end within the scope of the associated service unit. Thus, the business processes are "owned" by the service unit and can be adapted and refined to improve the operation of the service unit without involving other organizations. The discretion of the service unit manager is restricted by the interface specifications of other services it uses as well as the interface specifications for the services it provides. There also may be resource constraints if some resources are shared with other service units to improve resource utilization. Later, we will discuss some of these optimization issues further.

Note that the business processes need not be automated, but there must be infrastructure that supports the integration with other service units. From the typical technology view of SOA, the rapid integration should be automated through web services technology, but integration could be through other forms of message exchange, including email or paper. Automation using XML (W3C http://www.w3.org/XML/) message structures (W3C http://www.w3.org/TR/2003/REC-soap12-part1-20030624/) and electronic message exchange protocols will be preferred for speed, reliability, and flexibility. Conventional business applications have embedded business processes. These embedded business processes represent business needs and technical capabilities at a point in time when the applications were developed. They are obscured from view of the process owners, and they are difficult to adapt to changing business needs.

While it is not essential for SOA, an agile enterprise should endeavor to remove its business processes from its applications, express them in a standard form such as BPMN (Business Process Modeling Notation)[1] and automate

them with a BPMS (Business Process Management System). Note that BPMN 2.0 is currently under development and will define extended modeling capabilities, robust execution semantics, and model portability standards (OMG http://www.omg.org/spec/BPMN/2.0/). As a result, the integration of business activities occurs through the integration of business processes rather than the integration of business applications. SOA essentially provides a business process architecture. This business process integration must be through well-defined service interfaces that make the associated services accessible to a variety of consumers. The information technology infrastructure of the enterprise, and the Internet, provide the vehicle for exchange of messages both between service units within the enterprise and with services provided to customers or by outside suppliers, which may include outsourcing of accounting, human resource, and information technology services.

The business processes may be automated with different Business Process Management systems (BPMS). Integration through a messaging infrastructure insulates service units from differences in the implementation technologies of their consumers and providers.

Standards are nevertheless required for the format and content of messages exchanged. It is desirable that the format of all message types represent agreements between participants, but integration facilities can be used to transform messages for compatibility if the content is equivalent. At the same time, the meanings of the data elements must be consistent to be properly interpreted even if they require conversion. In general, data exchanged between services within an enterprise should conform to an enterprise logical data model so that whether or not messages are translated, all services "speak" fundamentally the same language. If there is not a consistent enterprise logical data model, the ability to share and reconfigure the use of services will be significantly impaired.

There may be many different message types involved in a consumer–provider relationship. Interactions may involve more complex protocols than a simple request-response. These protocols must be specified with a choreography. BPMN 2.0 (currently under development) (OMG http://www.omg.org/techprocess/meetings/schedule/BPMN_2.0_RFP.html) presents choreography so that the exchanges between service units can be explicitly defined independent of the internal business processes by which services are performed and consumed. A choreography and the associated message types will generally be associated with a type of service. This enables consumers to easily engage alternative services of the same type.

More complex interactions will be required for services envisioned in service science (Lusch et al. 2008) where a service may be expected to adapt its resources to the needs of the consumer rather than simply responding to a request. This adaptation is more likely to require a negotiation of requirements and value exchange. A case management process is typically driven by such evolving consumer requirements.

4 Case Management

In many cases, the interchanges between consumers, providers, and potentially other participants extend over long periods of time, and the actions taken by participants depend on changing internal and environmental factors. In such relationships, the actions of a participant may not be described as a repeatable sequence of activities and decisions but rather as actions to be taken based on changing circumstances. Such processes have been described as case management (de Man http://www.bptrends.com/publicationfiles/01%2D09%2DART%2D%20Case%20Management%2D1%2DDeMan%2E%20doc%2D%2Dfinal%2Epdf). Case management processes complement SOA.

Case management generally revolves around management of services related to a particular entity, often a person. A case file is the focus of related actions, and various actions are taken as the status of the entity or surrounding circumstances change.

For example, a case is created for a hospital patient when admitted. As the patient is examined, tests may be performed and treatments administered. Various tests and treatments may be determined as the condition of the patient evolves, and the case file tracks the patient status and associated actions. The case file is likely retained by the hospital and reactivated if and when the patient returns. In conventional business processes, the process performs a number of predefined activities to achieve a desired result. In case management, there may be many actions that

could be performed, but the selection of actions and the sequence in which they are performed may be different for each case.

We may characterize the actions taken as services rendered. The case management service manages the case file and the performance of relevant services. The performance of services may be driven by an expert, by rules, by a schedule, or by a combination of rules, schedule, and expertise.

There are a variety of circumstances where a case management model may be appropriate. In addition to the hospital patient, an employee record could be viewed as a case file. The employee case may drive benefits, payroll, promotions, and other actions as the employee's status changes over time. Court cases or welfare cases are other well-known examples involving people.

Maintenance of a machine could be managed as a case. Preventive maintenance should occur on a schedule. Periodic examinations may reveal deterioration requiring repairs. A failure will require diagnosis and repair. As the machine ages and repair costs escalate, the repair history may provide a basis for replacement.

An automobile repair history could be managed as a case, but more often, such a case begins when the automobile is brought in for repair and is completed when the automobile is returned to the owner. At the same time, there could be a lifetime case file maintained by the owner and individual cases for incidents of repair.

Case management processes tend to be long-running. An automobile repair case may endure for a few hours, days, or weeks, but a machinery maintenance case may go on for years, and a hospital or employment record may be maintained for decades.

Projects for development or construction could be viewed as case management processes. Regardless of how well the project plan is prepared, there will be changes in the sequence of actions and the scheduling of resources, and there will be rework. At the same time, many of the actions may be predictable sequences of activities that can be described with conventional process models. In all these examples, the case file is the focal point for determination of actions to be taken. Similar actions may be taken for similar cases, but the set of actions taken and the sequence in which they occur will vary. The actions typically involve specialized capabilities that are provided as services in a SOA.

The case management process pattern is not a good fit for conventional business process modeling tools. An OMG (Object Management Group) initiative is being defined to develop modeling specifications for case management.

5 Value Chains

Conventional business process models as well as case management models define mechanisms of control over when work is performed, but the flow of control does not always correspond to the flow of work products that produces customer value.

This flow of work products can be described as a value chain, and it provides the basis for understanding the impact of participating service units on a line of business.

