UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Ejecución y monitoreo de procesos de negocios distribuidos
Autores: Leonardo Damián Karabogosian (Nº Alumno 6991/9)
Director: Mg. Patricia Bazán
Asesor profesional: Lic. José Nicolás Martínez Garro
Carrera: Licenciatura en Sistemas – Plan 2003/07

Resumen

BPM (Business Process Management) ha cobrado importancia dentro del área de tecnología informática, la cual en los últimos años ha evolucionado desde el concepto de producto hacia el paradigma de servicios y soluciones. Por otra parte, el avance tecnológico tanto en términos de comunicación como en poder de cómputo, han hecho que Cloud Computing sea una opción potencial para la reducción de costos y mejoras en el procesamiento. La combinación de las tecnologías asociadas a Cloud Computing y BPM modifica aspectos tanto de diseño como de ejecución de los procesos de negocios. Los ambientes distribuidos en el contexto de los procesos favorecen el rendimiento y proponen incorporar el concepto de descomposición de procesos, permitiendo que los mismos se ejecuten tanto en un entorno cloud como en uno embebido. Si bien la descomposición de procesos es un tema abordado en los últimos años, el monitoreo de dichos procesos no ha sido demasiado explorado aún.

Este trabajo propone una implementación de una arquitectura para un sistema de monitoreo de procesos distribuidos utilizando Bonita Open Solution como motor de procesos, su API y el uso de conectores personalizados.

Palabras Claves

Procesos de negocio, BPM, BPMS, descomposición de procesos, Cloud Computing, ejecución distribuida, conectores, Bonita Open Solution, monitoreo de procesos de negocio.

Trabajos Realizados

- *Descomposición de procesos de negocio en varios subprocesos.*
- *Implementación de un conector en Bonita para la ejecución remota de procesos de negocios.*
- *Desarrollo de una aplicación web de monitoreo de procesos de negocio y actividades.*
- *Unificación de los subprocesos de negocio de los diferentes BPMS y visualización del diseño del proceso de negocio original.*

Conclusiones

La distribución de los procesos en un ambiente híbrido (embebido y cloud) permite el uso eficiente de los recursos tecnológicos y de la protección de los datos sensibles de la organización al mismo tiempo. La ejecución distribuida de los procesos y la aplicación de monitoreo presentada permiten incorporar las funcionalidades de flujo de control, flujo de datos, y monitoreo que se han perdido a causa de la descomposición de los procesos de negocio.

Trabajos Futuros

Disponer de una herramienta incluida de forma nativa en el BPMS que de manera automática lleve a cabo la descomposición de los procesos y el despliegue de los subprocesos; como así también de una arquitectura centralizada de monitoreo utilizando recolectores de eventos para evitar sobrecarga en cada uno de los nodos de la arquitectura de ejecución.

Fecha de la presentación: Marzo de 2015

Agradecimientos

Quisiera agradecer a todas esas personas que en estos años hicieron posible que yo me encuentre alcanzando este deseo, en especial:

A *Patricia Bazán* y a *José Martínez Garro* por la oportunidad que me dieron y el apoyo constante en la realización de este trabajo, y por los consejos y enseñanzas que me dejaron a lo largo de todo el camino.

A mis padres *Cecilia* y *Jorge*, que con su amor incondicional me apoyaron a lo largo de toda la carrera y por ser los mejores profesores que tuve y tendré en mi vida.

A mi hermana *Silvina* y a *Juan Manuel* que fueron los que me prestaban su tiempo en momentos difíciles.

A mis amigos "*Magios*" por el constante aliento que me brindaban en cada reunión y porque son una parte importante de mi vida.

A los muchachos de *Aknotec* por ser unos compañeros ejemplares y unas personas excepcionales.

Finalmente a la *Universidad Nacional de La Plata*, y especialmente a la *Facultad de Informática*, por permitirme crecer profesional y personalmente durante todos estos años.

Índice General

Capítulo 1 - **Introducción**

La combinación de las tecnologías asociadas a Cloud Computing y BPM (Business Process Management) modifica aspectos tanto de diseño como de ejecución de los procesos de negocios. Los ambientes distribuidos en el contexto de los procesos favorecen el rendimiento y proponen incorporar el concepto de descomposición de procesos, permitiendo que los mismos se ejecuten tanto en un entorno cloud o embebido. Si bien la descomposición de procesos es un tema abordado en los últimos años, el monitoreo de dichos procesos no ha sido demasiado explorado aún.

Este trabajo propone una implementación de una arquitectura para un sistema de monitoreo de procesos distribuidos utilizando Bonita Open Solution como motor de procesos, su API y el uso de conectores personalizados

1.1. Motivación

En los últimos años las empresas y organizaciones han apostado fuertemente a la eficiencia de los procesos de sus negocios. La utilización de BPMS para identificar y diseñar, ejecutar, monitorear y optimizar los procesos de negocios ha sido de gran ayuda para poder disminuir los costos, aumentar la productividad, mejorar los servicios a los clientes, crear un marco de organización y coordinación de actividades para el personal de la empresa u organización.

Otra de las tecnologías que actualmente está teniendo un interés masivo en las empresas y organizaciones es Cloud Computing. Con este nuevo paradigma, las empresas buscan proporcionar servicios de computación bajo demanda con una alta fiabilidad, escalabilidad y disponibilidad en un entorno distribuido. Implementar Cloud Computing significa un cambio de paradigma de los negocios y la infraestructura IT, donde el poder de cómputo, almacenamiento de datos y servicios se subcontratan a terceros y se ponen a disposición de las empresas y clientes, lo que ocasiona que los riesgos económicos y técnicos disminuyan.

Indefectiblemente, hace unos años, estas dos potentes herramientas en el área de IT, hicieron que se comience con el estudio e implementación de BPMS en la "nube". Esta combinación de técnicas de clouding y BPM ofrece un enfoque flexible y ágil, junto con las ventajas de ambos paradigmas. La gran capacidad computacional de los sistemas en el cloud y el "pago por uso" en lugar de enfrentar grandes inversiones en software y hardware, son dos grandes ventajas de la combinación de estas técnicas. Sin embargo, al utilizar un BPMS en la nube, se pierde el control sobre los datos sensibles del negocio, lo que conlleva a tomar un riesgo muy grande para las empresas de hoy en día.

Para poder utilizar BPM en el Cloud, y a la vez no tomar demasiados riesgos con la sensibilidad de la información, hace un tiempo se comenzó a estudiar la descomposición de procesos de negocios, donde parte de las actividades se hacen en un sistema BPMS local y otras actividades se ejecutan en el BPMS del cloud. Para la realización de este sistema híbrido, se estableció un modelo de distribución de procesos, actividades y datos, denominado PAD (Proceso-Actividad-Datos), en

donde se presentan varias formas de descomposición de procesos de acuerdo a las necesidades del negocio.

Actualmente podemos encontrar BPMS que se encuentren en servidores locales o en servidores localizados en el cloud brindando Software como Servicio (SaaS). La descomposición de procesos de negocios antes mencionada, permite que un proceso se ejecute en ambos ambientes, dependiendo de la lógica y modelo adoptado por el desarrollador del proceso al momento de la descomposición. Sin embargo, una vez que se realiza la descomposición del proceso, el monitoreo y seguimiento del proceso de negocio original en este sistema híbrido se dificulta, haciendo que se deba recolectar toda la información de cada uno de los servidores en donde se encuentran las partes del proceso.

Si bien el estudio de la descomposición de procesos de negocios se encuentra en una etapa bastante avanzada y podemos encontrar mucha información relacionada, el estudio de la integración de estos procesos descompuestos para el monitoreo y seguimiento de los procesos de negocio distribuidos es un campo en el cual se puede realizar un estudio de investigación y proponer un marco de trabajo que sirva como base para la optimización de procesos de negocio distribuidos.

En resumen, la motivación de este trabajo se basa en favorecer el seguimiento de procesos descompuestos en un sistema distribuido y cubrir de este modo la etapa de monitoreo y optimización del ciclo de vida de los procesos de negocios.

1.2. Planteo del problema

Con la finalidad de utilizar un BPMS local para la protección de los datos sensibles, y un BPMS en el Cloud para el procesamiento intensivo de actividades que necesiten una respuesta rápida, el inconveniente surge al descomponer el proceso de negocio original en dos o más partes que deben comunicarse para cumplir con el objetivo del proceso de negocio original.

Al utilizar, como mínimo, dos BPMS (uno en un sistema embebido y otro en el cloud), el monitoreo del proceso de negocio original se pierde. Por lo que resultaría necesario monitorear los subprocesos (producto de la descomposición del proceso original) por separado, unificar los datos obtenidos de cada uno, y analizar estos datos para obtener los posibles problemas o mejoras que pueden realizarse para aumentar la eficiencia del proceso de negocio. Realizar esto no solo significaría una carga de trabajo adicional para el analista de los procesos de negocio, sino que también sería necesario que el mismo conozca las diferentes herramientas en las que se ejecuta el proceso de negocio.

Estas tareas adicionales que surgen de la descomposición de procesos van en contra del paradigma de BPM, ya que el control y monitoreo de los procesos, una parte vital al momento de la optimización de los procesos de negocio, se vuelve cada vez menos efectivo, ocasionando la disminución de la eficiencia de todos los procesos de negocio de la organización.

1.3. Solución propuesta

El objetivo de esta tesis es investigar y elaborar un marco de desarrollo y ejecución para la ejecución de procesos de negocios distribuidos en diferentes motores de proceso de Bonita Open

Solution para que se comporten como si se tratase de un solo proceso, y a la vez, brindar una herramienta de monitoreo distribuido basado en servicios web para el proceso de negocio unificado.

Junto con el estudio de los procesos de negocio distribuidos, se presentará el desarrollo de un sistema de monitoreo en un ambiente híbrido entre motores de procesos de Bonita Open Solution, debido a que es un sistema de gestión de procesos de negocio que dispone de una interfaz de programación (API) que se basa en una arquitectura REST, en donde el intercambio de mensajes entre cliente y servidor se realiza a través de HTTP, y a la vez es un software libre y *open source*.

El sistema de monitoreo se basará en las siguientes etapas:

- Implementación de un conector en Bonita OS para la ejecución de un proceso de negocio descompuesto entre distintos servidores distribuidos.
- Investigación de la API REST de Bonita Open Solution para la instanciación de procesos remotos y el acceso a las propiedades de procesos ya instanciados y desplegados en el motor remoto.
- Confección de una base de datos local para la persistencia de la información relacionada a los procesos de negocios instanciados de forma local junto a la relación que tiene con los procesos de negocio remotos.
- Aplicación Web de monitoreo y seguimiento de procesos de negocios distribuidos. Utilización de Web Services y API REST para la integración de los procesos descompuestos y distribuidos, y la visualización unificada del proceso global para el correcto seguimiento del proceso de negocio.

Con la culminación del trabajo se esperará disponer de una aplicación que sea capaz de monitorear procesos distribuidos en varios motores de Bonita OS, para poder integrar el monitoreo y optimización en la descomposición de procesos de negocio.

A la vez, presenta un marco teórico y un puntapié inicial para la futura investigación de procesos distribuidos entre distintos BPMS utilizando otras tecnologías como soporte para los distintos motores de procesos de negocios.

1.4. Estructura de la tesina

A continuación se presentará la organización de la tesina, introduciendo al lector en cada uno de los capítulos que se compone la misma.

- **Capítulo 1 – Introducción**: Se presentan los componentes generales de la tesina, introduciendo la motivación por la cual se realiza la investigación y desarrollo de una herramienta para el monitoreo de los procesos de negocios distribuidos. Se plantea el problema inicial de la descomposición de los procesos de negocio, la solución propuesta para contrarrestar el problema inherente de la descomposición de procesos y la organización de sus partes.
- **Capítulo 2 – BPM y Cloud Computing. BPaaS**: se introduce a Business Process Management (BPM), Cloud Computing (Computación en la Nube), la relación entre estas dos tecnologías y cómo surge BPaaS (Business Process as a Service) a partir de la unión de estas dos. De esta manera se investiga acerca de los procesos de negocio distribuidos.

- **Capítulo 3 – Descomposición de procesos y monitoreo distribuido**: se da una introducción al uso de BPMS en la nube, para luego dar un marco teórico en la combinación de un esquema embebido y un esquema en el *cloud* a partir de un modelo PAD (Proceso – Actividad – Datos), y cómo a partir de este modelo se lleva a cabo la descomposición de procesos con un quinto patrón.
- **Capítulo 4 – Bonita y su capacidad de extensión de funcionalidades**: se hablará del BPMS Bonita Open Solution, una suite para la gestión de los procesos de negocio sus componentes, APIs, y conectores que la componen, y cómo utilizar estos componentes para extender la funcionalidad.
- **Capítulo 5 – Solución propuesta**: se presenta una solución propuesta para la ejecución y monitoreo de procesos de negocios distribuidos. Se introducirán temas generales como el diseño, ejecución y monitoreo de procesos de negocio distribuido, junto con la arquitectura necesaria para llevar a cabo un ejemplo sencillo.
- **Capítulo 6 – Validación con un ejemplo**: se presentará un ejemplo de un proceso de negocio y se llevará a cabo los pasos necesarios para la descomposición, ejecución, monitoreo y unificación de los subprocesos.
- **Capítulo 7 – Conclusiones y trabajos futuros**

Capítulo 2 - **BPM y Cloud Computing. BPaaS.**

En este capítulo se presentará una introducción a Business Process Management (BPM), Cloud Computing (o Computación en la Nube), la relación entre estas dos tecnologías y cómo surge BPaaS (Business Process as a Service) a partir de la unión de estas dos. De esta manera se intenta dar un marco teórico como puntapié inicial en la investigación de los procesos de negocio distribuidos.

2.1. Business Process Management (BPM)

Business Process Management, o la gestión de procesos de negocio, se basa en la observación de que cada producto que una organización brinda al mercado es el resultado de una serie de actividades que deben realizarse para obtenerlo. Los procesos de negocio son para la organización una herramienta clave para organizar estas actividades y entender de mejor manera las interrelaciones entre estas.

La tecnología de la información en general y los sistemas de información en particular, juegan un rol importante en la gestión de procesos de negocio ya que muchas de las actividades que realiza una organización están soportadas por los sistemas de información.

Los procesos de negocio son un concepto importante utilizado para facilitar la colaboración entre las personas y los recursos de la organización (tales como los sistemas de información), y de esta manera entender cómo opera internamente una organización, con el fin de alcanzar los objetivos de negocio de forma eficiente y efectiva en un mercado que cambia continuamente, en el cuál es imprescindible la rápida adaptación a estos cambios.

En [1], podemos encontrar una definición precisa de lo que es un *proceso de negocio*: consiste en un conjunto de actividades que son realizadas en coordinación en un ambiente técnico y organizacional. Estas actividades en conjunto alcanzan el objetivo del negocio. Cada proceso de negocio es realizado por una única organización, pero puede interactuar con procesos de negocios realizados por otras.

La *gestión de procesos de negocio* (BPM) incluye conceptos, métodos y técnicas para ayudar con el diseño, la administración, configuración, publicación y análisis de los procesos de negocio. La base de BPM es la representación explícita de los procesos de negocio con sus actividades y las restricciones de ejecución entre ellos.