SOA changes the structure and dynamics of the business processes and organization. A business process can no longer be designed to define a single stream of activities by different organizations as they contribute value toward an end product. In a SOA, organizations may contribute to many different lines of business at different points in the value creation process. A process for a line of business must engage a variety of services that are shared with other lines of business, and those services may engage other services to support their efforts. Business process improvement must comprehend the various ways services are engaged to produce customer value for all lines of business.

Organization structures must also change. Traditionally, a line of business could create an organization and control all the capabilities needed to deliver its value to its customers. With SOA, the line of business manager must give up control of shared services. Figure 1b, above, illustrated this. The shared services must not be optimized for the

particular needs of one of its product-line consumers, but must optimize its operation for all of its consumers – essentially it must be optimized for the enterprise with an understanding of its impact on multiple lines of business.

Understanding the roles of service units in the creation of customer value requires a different perspective on the relationships between service units. A value chain provides this perspective. The concept of a value chain was first introduced by Michael Porter in his 1985 book (Porter 1985). While the initial concept aligns most easily to manufacturing enterprises, it has also been applied to other sectors (Stabell and Fjeldstad 1998).

The value chain enabled top management to focus on the delivery of value to the customer and evaluate the enterprise capabilities in that context. A primary value chain involves capabilities that are directly involved in the delivery of customer value. There are other support services that are involved in the management and effective operation of the enterprise such as accounting and human resource management.

The primary value chain typically focuses on a half-dozen high-level business functions and a hierarchical breakdown of the capabilities that contribute to producing customer value. Here, we expand that high-level executive management concept to a level of detail where the capabilities that contribute value are service units– capabilities that may be shared across multiple lines of business.

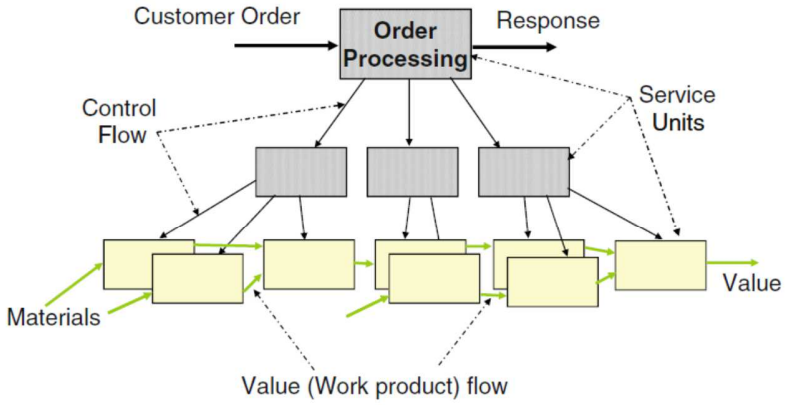


Fig. 3 Relationship of control flow and value flow

These value chain models are often described as process models, but there is an important difference between business processes and value chains, particularly in a SOA.
Figure 3 illustrates the relationship between the flow of control defined by business processes and the flow of value through the value chain. The business processes define how service units produce results, and the value chain defines the flow of value across service units and organizations to achieve customer value.

For example, in a manufacturing operation, business processes define production scheduling. Parts are produced in batches. The scheduling process optimizes the cost of setup against the cost of delayed production and inventory retention. The value chain describes the flow of parts and assemblies between production operations, i.e., capabilities, to produce customer value. The flow of control to request production and produce schedules is quite different from the flow of value between capabilities. A new schedule might be requested from a scheduling service on a daily basis, while the movement of parts between departments will likely occur as batches are completed. In the manufacturing operation, some batches of parts may meet the needs of multiple products and different customer orders, and multiple users of parts may draw on a single inventory. So the movement and consumption of parts, in other words, the value chain flow, is quite removed from requests for the daily production schedule.

The value chain is not directly concerned with orders and batching, but focuses on the capabilities needed to produce a product. The value chain will focus on the unit cost of production and time to deliver of each of the capabilities and thus the cost and timeliness for each product delivered to a customer. The scheduling process will have a significant impact on the production cost and time of each unit. A value chain can be expressed as a

capability dependency network. Figure 4 depicts such a dependency network. The value of the end product depends on production and delivery capabilities along with warranty repair and preventive maintenance.

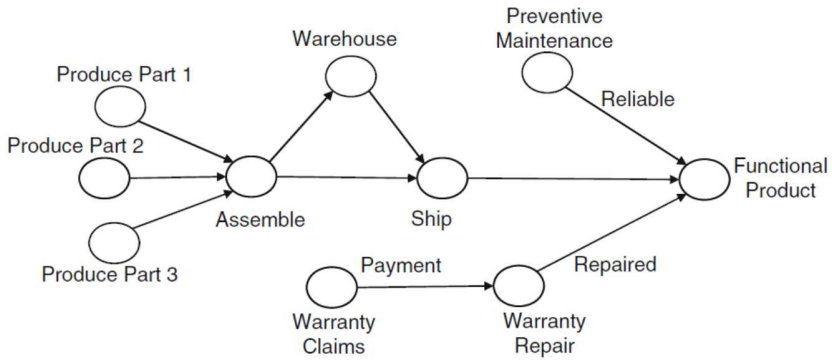


Fig. 4 A value chain dependency network [figure originally published in Cummins (2009, p. 66)]

For delivery of the finished product, shipping depends on the availability of the product from a warehouse or the assembly operation, and the assembly operation depends on the production of parts.

A detailed value chain can be used to analyze a conventional business as well as a service-oriented business. In both cases, the analysis focuses on capabilities and their contributions to customer value. However, in an SOA, detailed value chain analysis is both more important and more meaningful. In a SOA, the service unit capabilities are shared by multiple lines of business. In order to understand the cost and timeliness of each line of business, it is necessary to consider the impact of the service units that contribute to that line of business. In a conventional organization, all the capabilities in that silo contribute to that line of business and only that line of business. In addition, the clear boundaries of service units clarify the costs and the contribution of each service unit.

Product development is typically included in the primary value chain. Unlike other value chain capabilities, its capability is not applied to each unit of production but defines how the production and delivery capabilities will produce customer value. For example, in a manufacturing enterprise, product development will define the design of parts and assemblies, special tools, and details of production processes typically utilizing existing general capabilities such as standard tools, machines, and personnel.