Hoy en día las organizaciones se benefician de los procesos de negocio ejecutándolos en sistemas de software que coordinan las actividades y monitorean los resultados de cada una de éstas actividades. Estos sistemas de software son denominados BPMS (Business Process Management Systems) o Sistemas de Gestión de Procesos de Negocio. Un *BPMS* es un sistema de software genérico que se utiliza para representar y coordinar las actividades involucradas en los procesos de negocio.

Un BPMS no sustituye a otros sistemas o aplicaciones, sino que se encarga de coordinar éstas para la consecución de los procesos de la organización. Es decir, en un momento determinado, el

BPMS puede ejecutar cualquiera de estas herramientas a través de las interfaces que otorgan las mismas.

Además, el BPMS es capaz de monitorear cada una de las instancias de un proceso de negocio y dar una visión de aquellas actividades que han sido realizadas, el tiempo que tomó realizarlas y si han terminado en éxito o fracaso.

Para ilustrarlo con un ejemplo, puede ver en la Figura 2.1 [1] el simple caso de una orden de compra, en la cual, al recibirse la orden de compra, se envía la factura, se recibe el pago, se envían los productos y se archiva la orden de compra.

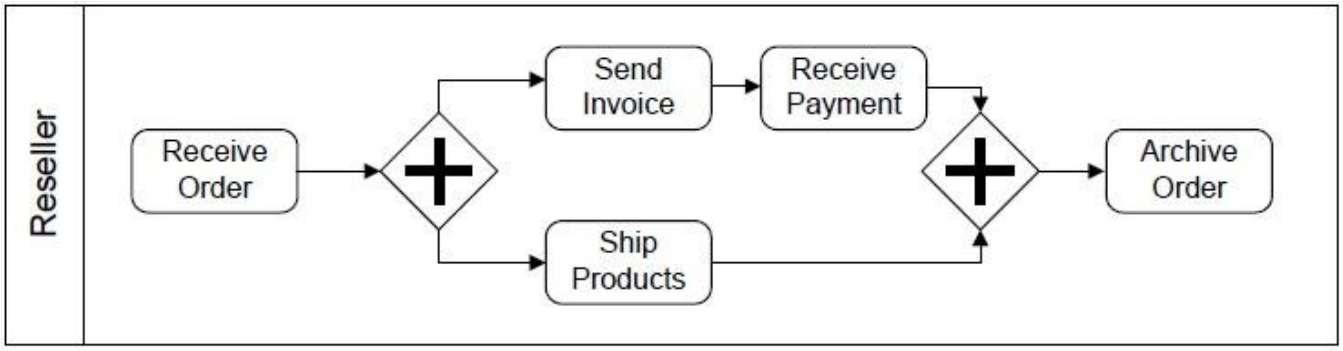


Figura 2.1. Diagrama de un proceso de negocio de una orden de compra simple

El diagrama que se muestra en la Figura 2.1 es el modelo del proceso de negocio. Un *modelo de proceso de negocio* consiste en un conjunto de modelos y ejecución de actividades entre ellos, éstos son los principales objetos para implementar los procesos de negocio. Cada orden de compra que se procesa de acuerdo a este modelo recibe el nombre de instancia de proceso de negocio. Por lo tanto, una *instancia de proceso de negocio* representa un caso concreto en el negocio operacional de la organización, que está formado por instancias de actividades.

Las instancias de los procesos de negocio son ejecutadas, controladas y monitoreadas por los BPMS como componente de software centralizado, lo que se conoce comúnmente como *orquestación de procesos*.

De la misma manera que las actividades de un proceso de negocio interactúan entre sí, los procesos de negocio también son capaces de interactuar con otros. Esta interacción de los procesos de negocio se denomina *coreografía de procesos*. Esta interacción se realiza a través del envío y recepción de mensajes entre los procesos, tal como se puede observar en la Figura 2.2 [1] entre un proceso Comprador y un proceso Distribuidor. Para que la interacción entre ambos procesos de negocio sea correcta, antes de comenzar a interactuar entre ellos se debe acordar entre ambos procesos la coreografía que se va a utilizar.

Para reforzar las definiciones de orquestación y coreografía, podemos citar a M.Juric en [2], quién nos dice que la orquestación sigue la noción de un proceso central, el cual toma control sobre los servicios de los procesos involucrados y coordina la ejecución de las diferentes operaciones de los servicios en el proceso. Los servicios involucrados no conocen y no necesitan saber que están en un proceso de negocio, solo el coordinador central es quien sabe esto. Esta orquestación centralizada es lo que se conoce como un proceso de negocio ejecutable. Por otra parte, la coreografía no utiliza un control centralizado del proceso, en lugar de esto, cada servicio de negocio sabe exactamente

cuándo ejecutar sus operaciones y con quién va a interactuar. Se basa principalmente en un esfuerzo colaborativo enfocado en el intercambio de mensajes entre procesos de negocio, es por esto que los participantes en una coreografía necesitan especificar las operaciones permitidas entre ellos.

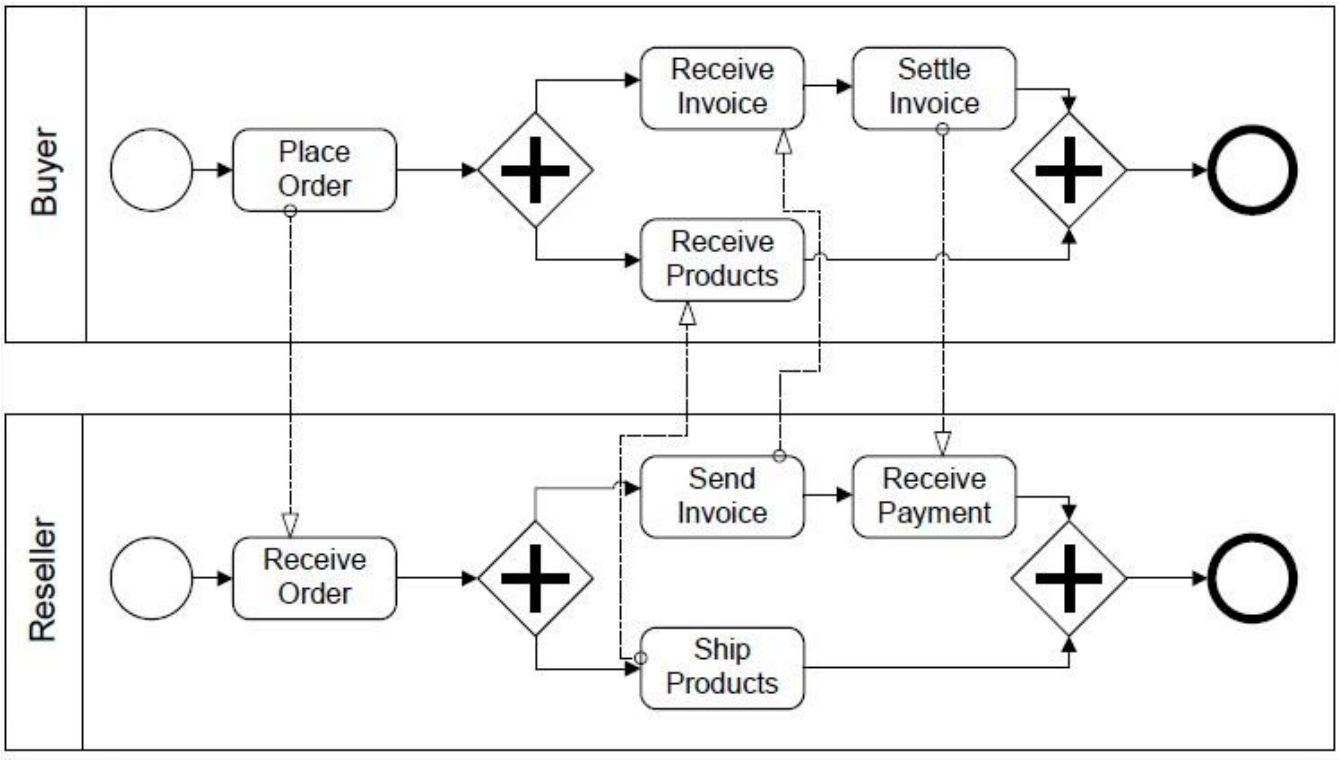


Figura 2.2. Interacción de dos procesos de negocio. Coreografía entre procesos.