Product development has a value chain to produce a product delivery capability. The line of business is effectively an internal customer for the value produced by product development – the ability to produce the end customer product. The process by which product development delivers value can be characterized as a case management process as described above, where various actions are performed as various aspects of the product delivery capability are developed.

Costs of production capabilities can be allocated to individual units of production as discussed earlier. On the other hand, costs of product development are associated with the full product lifecycle. The full cost of a product includes development and production, so the line of business must allocate the cost of development to units of production based on a projected volume of business.

An enterprise includes other capabilities that are not directly involved in the delivery of value to a customer. These capabilities provide value to internal customers and are characterized as support services. Accounting, human resource management, procurement, and information technology services are high-level capabilities provided as support services. These general capabilities provide a number of services to their internal customers, and they also have value chains.

The costs of these services are not incurred for each unit of production for end customers but are part of the overhead of the primary value chains that produce end customer value.

6 Enterprise Optimization

We have focused on optimization by consolidation of primary value chain capabilities across lines of business. The support services discussed above represent traditional consolidations of business activities to achieve efficiency and control of business operations. SOA also supports further optimization of enterprise operations.

Earlier, we observed that SOA supports consolidation of capabilities to achieve improved cost, quality, and timeliness of production. This consolidation removed capabilities from lines of business so that they could be shared, resulting in a loss of control of line of business managers over the capabilities they may have controlled directly to deliver customer value. This requires a change in thinking and approach to optimization and requires that optimization be considered from different perspectives: service unit optimization, line of business optimization, resource utilization optimization, and enterprise optimization.

The service unit boundary is key to the appropriate division of responsibility. The service interface, the specification of interactions, and performance requirements define what a service unit must do to meet the needs of its consumers.

The interface should be designed to preserve flexibility in the design of how the service unit actually performs the service. Value chains define what the services contribute to customer value while the organizational hierarchy defines control over the management of service unit resources. Overall enterprise optimization is based on the specification of services and organizational groupings that meet the needs of multiple lines of business and enable optimization of resource utilization. Changes to service specifications and the scope of service units must be evaluated at an enterprise level to consider the impact on multiple lines of business.

Each service unit is responsible for optimization of its internal operations. In addition to its internal processes, this may include development of specialized techniques and skills, workload balancing and scheduling, and hiring and firing of personnel. This localized optimization enables innovation and initiative throughout the enterprise.

A line of business manager must optimize his or her value chain. This involves consideration of the roles and capabilities of shared services. This may involve analysis of the ways services are used as well as potential changes to the services.

For example, in a manufacturing enterprise, a customer order could initiate the production of each of the parts that go into a finished assembly. This could result in significant delay in response to customer orders. On the other hand, many parts and potentially final assemblies may be produced in anticipation of customer orders. The line of business should consider the trade-off between rapid response to customer orders and the cost of carrying inventory as well as the potential obsolescence of parts and assemblies that remain in inventory when new models enter production. Note that an enterprise should be able to accommodate different tradeoffs for different lines of business.

A line of business manager has three alternatives to consider for improving needed capabilities: (1) advocate for changes to existing services, (2) establish and manage a duplicate capability optimized for the line of business, or (3) look for opportunities to outsource the capability to more accommodating service providers.

These options will be tempered at the enterprise level by consideration of the consequences to other lines of business. Changes to shared services may adversely affect the ability to meet the needs of other lines of business. A duplicate capability will reduce economies of scale, but that might be offset by economies of specialization.

Outsourcing of the capability reduces enterprise control over the capability and will reduce economies of scale for other lines of business unless they also outsource. If the capability is a source of competitive advantage, the advantage may be lost if the capability is outsourced.

Economies of scale also may be achieved through better resource utilization across multiple service units. This is typically achieved by organizationally grouping similar capabilities. For example, Fig. 5 illustrates a grouping of similar capabilities in the value chain introduced in Fig. 4. These groupings enable the larger organizations to balance workloads, share expensive resources, and achieve other forms of synergy across the service units in their group.

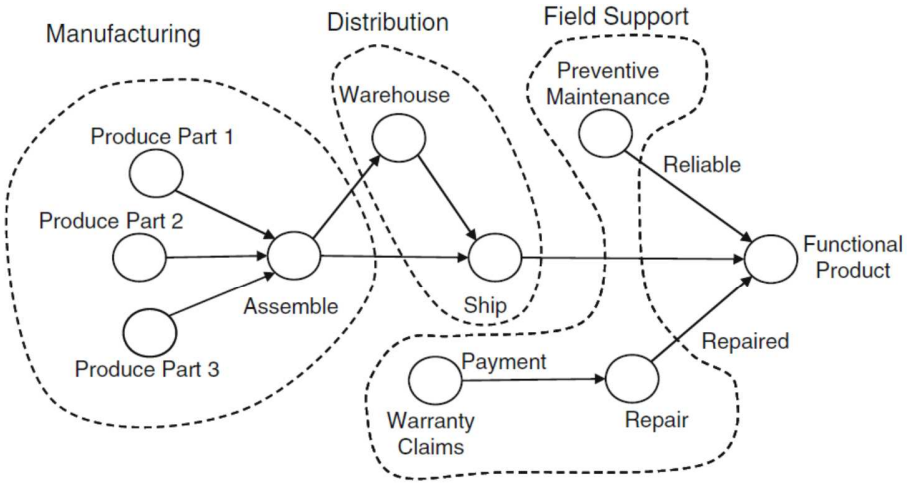


Fig. 5 Value chain abstraction for executive perspective [figure originally published in Cummins (2009, p. 67)]

This grouping supports the value chain abstractions characteristic of the Michael Porter value chain models. In the example, the manufacturing group has the opportunity to share personnel to balance workloads and schedule production in batches to reduce setup costs as discussed earlier. The distribution group has the opportunity to configure shipments for efficient delivery by holding some in the warehouse pending convenient load composition and routing. The field services group has the opportunity to share personnel and schedule repairs along with maintenance to reduce technician travel times.

Batching also affects customer value by delaying delivery, so line of business managers may play a role in determining the use of batching in a trade-off between cost and timeliness. In addition, the larger organizations may be able to justify consolidation of capabilities at a finer level of granularity. So, in the manufacturing environment, there will be separate service units responsible for machinery maintenance and materials management (movement and storage of materials), rather than each service unit performing these operations on their own. Of course, there must then be consideration of the organizational placement of these consolidated capabilities to balance economies of scale vs. responsiveness to the needs of the service units they support.