La representación gráfica de los procesos de negocios visto en los ejemplos, se centran en la estructura del proceso y en la interacción entre los participantes en lugar de los aspectos técnicos de su ejecución. Este es un tema importante en el modelado de procesos de negocio, ya que en la definición de los procesos de negocio y su interacción no se especifica cuales estrategias de implementación o plataformas que deben utilizarse.

2.1.1. Ciclo de vida de los procesos de negocio

El ciclo de vida de los procesos de negocios consiste en fases que están organizadas en una estructura cíclica, y que a pesar de estar relacionadas unas con otras, no imponen un orden temporal estricto en las que se tienen que ejecutar. De esta manera, el diseño y desarrollo de los procesos de negocio puede realizarse de una manera incremental y evolutiva, en donde es posible la concurrencia de actividades entre múltiples fases.

En la Figura 2.3 [1] podemos ver el ciclo de vida de los procesos de negocio y las fases que lo componen.

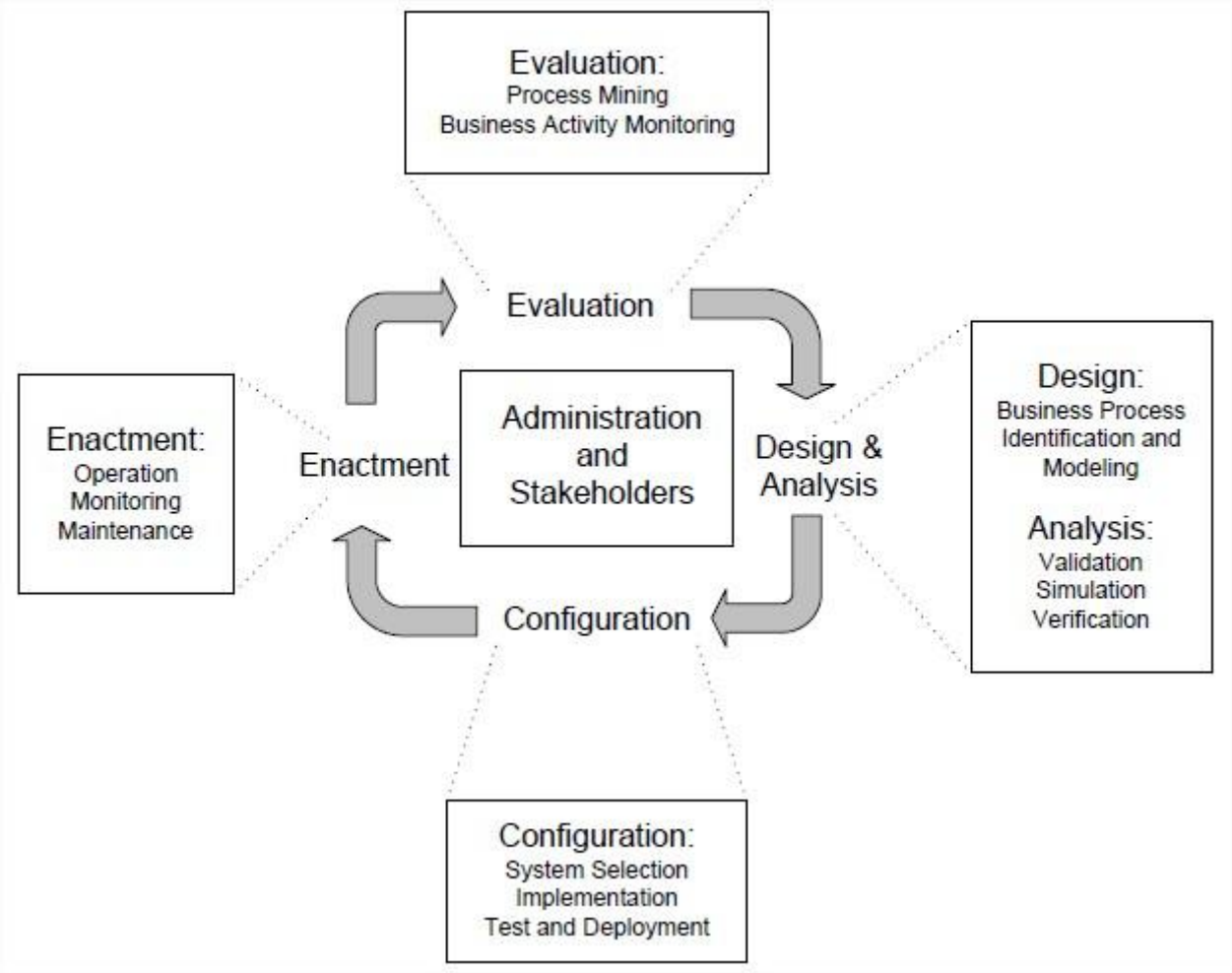


Figura 2.3. Ciclo de vida de los procesos de negocio

La fase de **Diseño y Análisis** consiste en la identificación, revisión, validación y representación de los modelos de los procesos de negocio. En esta fase se estudian los aspectos técnicos y organizacionales de las actividades que se deben llevar a cabo y se vuelcan a un diagrama utilizando una notación estándar llamada BPMN (Business Process Management Notation). Luego, este diseño se analiza en forma conjunta con las diferentes *stakeholders* del proceso de negocio para validar que realiza lo solicitado.

Una vez que el modelo es diseñado y analizado, el proceso pasa a la fase de **Configuración**, donde el proceso de negocio debe ser implementado. Acá se toman en cuenta aspectos relacionados a donde van a ubicarse y ejecutar los procesos de negocio, si es necesario utilizar un BPMS como plataforma para la ejecución de los procesos, o si lo que se quiere es intervención humana con un sistema de flujo de trabajo. Una vez realizado esto, el proceso de negocio necesita ser probado para verificar que cumpla con los requerimientos y que se comporte como se espera.

Luego de la fase de Configuración, las instancias de los procesos de negocio pueden ser publicados. La fase de **Publicación** (o Promulgación) consiste en desplegar el proceso de negocio para la ejecución dentro de la organización. El BPMS controla la ejecución de los procesos, realizando la orquestación de las actividades, y garantizando que se cumplan las restricciones especificadas en el modelo del proceso. Con el uso del BPMS se almacenan los datos de la ejecución

de cada instancia en archivos log. Estos datos son utilizados en la siguiente fase del ciclo de vida para realizar la evaluación de los procesos.

La fase de **Evaluación** utiliza los datos de la ejecución de los procesos para evaluar y mejorar los modelos de los procesos y su implementación. Al mismo tiempo, en esta fase se utiliza un módulo especial de los BPMS llamado *Business Activity Monitoring (BAM)* que monitorea los procesos de negocio en tiempo real y determina las actividades que no tienen una eficiencia aceptable, que junto a los archivos de logs contribuye a la mejora continua de los procesos de negocio.

Por último, en el centro del ciclo de vida de **Administración y Stakeholders (las partes interesadas)**, se encuentran los encargados de llevar a cabo las diferentes fases de los procesos de negocio a lo largo del ciclo de vida. Desde el inicio del proceso de negocio hasta el final del mismo, se pueden clasificar algunos roles de los stakeholders como los siguientes: dueño del proceso, ingeniero del negocio, diseñador del proceso, participante del proceso, actor o empleado, responsable del proceso, arquitecto del sistema, desarrollador.

2.1.2. Business Process Management Notation (BPMN)

El objetivo primario de BPMN es proveer una notación que sea legible y entendible para todos los usuarios de negocios, desde los analistas que realizan el diseño inicial de los procesos y los responsables de desarrollar la tecnología que ejecutará estos procesos, hasta los gerentes de negocios encargados de administrar y realizar el monitoreo de los procesos. Además, con el uso de BPMN, los usuarios y proveedores de servicios pueden comunicar los procesos de negocio de una forma estándar.

BPMN ha sido desarrollado para proveer a los usuarios de una notación estándar de forma análoga a como UML estandarizó el mundo de la ingeniería del software. Sin embargo, BPMN y UML usan enfoques muy diferentes para modelar procesos de negocio.

BPMN define un diagrama de procesos de negocio (BPD) basado en una técnica adaptada de diagramas de flujo para la creación de modelos gráficos de operaciones de procesos de negocio. Un modelo de procesos de negocio, es una red de objetos gráficos que representan las actividades (por ejemplo tareas) y los controles de flujo que definen su orden de ejecución.

Dentro de la notación de BPM se provee un conjunto de cuatro categorías de notación que permite al lector del diagrama de procesos de negocio reconocer fácilmente los elementos básicos y comprender el diagrama. Estas cuatro categorías básicas de elementos, que podemos ver en la Tabla 2.1 [3], son:

- Flow Objects (objetos que representan el flujo)
- Connecting Objects (objectos conectores)
- Swimlanes (andariveles)
- Artifacts (artefactos)

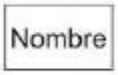
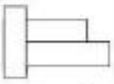
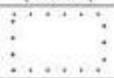
Categoria	Elemento	Descripción	Gráfica
Flujos	Evento	Es algo que sucede durante el curso del proceso de negocio. Afectan al flujo del proceso. Normalmente tienen una causa (disparador) o un impacto (resultado). Dependiendo de cuando afectan al flujo serán eventos iniciales, intermedios o finales.	
	Actividad	Es un termino genérico para el trabajo que realiza una compañía. Puede ser atómica (tarea) o compuesta (sub-proceso). Para indicar la no atomicidad se coloca un signo + en la esquina del símbolo de actividad.	
	Gateway	Se utiliza para controla la convergencia o divergencia de flujos. Representa una decisión para mezclar o unir caminos.	
Conexiones	Flujo de secuencia	Se utiliza para mostrar el orden o secuencia en que las actividades se realizan en un proceso	
	Flujo de mensajes	Se utiliza para mostrar el flujo de mensajes entre dos participantes separados.	
	Asociación	Se utiliza para mostrar entradas y salidas de actividades.	
Swimlanes	Pool (fondo común	Representa un participante en un proceso. Actúa como contenedor grafico para particionar un conjunto de actividades.	
	Lane (sendero)	Es una sub-particion dentro de un pool y puede extenderse a todo lo largo o ancho del pool. Se utilizan para organizar y categorizar actividades.	
Artefactos	Objetos de datos	Mecanismo para mostrar como los datos son requeridos y producidos por las actividades. Se conectan a las actividades por asociaciones.	
	Grupos	Se utiliza para documentación o para propósitos de análisis, pero no afecta al Flujo de Secuencias	
	Anotaciones	Mecanismo para que quien esta modelando provea información adicional para el lector del diagrama.	

Tabla 2.1. Elementos básicos de BPMN

Con BPMN, el control y los mensajes de flujo entre procesos son primeramente modelados. Para modelar un flujo sólo se modelan los eventos que ocurren. Las decisiones entre flujos se modelan con gateways. Si bien el modelado está centrado en flujos y eventos, principalmente se centra en los eventos, que son los elementos disparadores de determinadas situaciones.

Un proceso en el flujo puede contener subprocesos que cuando son atómicos se denominan tareas. Los eventos se dividen en iniciales, intermedios y finales según se desencadene al inicio, durante o al finalizar el flujo del proceso.

Además, se puede especificar "quién hace qué", ubicando los procesos en pools que denotan quién hace la tarea pudiendo particionar el pool en lanes o senderos. Típicamente un pool representa a toda la organización mientras que el lane representa un departamento dentro de la misma. [3]

De esta manera, podemos ver que la notación BPM ha sido diseñada para permitir construir diagramas fáciles de leer y de entender, sin importar la complejidad del diagrama, y que además puedan ser leídos por cualquier lenguaje de ejecución de procesos de negocio. [4]

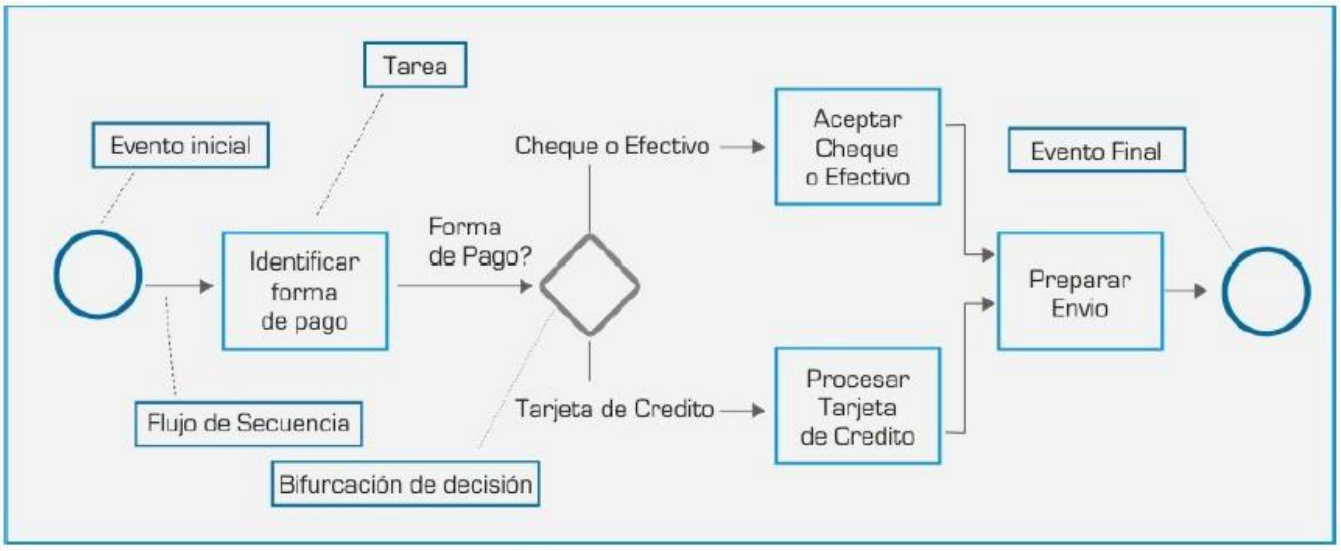


Figura 2.4. Ejemplo de un diagrama de proceso de negocio simple

Para mostrar un ejemplo de un diagrama, en la Figura 2.4, extraída de [3], podemos ver un proceso de negocio graficado que representa el proceso de recepción del pago de un cliente. El proceso se inicia con la actividad de Identificar la forma de pago. Se prevén dos posibles formas de pago: en efectivo o con tarjeta de crédito. En cada caso se aplica la actividad de aceptar el pago según la forma del mismo y se pasa a la actividad de empaque de la mercadería, finalizando así el proceso.