At the enterprise level, the consolidation and grouping of capabilities and specification of services must be considered from an overall enterprise perspective. Top management leadership will be required to separate sharable capabilities from the lines of business or other organizations that depend on them. In addition to efficiency and timeliness, some consolidations may be implemented to improve consistency and control. These are important aspects of services provided by finance, human resource management, procurement, and information technology services.

Ultimately, top management must mediate trade-offs between lines of business and alternative sources of capabilities. The enterprise becomes a form of matrix management as depicted in Fig. 6. The groupings of similar service units form functional organizations. The line of business value chains cut across the functional organizations where the intersections of the matrix are the shared services. The functional organizations are primarily concerned with meeting service performance requirements and management of resources. The line of business managers are focused on optimization of their value chains to deliver customer value with competitive advantage. Service unit managers focus on meeting their service specification requirements, optimizing their internal operations to improve cost, timeliness, and quality.

An important component of optimization is the costing of services. The cost of each unit of service should be determined with reasonable accuracy and incorporated into the costs of its consumers. This cost includes both the direct cost of operations and materials consumed to produce a unit of service, and the indirect costs, the overhead, of supporting services.

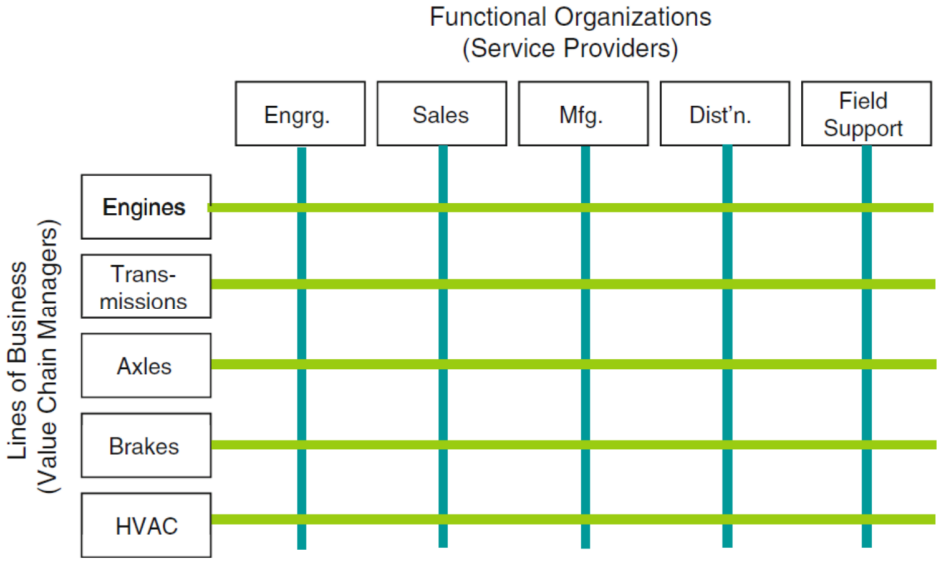


Fig. 6 SOA matrix management

When a service is shared by multiple lines of business, it is important that its costs be appropriately allocated so that the profitability of each product can be accurately determined. These costs will affect pricing and potentially decisions related to investments in growth or withdrawal from a market.

The cost and performance per unit of production of a service unit may depend on the specific service requirements. For example, the cost of shipping may depend on the size, weight, and potential hazards associated with the product, and it may also depend on the location of the customer. The cost of assembly will likely be significantly different for products of different designs, and there may be product differences between products of a single line of business.

Furthermore, cost and performance will be affected by workload. An increase in the number of units produced will likely increase the time to respond to a customer order, but the fixed cost of overhead allocated to each unit of production will be reduced. These factors will affect decisions about the scope of service unit operations as well as product pricing and marketing.

Costs are also important in considering improvements in operations. Line of business managers should be able to compare the cost of a service with equivalent services from alternative sources. Alternatively, several operating units may have resources dedicated to activities that could be consolidated for economies of scale.
If current costs are not well-understood, then it will be difficult to assess the business value of consolidation. Operations managers will be reluctant to relinquish control of capabilities that are important to their success. This applies, as well, where several different service units have a dedicated capability that might be consolidated into a shared service unit.

For example, a manufacturing department operates a group of presses and has a team of people that performs maintenance and repair on those presses. Other departments with different manufacturing capabilities also have their own maintenance and repair teams. Consolidation of the maintenance and repair capability could yield economies of scale that would reduce cost and improve response time for repairs. When a consolidated service unit is proposed, its costs must be computed on the same basis as current costs in order to support an objective evaluation.

If some departments are using production personnel to perform maintenance, or some overhead costs are not included for current maintenance and repair activities, then the cost of the current, dedicated capabilities will appear inappropriately low.

Costing must be applied to support services as well as primary services so that overhead costs can be properly applied. Because these costs apply to multiple units of production and multiple service units, each unit of production should carry an appropriate allocation. In addition, some costs of production may be different based on product mix

and volume. Consequently, effective costing is an art. Costs of units of service will be approximations, but should reasonably reflect actual costs in order to provide proper support for management decision-making.

Costing may also play a role in optimization by motivating changes of behavior of service consumers. The classic example is to charge more for a utility during peak hours so that consumers will make discretionary use of the utility during off-peak hours.

Such approaches are a topic of service science (Larson 2008) where the interaction of service characteristics and consumer behavior are studied to improve desired outcomes. For example, long lines at some voting facilities may alter the outcome of elections by discouraging some voters.

7 Enterprise Agility

SOA not only provides opportunities to improve cost, quality, and timeliness through consolidation, but the design of services for sharing by multiple lines of business improves enterprise agility (Cummins 2009). This benefit comes from (1) the ability to optimize processes within individual service units with minimal impact on the rest of the enterprise, (2) the ability to incorporate service units in new business processes as building blocks for new business endeavors, and (3) improved ability to adjust to changes in scale.

Since business processes begin and end within the scope of a service unit, and since service units are designed to function independent of the consumers they serve or the providers they use, the processes of a service unit can be changed locally as long as they conform to the defined service interfaces.

Top management should be able to consider a value chain for a new line of business that incorporates existing service unit capabilities and identifies capability gaps. The existing service units provide a jump-start in the ability to enter the new line of business and should provide reasonably reliable data on their contributions to cost, quality, and timeliness. The gaps then represent potential barriers to entry. This composition from existing capabilities will significantly reduce the time and cost to create a new line of business as compared to building a new organization, and the business case will be much more reliable.

Business changes often include changes in scale. An enterprise may need to scale up or down to adapt to changing market conditions. In a new market, the ability to scale with explosive growth may be the difference in maintaining market share in the long term. Consolidation of capabilities to serve multiple lines of business reduces the impact of changes in scale of individual lines of business.

In addition, SOA enhances the opportunity to outsource capabilities. Outsourcing may be used to off-load work in good times, and reduce costs in bad times. This is particularly true for commodity services such as accounting, human resource management, and information technology. A new enterprise may be unable to compete if the market for its product explodes because it cannot scale up its capacity to meet demand. An existing enterprise may be unable to scale down its capacity when market demand drops resulting in excessive product costs. An outsourcing service provider that realizes economies of scale across multiple clients, is impacted less by changes in workloads of individual clients, and accepts the burden of adjusting its capacity to the changing demands of its clients.

This strategy is not limited to dealing with changes in market demand but may be applied to transitional workloads such as significant transformation initiatives or new product development programs. It may also be applied where a needed capability can be immediately and more reliably obtained from an external provider.

Agility requires not only the ability to change, but the ability to recognize the need for change. The enterprise must incorporate business processes to sense threats and opportunities and take appropriate action. These threats and opportunities are signaled by changes in the enterprise ecosystem that have significance beyond the normal operation of the business. I characterize these as disruptive events.

Disruptive events may involve loss of personnel, supplier disruptions, inventions, changes in market demand, actions by competitors, new technology, natural disasters, cost of materials, and many other situations. At an enterprise level, these are the factors that are typically considered “influencers” that affect strengths, weaknesses, opportunities, or threats (SWOT). For the line of business manager, these events may affect the line of business

sales or the ability to meet customer expectations for cost, quality, and timeliness. For the service unit manager, these events will affect the ability of the service unit to perform according to defined levels of service or to compete with alternative sources of its capability.

Facilities to respond to disruptive events have been described as an event driven architecture (EDA). This can be viewed as an extension to SOA that detects and filters events to determine their relevance and initiates appropriate actions. There is specialized technology to detect and filter events from monitoring business and market changes. In addition, personnel throughout the enterprise may become aware of disruptive events that cannot be detected electronically. Event detection must be complemented by business processes that bring disruptive events to the attention of people who understand the implications. In some cases, there may be only operational impact to one or a few service units. In other cases, the disruptive event may affect the marketplace or the competitiveness of a product. Still other disruptive events may signal the need for strategic business change.

Business processes must support rapid resolution of the impact of disruptive events. This may be addressed by a service unit that receives events as inputs and properly directs them for appropriate action. This should probably be part of a broader enterprise intelligence capability. Over time, events that occur more frequently may be resolved automatically rather than requiring human judgment and planning. Case management automation will support the evolution from ad hoc to more predictable processes. This will further improve enterprise agility.

The full agility enabled by SOA requires a substantial transformation of the enterprise into a composition of service units.

8 Transformation to SOA

Transformation to SOA clearly has far-reaching effects on the operation of the enterprise. SOA essentially brings a new business paradigm – a new way of thinking about the organization and operation of the business. The associated changes cannot occur all at once but must be developed over a period of years.

HP Enterprise Services applies a SOA Maturity Model to guide this transformation. It provides assessment of the maturity of an enterprise in a number of dimensions, covering both business operations and information technology operations. The business must change to create and manage service units and implement new mechanisms of planning and governance. The information technology organization must change to provide appropriate infrastructure and supporting services for the integration of service units and the exploitation of technology in support of business operations.

The maturity model has five levels of maturity – level 1 is the status quo with minimal awareness of SOA. A transformation to level 2 brings a focus on specific opportunities for consolidation of capabilities with significant business value, primarily through economies of scale. These are approached as initial examples of service units that support multiple lines of business. The demonstration of value from these consolidations is the basis for development of a strategic plan for a more comprehensive development of services and supporting infrastructure described as level 3. Level 4 focuses on optimization with appropriate models, metrics, and management processes. Level 5 is the achievement of agility with formal processes for recognizing disruptive events, determining appropriate actions, and implementing the needed transformations. At level 5, change is an ongoing way of life.

9 Conclusion

SOA and BPM are complementary disciplines that, together, will yield valuable synergy and competitive advantage for those who exploit them. SOA provides an enterprise architecture discipline for organizing what is done, while BPM provides a design discipline for how it is done.

Like BPM, there have been aspects of SOA since the beginning of bureaucracies. The formation of accounting, human resource management, and procurement organizations represents the implementation of internal services, consolidating pervasive capabilities for economies of scale and control. The concept of a customer order is effectively a basic request for service as are many other internal business forms such as a purchase request, a material requisition, a personnel requisition, a payment order, and a work order.

The introduction of automated data processing led to the institutionalization of organization structures and business processes in the effort to achieve speed and reliability. In the early days of information systems, significant business changes occurred over decades. Information technology solutions optimized operations for the business and technology the way they were. They were not optimized for change. Today, significant business changes are under way all the time. Many of the principles of good enterprise design have been obscured by inflexible, technology oriented business solutions. Information technology must not only be removed as a barrier to change but must also support change.

SOA brings a new business paradigm, a new way of thinking about the organization of the enterprise to achieve economies of scale and agility. Integration and modeling technologies now enable management of the complexity and consistent application of SOA principles both within an enterprise and in relationships with customers and suppliers.

Competition has become intense as the marketplace has become global. Enterprises cannot afford to overlook opportunities for economies of scale. The ability to change has become a major factor in the survival of enterprises. Business Process Management systems (BPMS) provide improved ability to change and optimize how work is done, and SOA provides a discipline for management of capabilities to achieve economies of scale and adapt to new business challenges and opportunities. The disciplines, processes, and organizational approaches to managing change are still evolving.

Several things are clear. Strategic planning must become a continuous process. The design and management of the enterprise has become more complex and can no longer be delegated to individual lines of business. Top management cannot comprehend nor direct all of the changes necessary to optimize the operation of the enterprise – optimization must be managed at different levels of scope and authority.[2] This complexity must be managed through collaboration and the support of computer based models. Models must help managers understand problems and trade-offs, evaluate alternatives, formalize the design of the enterprise, and drive implementation and automation.