2.2. Cloud Computing

La Computación en la Nube (Cloud Computing) está cambiando la forma en que las industrias y las organizaciones hacen sus negocios, en que los recursos dinámicamente escalables y virtualizados se proporcionan como un servicio a través de Internet. Este modelo crea una oportunidad nueva para las organizaciones que quieren brindar servicios bajo demanda con alta confiabilidad, escalabilidad y disponibilidad en ambientes distribuidos. [5]

El Instituto Nacional de Estándares y Tecnología (NIST) define a Cloud Computing en [6] como un modelo para habilitar acceso conveniente por demanda a un conjunto compartido de recursos computacionales configurables, por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios, que pueden ser rápidamente aprovisionados y publicados con un esfuerzo mínimo de administración o de interacción con el proveedor de servicios. Este modelo de nube

promueve la disponibilidad y está compuesto por cinco características esenciales, tres modelos de servicio y cuatro modelos de despliegue (Figura 2.5).

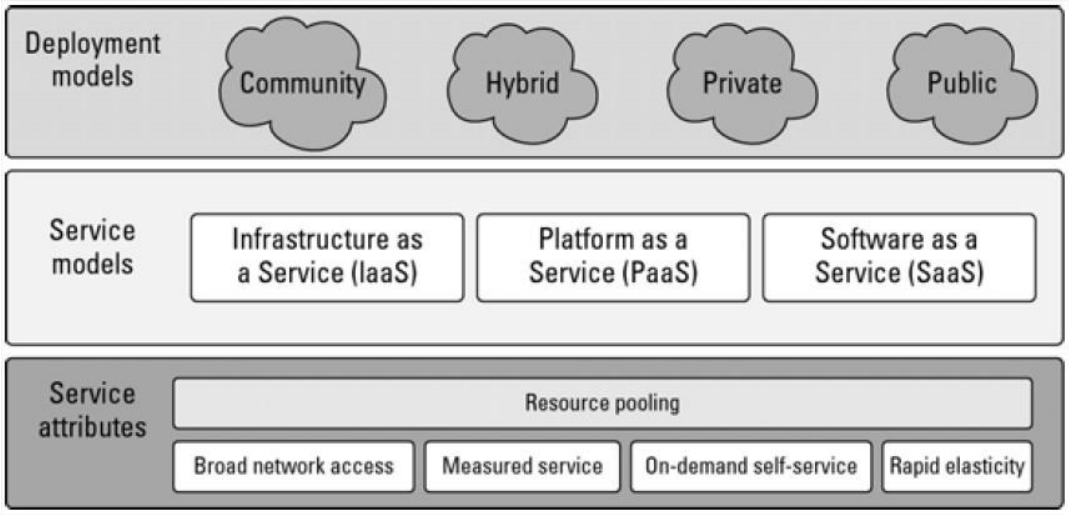


Figura 2.5. Modelo de Cloud Computing por el NIST [7]

2.2.1. Características Esenciales

Dentro de las cinco características esenciales de Cloud Computing según el NIST [6][7][8] tenemos:

1. **Bajo demanda y Autoservicio:** El usuario dispone automáticamente de las distintas necesidades de recursos que el proveedor de servicios de cloud brinda sin necesidad que éste último tenga que realizar intervenciones manuales. El usuario del cloud puede configurar a su criterio las capacidades de cómputo tales como tiempo de servidor, almacenamiento, memoria RAM, cantidad de procesadores, velocidad de red, etc.

   Esta característica aporta un gran beneficio al usuario dado que reduce en gran medida las complicaciones y costos que normalmente conllevan la adquisición de recursos propios IT, siendo así uno de sus grandes beneficios.

2. **Amplio acceso a la red:** El Cloud Computing permite el acceso a los datos desde cualquier lugar. Solo se necesita un navegador web y conexión a Internet para disfrutar de los servicios en la nube, no hace falta tener un sistema operativo determinado o instalar un software específico en cada cliente. La combinación de dispositivos móviles (tablets y smartphones) y fijos crea nuevas oportunidades en el desarrollo de la actividad empresarial, permitiendo plena operatividad.

   Esta característica es especialmente importante en organizaciones distribuidas geográficamente, permitiendo el acceso a los recursos con independencia de aspectos como la ubicación y la jornada laboral. Es importante puntualizar que esta característica establece una limitación, ya que no es posible utilizar las aplicaciones en la nube si no hay conexión a Internet.

3. **Pooling de recursos:** Esta característica permite a los distintos proveedores compartir sus recursos tanto físicos como virtuales entre los distintos usuarios, disminuyendo costes, maximizando la disponibilidad y de acuerdo a los requerimientos de los usuarios. Para poder habilitar este modelo *multi-tenant* de forma óptima, es necesario que los recursos disponibles (capacidad de computación, almacenamiento, velocidad de red, etc.) se asignen y balanceen de forma automática en base a las distintas peticiones de los usuarios.

   Los usuarios pueden ignorar el origen y la ubicación de los recursos a los que acceden, aunque sí es posible que sean conscientes de su situación a determinado nivel, como el país donde se localiza el CPD (centro de procesamiento de datos).

4. **Escalabilidad y rapidez:** La sencillez con la que se pueden añadir o eliminar recursos supone una ventaja frente al modelo tradicional. En Cloud Computing es posible añadir o eliminar recursos en cuestión de minutos, aumentando o disminuyendo el almacenamiento o el número de procesadores sin que la aplicación se vea afectada.

   En el ámbito software, la flexibilidad es muy alta, pudiendo incorporar nuevas funcionalidades a todos los usuarios de forma más rápida que sobre sistemas tradicionales.

5. **Servicio medido:** Los sistemas en la nube controlan automáticamente y optimizan el uso de recursos mediante una capacidad de medición a algún nivel de abstracción adecuado al tipo de servicio; por ejemplo, almacenamiento, procesamiento, ancho de banda y cuentas de usuario activas. El uso de estos recursos puede ser monitoreado, controlado y reportado, proporcionando transparencia tanto para el proveedor como para el consumidor por el servicio utilizado. Esto posibilita diferentes modalidades de pago como:
   - Pago por disponibilidad del servicio: se acuerda un precio por el tiempo en el que los recursos contratados están habilitados al usuario.
   - Pago por uso: se basa específicamente en los servicios consumidos por el usuario (almacenamiento realizado, transacciones comerciales realizadas, etc.), de forma análoga a como se paga el servicio de electricidad, gas o agua potable.
   - Pago por paquetes escalables: el pago se realiza por reservas de slots fijos que se pueden incrementar en unidades acotadas.

2.2.2. Modelos de Implementación

Las arquitecturas de Cloud pueden ser analizadas de dos perspectivas diferentes, una es de un punto de vista organizacional y otra es de una visión tecnológica. En esta parte, veremos el punto de vista organizacional (o modelo de implementación) en donde la distinción se basa en cómo los usuarios y/o proveedores están separados. Mientras que en la *Sección 2.2.3: Modelos de Servicio*, veremos la parte técnica de la arquitectura de Cloud, que está más orientada a las características del funcionamiento de la nube [9].

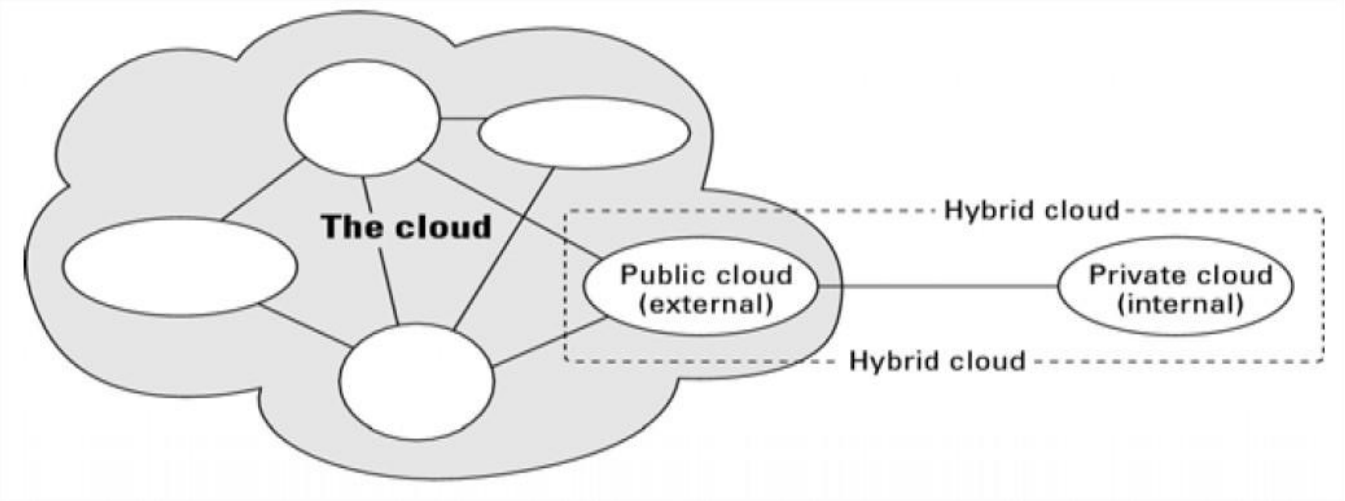


Figura 2.6. Modelos de Implementación de Cloud Computing [7]

El modelo de implementación define el propósito de la nube y en dónde se encuentra la nube. Según el NIST, existen cuatro modelos de implementación [6][7][8][9]:

1. **Nube Pública:** Una nube pública (también llamada 'nube externa') comprende todas las ofertas de cloud donde los proveedores y los usuarios potenciales no pertenecen a la misma unidad organizativa. Los proveedores ofrecen los servicios de cloud al público a través de un portal web de autoservicio donde los usuarios pueden especificar los servicios que desean contratar.
2. **Nube Privada:** En una nube privada (o "nube interna", o "IntraCloud") los proveedores y los usuarios pertenecen a la misma organización, que por lo general se trata de organizaciones grandes. La principal cuestión de este modelo es que se desea brindar mayor seguridad a la información sensible y a la vez aumentar la calidad del servicio en la nube.
3. **Nube Compartida:** Este modelo de implementación permite que distintas organizaciones con necesidades comunes (investigaciones, requerimientos de seguridad, políticas, cumplimientos) compartan una misma infraestructura o servicio Cloud, permitiendo una mayor agilidad en términos de colaboración o interoperabilidad entre ellas.
4. **Nube Híbrida:** El modelo híbrido interconecta a dos o más tipologías de nubes, privadas, públicas o compartidas, permitiendo la portabilidad de datos o aplicaciones entre ellas. De acuerdo a la seguridad que desea utilizar sobre su activo principal que es la información, la organización transfiere los datos y/o funcionalidades nubes públicas, privadas o compartidas.

2.2.3. Modelos de Servicio

Desde que se ha desarrollado la computación en la nube, los proveedores de cloud ofrecen diferentes modelos de servicio asociados a ellos. De esta forma, en la actualidad existe una gran cantidad de modelos de servicio que siguen la siguiente forma:

XaaS, or <Something> as a Service, o en español, <Cualquier cosa> como Servicio. [7]

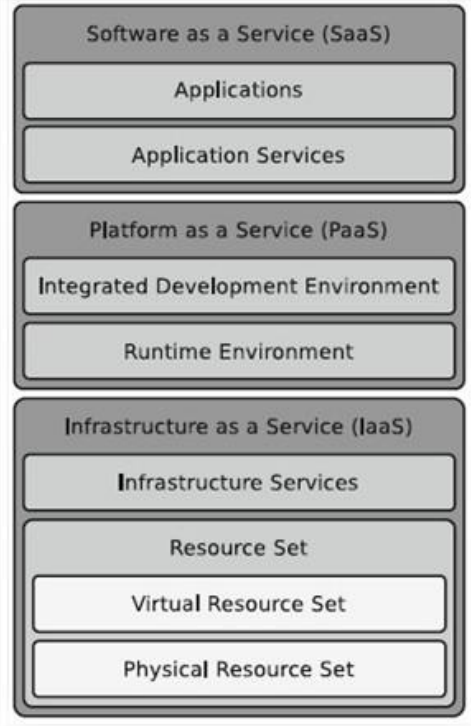


Figura 2.7. Pila de Servicios en Cloud Computing [9]

Sin embargo, existen tres tipos de servicios que han sido aceptados universalmente (conocido como modelo SPI) y que se pueden ver en la Figura 2.7, estos son:

1. **Software como Servicio (Software as a Service - SaaS):** en este modelo, el proveedor pone a disposición una aplicación completa como servicio para el usuario final. Este servicio se ejecuta como una instancia única de software que corre en la infraestructura que el proveedor gestiona. Los clientes acceden a estas aplicaciones a través de interfaces de cliente livianas como navegadores web, dispositivos móviles o interfaces de programas. Entre los servicios más comunes que podemos encontrar están: Google Maps, Google Docs, Microsoft Windows Live, Twitter, Facebook, entre otras.
2. **Plataforma como Servicio (Platform as a Service - PaaS):** este servicio cuenta con un ambiente de programación y un ambiente de ejecución que está dirigido más que nada a desarrolladores en lugar de usuarios finales. Entre algunos de los servicios más conocidos podemos encontrar la App Engine de Google, Azure de Microsoft, o Facebook Platform de Facebook, donde cada uno de estos servicios tienen un lenguaje de programación y herramientas que brinda el proveedor. El cliente no controla ni gestiona la infraestructura, sólo es responsable de instalar y desplegar las aplicaciones que desarrolla.
3. **Infraestructura como Servicio (Infraestructure as a Service - IaaS):** el proveedor del servicio le brinda a los clientes procesamiento, almacenamiento, dispositivos de red y otros recursos fundamentales, en la mayoría de los casos a través de la tecnología de virtualización, en donde cliente es el responsable de instalar el sistema operativo y aplicaciones que le sean necesarias, y gestionar el almacenamiento y la interacción del usuario con el sistema.

Como se mencionó anteriormente, los modelos de servicio que ofrece la nube siguen el esquema XaaS ("Cualquier cosa como Servicio") como regla general, sin embargo, cualquier modelo de servicio que una organización desee establecer gira en torno al modelo SPI (SaaS, PaaS, e IaaS), esto significa que las organizaciones utilizan algún servicio del modelo SPI como base para ofrecer los servicios propios. Entre algunos ejemplos de modelos de servicios que generan las organizaciones podemos encontrar: StaaS (Storage as a Service), se brinda espacio de almacenamiento como servicio; IdaaS (Identity as a Service), pensado como un inicio de sesión único para la nube; SaaS (Security as a Service), modelo de tercerización para la gestión de la seguridad; entre muchos más.

Para BPM también existe un modelo de servicio en cloud computing llamado BPaaS, o Business Process as a Service, que consiste en un modelo de servicio donde las aplicaciones que se ofrecen en cloud son del tipo procesos de negocio o workflows. Este modelo se utiliza como uno de los temas centrales de este trabajo, y se presenta en la siguiente sección.

2.3. BPaaS: Business Process as a Service

Procesos de Negocio como Servicio (BPaaS) es un modelo de servicio en el *cloud* donde las aplicaciones que se ofrecen son del tipo procesos de negocio o *workflows*.