The models must support top management, line of business and service unit perspectives for optimization and rapid adaptation.

References

BPMN (Business Process Model and Notation) 2.0 RFP, Object Management Group (OMG). http://www.omg.org/spec/BPMN/2.0/
BPMN (Business Process Modeling Notation) specification, OMG (Object Management Group). http://www.omg.org/spec/BPMN/1.1/
Cummins FA (2009) Building the agile enterprise with SOA, BPM and MBM. Elsevier/Morgan Kaufman Publications, Amsterdam,
Boston de Man H. Case management: a review of modeling approaches. BPTrends. http://www.bptrends.
com/publicationfiles/01%2D09%2DART%2D%20Case%20Management%2D1%2DDeMan
%2E%20doc%2D%2Dfinal%2Epdf
Larson RC (2008) Service science: at the intersection of management, social and engineering sciences. IBM Syst J 47(1):41–51
Lusch RF, Vargo SL, Wessels G (2008) Toward a conceptual foundation for service science: contributions from service-dominant logic. IBM Syst J 47(1):5–14
Porter ME (1985) Competitive advantage: creating and sustaining superior performance. Free Press, New York
Stabell CB, Fjeldstad OD (1998) Configuring value for competitive advantage: on chains, shops and networks. Strat Manage J 19(5):413–417. http://www.agbuscenter.ifas.ufl.edu/5188/miscellaneous/configuring_value.pdf
SOA Reference Model, OASIS (Organization for the Advancement of Structured Information Systems). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm
SOAP version 1.2 Part 1: messaging framework specification, WorldWide Web Consortium (W3C). http://www.w3.org/TR/2003/REC-soap12-part1-20030624/
vom Brocke J, Rosemann M (2010) Handbook on business process management. Strategic alignment, governance, people and culture, vol 2. Springer, Heidelberg
XML (eXtensible Markup Language) specification, WorldWide Web Consortium (W3C). http://www.w3.org/XML/

6. Glosario de términos

.NET: Es un framework desarrollado por Microsoft que corre sobre la plataforma Windows (aunque ha sido portado a otras plataformas), este framework permite interoperabilidad entre distintos lenguajes.

Ajax: Asynchronous Javascript and XML, es una tecnología asíncrona que permite realizar pequeñas interacciones con el servidor desde el cliente en segundo plano, de esta manera no es necesario recargar la página web completa, mejorando la interacción, la usabilidad y la velocidad de las aplicaciones. Ajax hoy en día constituye la base para lo que serán las nuevas aplicaciones web y creaciones de frameworks como Angular.js o backbone.js.

Ajax-Forms: Es el nombre que le proporciona Intalio a las interfaces de usuario que pueden ser creadas a través de GI (General Interface).

Ant: Es una librería Java y una herramienta por línea de comandos que permite ejecutar procesos descritos en un archivo de construcción donde se especifican los objetivos a realizar. Es muy similar a make.

Apache ODE: Orchestration Director Engine, es un software capaz de ejecutar procesos de negocio escritos en lenguaje BPEL.

Artefacto: En maven, un artefacto es un proyecto de software con una versión específica.

Autenticación: Es el proceso de intento de verificar la identidad digital contra un sistema donde se cuenta con credenciales, el remitente, es decir, el que intenta autenticarse es generalmente el usuario del sistema.

Back-end: Es la parte de una aplicación web que funciona de lado del servidor, en esta parte podemos emplear por ejemplo Servlets.

BIRT: Business Intelligence and Reporting Tools, es una herramienta de código abierto basada en eclipse que se integra con Java para generar reportes.

BPEL: Business Process Execution Language, es un lenguaje para la composición de servicios web, está basado en XML y agrega lógica de negocio para así ir ayudar a la conformación de un proceso.

BPMN: Business Process Model Notation, es la notación gráfica para especificar procesos de negocio en un modelo de procesos, la versión actual es la 2.0.

BPMS: Business Process Management Suite, o mejor conocido como un software que contiene procesos de negocio generalmente escritos en código BPMN y/o BPEL, así como herramientas

para la administración, análisis, ejecución, monitoreo y control. Algunas implementaciones pueden ser Intalio BPMS, Bizagi, Aura, JBPM, etc.

Bug: Se refiere a un error en el software que generalmente produce un resultado no esperado e indeseado.

Bytecode: Es un código intermedio que es portable, este conjunto de instrucciones está diseñado para ser ejecutado de forma eficiente por un software interprete. Este código no está hecho para ser leído por personas, ya que son compactos, generalmente emplean muchas literales, números, códigos, constantes y referencias a direcciones de memoria.

Continum: Al igual que cruise control o damage control es una herramienta de integración continua.

Correlación: Es el término empleado para describir la conexión entre dos mensajes en un proceso BPM para mantener siempre la misma instancia del proceso entre componentes del mismo.

Cross-context: Se refiere a la interacción entre servicios o scripts que se encuentran en diferente dominio, esto generalmente no es posible por políticas de seguridad, sin embargo se han creado diversas librerías, mods y frameworks que permiten realizarlo.

Cruise Control: Es herramienta de integración continua de código abierto basada en java que permite la compilación automática de proyectos java, utilizando ant o maven.

CSS: Es un lenguaje de hojas de estilo para establecer la disposición, conformación y el aspecto de un elemento dentro de una página web.

CSS2: Es la evolución de CSS1, en CSS2 se permite especificar diferentes estilos de acuerdo al tipo de medio, como browsers, impresoras, móviles, etc.

CSS3: Es la tercera especificación para las hojas de estilo, en esta versión se permiten realizar transformaciones, transiciones, animaciones y la selección es mucho más sencilla para establecer aspecto a partir de atributos incluso.

DamageControl: Fue una herramienta de integración continua, actualmente no está en desarrollo.

Eclipse: Es un IDE diseñado en principio para programar utilizando el lenguaje Java, posteriormente se extendió a otros lenguajes de programación inclusive a desarrollo de manuales y diagramas entre otras cosas. Es uno de los más utilizados por su gran cantidad de plugins y configuraciones.

EJB: Los Enterprise Java Beans son componentes que funcionan de lado del servidor proporcionando una arquitectura modular para aplicaciones empresariales.

ESB: Enterprise Service Bus, es un bus de servicio empresarial, el cual permite comunicar a través de mensajes múltiples aplicaciones que emplean diversos protocolos, empleando generalmente SOA.

ETL: Extract, Transform and Load, es un proceso que permite mover datos desde múltiples fuentes, transformarlos, filtrarlos y cargarlos en otras bases de datos, o sistemas.

Fase: En un proceso E2E una fase es el conjunto de elementos diagramados desde principio a fin, se le denomina fase porque es una abstracción más general que un escenario. De esta manera un proceso contiene una o más fases, y una fase uno o más escenarios.

Framework: Es un conjunto de conceptos, tecnologías, artefactos que sirven para la base y desarrollo de nuevo software.

Front-end: Se refiere a la parte de una aplicación web que es visible por el usuario y con la que interactúa, generalmente cuando se emplea Java se recurre a JSP, HTML, CSS y javascript.

Git: Es un sistema distribuido de administración y control de código fuente que se enfatiza en la velocidad. A diferencia de subversión, cada copia de trabajo es un sistema completo de control de versiones el cual es independiente de la red o de un servidor central.

Grid: Es una estructura que permite mostrar los datos de forma ordenada y detallada, a través de una cuadricula. A diferencia de una tabla, un grid permite realizar operaciones adicionales sobre las columnas que lo componen, como búsquedas, ordenamientos, filtros, entre otros.

GWT: Google Web Toolkit, es un conjunto de herramientas de código abierto que permite desarrollar aplicaciones javascript creadas a partir del lenguaje java.

Hadoop: Es un framework de código abierto desarrollado por Apache que permite trabajar con grandes cantidades de datos a través de un entorno distribuido (clúster) empleando modelos simples de programación y es altamente escalable.

Hbm: Es un archive empleado por hibernate para realizar el mapeo entre un objeto y un elemento en una base de datos por ejemplo una tabla.

Hibernate: Hibernate ORM, permite la interacción mediante la abstracción de objetos hacia una base de datos relacional generalmente, asimismo incorpora un lenguaje HQL que facilita el manejo de consultas.

Highcharts: Es una librería javascript para crear gráficos vistosos.

Ibatis: Es un framework de persistencia empleado en conjunto con java, a diferencia de hibernate, ibatis no genera código SQL por lo que la implementación de las consultas está a cargo del desarrollador.

IDE: Integrated Development Environment, es un ambiente de programación el cual se compone de múltiples utilidades para facilitar el desarrollo de programas o sistemas. Al menos consta de un editor de código, un compilador y un depurador.

Issue: Se atribuye generalmente a la necesidad de una mejora debido a un fallo, una característica faltante o simplemente un deseo de funcionalidad sobre un sistema de software.

J2EE: Es una plataforma empresarial de Oracle. Esta plataforma proporciona un API y un entorno de ejecución para desarrollar y ejecutar software de tipo empresarial, incluyendo servicios de red y web, así como características de escalabilidad, multicapa, confiabilidad y seguridad en aplicaciones que funcionan sobre la red.

Javascript: Es un lenguaje de programación interpretado, es parte de los navegadores web permitiendo a interactuar de lado del cliente alojando scripts en la página web. Últimamente se ha creado un servidor node.js con el cual se puede trabajar con javascript en backend.

jqGrid: Es un plugin basado en jQuery para incrustar grids con múltiples funcionalidades dentro de una página web, que además pueden interactuar con el lado del servidor.

jQuery: Es una librería javascript, permite la manipulación, manejo de eventos, animaciones y uso de ajax de una forma simple.

JSON: Javascript Object Notation, es una notación literal de objetos javascript.

JSP: Java Server Pages, es una tecnología que ayuda a los desarrolladores web a crear páginas web dinámicas, se puede hacer una analogía con PHP, solo que estas páginas emplean código Java, tanto para servlets como para jsp's es necesario contar con un contenedor o servidor de aplicaciones como Apache Tomcat.

LDAP: Lightweight Directory Access Protocol, es un protocolo que permite el acceso a un servicio de directorio ordenado y distribuido. En este directorio se encuentran un conjunto de objetos con atributos organizados de forma jerárquica y lógica.

LIUS: Lucene Index Update and Search, era un proyecto que empleaba lucene para poder realizar indexación, búsqueda y actualizaciones sobre archivos de texto plano y también archivos compatibles con Microsoft office y Open Office, así como PDF.

Maven: Es una herramienta para la gestión y construcción de proyectos java donde las dependencias se alojan en un repositorio central en internet.

Mime-type: Es un conjunto de identificadores empleados para establecer el tipo de contenido que se debe desplegar al abrir un archivo. Usados generalmente en clientes de correo electrónico, navegadores web y motores de búsqueda.

ORM: Object Relational Mapping, es una técnica empleada para poder convertir datos entre tipos incompatibles en un lenguaje de programación orientado a objetos, generalmente empleado para abstraer bases de datos.

PHP: PHP Hypertext Preprocessor, es un lenguaje de scripting que es popularmente empleado para el desarrollo de aplicaciones web.

POST: Es uno de los tipos de peticiones que se pueden realizar soportados por el protocolo HTTP, la característica principal es que el contenido de la petición va envuelto en el cuerpo de la misma en contraste con GET donde la información se envía a través de su URI.

Procmail: Es un agente de entrega de correo, permite ordenar los mensajes entrantes en varios directorios y filtrar spam entre otras cualidades.

PROTECO: PRograma de TEcnología en COmputo, Es un programa de formación de personal docente y de investigación de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.

Prototype: Es un framework de javascript que simplifica la programación orientada objetos a partir de javascript.

Repositorio: Es un sitio centralizado donde se mantiene y almacena información digital, generalmente archivos y código fuente.

Script.aculo.us: Es una librería javascript que ayuda al desarrollo de interfaces de usuario, proporcionando una vistosa y sencilla forma de crear.

SCRUM: Es un framework de administración de proyectos de desarrollo de software, ágil, iterativo e incremental. Scrum permite la creación de equipos organizados por sí mismos, alentando la intervención de todos los participantes y la comunicación de los mismos.

Servlets: Proporcionan a los desarrolladores web una tecnología simple, que permite extensibilidad de la funcionalidad de un servidor web además de poder acceder a sistemas existentes, estos se ejecutan de lado del servidor.

SOA: Es un patrón de diseño de arquitectura de software basado en piezas discretas de software proviendo funcionalidad de aplicaciones como servicios para otras aplicaciones. Esto es conocido como orientación a servicios, SOA es independiente de cualquier compañía, producto o tecnología.

SOAP: Simple Object Access Protocol, Es un protocolo que define como se debe realizar la comunicación entre dos objetos por medio de XML, tiene como características principales: extensibilidad (seguridad y ruteo), neutralidad (funciona sobre cualquier transporte) e independencia (funciona sobre cualquier lenguaje de programación).

Soap-UI: Es una herramienta multipropósito principalmente empleada para probar servicios web.

SOUI: Service Oriented User Interface, Es una propuesta de arquitectura donde las interfaces son construidas con javascript pero intercambian información a través de servicios web REST.

Subversion: Es un sistema de control de versiones, originalmente construido por CollabNet, éste ha sido donado a Apache.

Tempo: Intalio Tempo, es un conjunto de componentes de tiempo de ejecución que soportan tareas que interactúan con humanos. Está basado en una arquitectura SOA y ha sido pensado para permitir la interoperabilidad entre tecnologías como BPEL, BPEL4People, Xforms, REST y servicios web.

Token: Es una cadena de caracteres que tiene un significado en cierto ambiente de ejecución de una aplicación web. Generalmente relacionado con un proceso de autenticación.

Tomcat: Apache Tomcat es una implementación de código abierto que funciona como contenedor de aplicaciones que utilizan Java Servlets y Java Server Pages como tecnología.

Web 2.0: La web 2.0 describe principalmente páginas web que emplean tecnología que permite dinamizar el contenido provisto, así como la interacción.

Web 3.0: También conocida como la web semántica, basada en la idea de añadir metadatos semánticos y ontológicos a lo existente en la www, todo con el fin de emplear agentes inteligentes para ampliar la interoperabilidad entre sistemas.

WSDL: Web Service Description Language, es un lenguaje que describe interfaces de funcionalidad que ofrece un webservice, está escrito en forma de XML, éste wsdl especifica como un servicio debe ser llamado, que parámetros necesita y la estructura de retorno.

X-Forms: Es un formato XML para la especificación de interfaces de usuario y sirve como modelo de procesamiento de datos.

XML: eXtensible Markup Language, es un lenguaje de etiquetas utilizado para almacenar datos de forma legible.

XSLT: eXtensible Stylesheet Language Transformations, es un lenguaje que permite realizar transformaciones de documentos XML a otros, u otros objetos como archivos de texto plano o paginas HTML.

Bibliografía

1. Apache Software Foundation. (05 de Septiembre de 2013). *Archiva* . Recuperado el 23 de Septiembre de 2013, de Apache: http://archiva.apache.org/index.cgi

2. Cunningham & Cunningham, Inc. (2010, 5 19). *Test Driven Development.* Recuperado el 11 de Octubre de 2011, de http://c2.com/cgi/wiki?TestDrivenDevelopment

3. D3 Foundation. (2007). *D3: Design Driven Development.* Recuperado el 11 de Octubre de 2011, de http://www.designdrivendevelopment.org/

4. David F. Ferraiolo, D. R. (1992, Octubre 13). *National Institute of Standards and Technology*. Recuperado el 06 de Noviembre de 2013, de Computer Security Division: http://csrc.nist.gov/groups/SNS/rbac/documents/ferraiolo-kuhn-92.pdf

5. Eclipse Software Foundation. (27 de Mayo de 2013). *Eclipse*. Recuperado el 23 de Septiembre de 2013, de Hudson: http://wiki.eclipse.org/Hudson-ci/Meet_Hudson

6. Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.*

7. Extreme Programming. (1999). *Extreme Programming.* Recuperado el 11 de Octubre 10 de 2011, de http://www.extremeprogramming.org/

8. IBM. (2009, Febrero 18). *IBM*. Recuperado el 25 de Febrero de 2013, de IBM: http://pic.dhe.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=%2Fcom.ibm.websphere.express.doc%2Finfo%2Fexp%2Fae%2Fcovr_j2ee.html

9. Intalio, Inc. (2010). *Intalio.* Recuperado el 11 de Octubre de 2011, de http://www.intalio.com

10. Magpiebrain. (2005, 02 14). *Magpiebrain.* Recuperado el 11 de Octubre de 2011, de http://www.magpiebrain.com/2005/02/14/the-agile-release-process/

11. Microsoft. (2006, 1). *MSDN Magazine.* Recuperado el 11 de Octubre 2011, de http://msdn.microsoft.com/en-us/magazine/cc163665.aspx

12. Object Management Group. (1997). *Business Process Management Initiative.* Recuperado el 11 de Octubre de 2011, de http://www.bpmn.org

13. Oracle. (2010). *Sun Developer Network (SDN).* Recuperado el 11 de Octubre de 2011, de http://java.sun.com/blueprints/corej2eepatterns/Patterns/

14. Saitosoft. (2012). *Saiosoft*. Recuperado el 23 de Septiembre de 2013, de http://saitosoft.com.mx/?page_id=30

15. Saitosoft. (2007). *Saitosoft*. Recuperado el 11 de Octubre de 2011, de Saitosoft: http://saitosoft.com/

16. SICUMA. (s.f.). *Sistemas de información y Cooperativos de la Universidad de Malaga*. Recuperado el 25 de Febrero de 2013, de http://www.sicuma.uma.es/sicuma/independientes/argentina08/Badaracco/j2ee.htm

17. Sonarqube. (n.d.). *Sonarqube*. Recuperado el 23 de Septiembre de 2013, de http://www.sonarqube.org

18. Sourceforge. (2013, Septiembre 23). *Sourceforge*. Recuperado el 23 de Septiembre de 2013, de Checkstyle: http://checkstyle.sourceforge.net/

19. Springsource. (n.d.). *Springsource*. Recuperado el 22 de Octubre de 2013, de http://docs.spring.io/spring-security/site/features.html

20. The Apache Software Foundation. (2006, 08 30). *Apache ODE*. Recuperado el 11 de Octubre de 2011, de http://ode.apache.org

21. Thompson, I. (2007, 10). *Promonegocios*. Recuperado el 22 de Julio de 2011, de http://www.promonegocios.net/empresa/concepto-organizacion.html

22. Trac. (2013, Mayo 15). *Trac*. Recuperado el 23 de Septiembre de 2013, de http://trac.edgewall.org/