BPaaS, al igual que varios modelos de servicios en cloud, permite a los usuarios utilizar un BPMS situado en la nube bajo una modalidad de "pago por uso", en lugar de enfrentarse a grandes costos de inversión en software, hardware y mantenimiento. Por esto, con la popularización de la computación en la nube, se espera que BPaaS sea una solución para aquellas pequeñas y medianas organizaciones que no pueden afrontar los costos de un software empresarial. [10][11][12]

Es común ver a BPaaS como un modelo especial situado por sobre la capa del modelo SaaS (Figura 2.8), en el que los proveedores de la nube proporcionan métodos para el modelado, utilización, personalización, y ejecución distribuida de procesos de negocios [12].

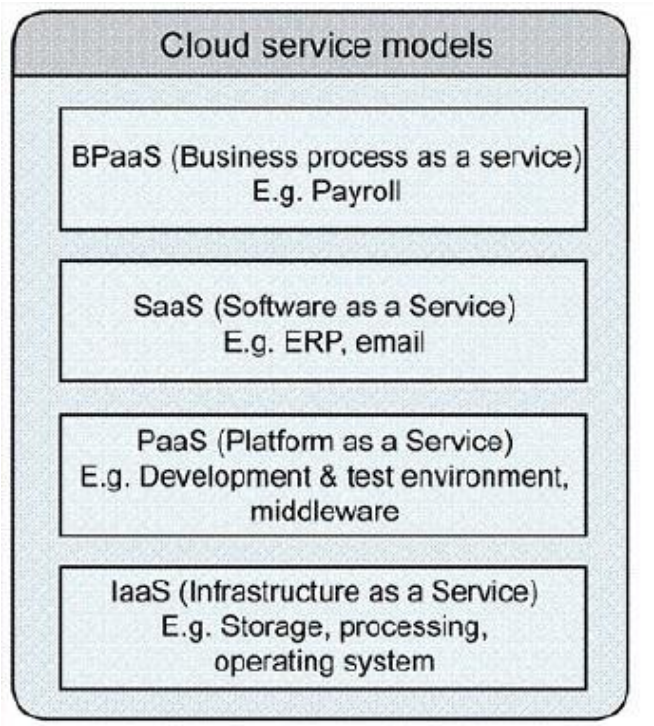


Figura 2.8. BPaaS en la pila de modelos de servicio de Cloud Computing

Dado que BPM es una disciplina que ayuda continuamente a las organizaciones a optimizar los procesos operacionales que tienen un gran impacto en los objetivos de negocio, las organizaciones utilizan cada vez más los BPMS para modelar y ejecutar los procesos de negocios, administrar tareas humanas y de los sistemas, monitorear y auditar el rendimiento de los procesos de negocio y mejorarlos continuamente. Por esto, la integración de un BPMS en un modelo SaaS provee un ambiente flexible y a bajo costo, no solo para aquellas pequeñas y medianas organizaciones que desean tener una infraestructura de IT que pueda incrementar el nivel de servicio que se les brinda a sus clientes, sino también para organizaciones de gran magnitud que se ven envueltas en grandes costos de instalación, despliegue, mantenimiento y actualización de su infraestructura IT.

Básicamente, este modelo de servicio necesita de dos componentes principales para el desarrollo y ejecución de sus procesos, que son utilizados para lograr las funcionalidades de un *workflow* tal como se ejecutaría en un sistema embebido:

- BPMS (*Business Process Management System*): herramientas para diseñar, ejecutar, monitorear y optimizar los procesos de negocios.
- BAM (*Business Activity Monitoring*): herramienta para monitorización de ejecución de procesos de negocio en tiempo real.

Sin embargo, una desventaja en la subcontratación de la ejecución de los procesos de negocio, es que los usuarios pierden el control de los datos sensibles que hacen al funcionamiento de sus organizaciones. También, al mantener toda la aplicación en la nube, se estaría desaprovechando el poder de cómputo del *cloud* cuando las actividades de los procesos que se despliegan en ese entorno no requieren de alta capacidad en su performance pero sí requieren un frecuente acceso a datos, de esta manera se intensifica el tráfico y hace más costoso el servicio.

Una de las soluciones a estas desventajas que se han nombrado anteriormente, y que se ampliará en el Capítulo 3, es crear un sistema híbrido, formado por un motor de procesos de negocio en el cloud y otro local, en el cuál los procesos se descomponen en dos tipos de actividades:

1. Aquellas actividades que requieren mayor poder de procesamiento y no manejan información sensible.
2. Aquellas actividades que manejan información sensible, o gran volumen de información.

Al descomponer los procesos de negocio en estos dos tipos de actividades podemos situar las actividades del punto 1 en la nube, y las del punto 2 en un BPMS local. [11] [13]

Para realizar esta descomposición se tienen que tener en cuenta varios aspectos de la estructura y funcionalidad de los procesos de negocio, y se tratarán a continuación en el próximo capítulo.

Capítulo 3 - **Descomposición de procesos y monitoreo distribuido**

En este capítulo se dará una introducción al uso de BPMS en la nube, para luego dar un marco teórico en la combinación de un esquema embebido y un esquema en el *cloud* a partir de un modelo PAD (Proceso – Actividad – Datos). A continuación de la combinación de ambos esquemas se explicará cómo es posible realizar la descomposición de procesos en dos o más subprocesos utilizando un quinto patrón derivado del modelo PAD y cómo determinar la distribución óptima de las actividades que aprovecharían las ventajas de un sistema embebido y un sistema en el *cloud*. Por último, se introduce al lector en el monitoreo de procesos de negocios, la importancia que tiene dentro de un BPMS y los desafíos que requieren realizar la tarea una vez que se llevó a cabo la descomposición de procesos (tema que se tratará con más profundidad en el Capítulo 5).

3.1. BPMS en ambientes distribuidos

Si bien los BPMS basados en *Cloud Computing* pueden ayudar a las pequeñas y medianas organizaciones a mejorar la eficiencia de los procesos y reducir sus gastos de infraestructura, todavía existen algunas barreras al utilizar este servicio.

Los dos temas principales de estudio en la adopción y del uso de un BPMS en el *cloud* son:

- **Protección de la privacidad**: algunos datos gestionados por los procesos de negocios pueden ser confidenciales y las organizaciones no están dispuestas a correr el riesgo de un posible robo de información. Si bien los servicios de *cloud* se encuentra ocultos detrás de barreras de seguridad provistas por la infraestructura de red y tienen un avanzado mecanismo para mantener la división de información entre sus clientes, los dueños de la información sensible temen perder el control de estos datos, o puede ocurrir que por reglas contractuales con sus clientes no se les permita almacenar información de éstos en otros lugares que no sean servidores propios.
- **Actividades que no son altamente computacionales**: por otro lado, la eficiencia y efectividad de las actividades que no son altamente computacionales puede disminuir debido a que la transferencia de los datos puede tomar más tiempo que el procesamiento mismo de los datos. Además, los costos de la actividad pueden incrementarse debido a que la transferencia de los datos es uno de los elementos de facturación en un sistema de *cloud computing*.

Hoy en día, en la mayoría de las soluciones donde se utilizan BPMS, el motor de proceso, las actividades y los datos de los procesos se encuentran alojados en un mismo sitio, ya sea en un sistema embebido o en un ambiente de *cloud computing*, donde presentan las ventajas e inconvenientes de cada uno de estos. Para poder utilizar los principales beneficios de ambos sitios, esto es, privacidad de datos en un sistema embebido, y poder de cómputo y ubicuidad relacionado a un sistema de *cloud*, se ha investigado en [14] un modelo de distribución, denominado PAD (Proceso-Actividad-Datos), en el que el arquitecto de procesos de negocio puede separar uno de estos procesos de acuerdo a los beneficios que brindan cada una de estas ubicaciones. Los ambientes

distribuidos en el contexto de los procesos de negocio favorecen el rendimiento y proponen incorporar el concepto de descomposición de procesos, permitiendo que los mismos se ejecuten tanto en un entorno *cloud* o embebido. [10][11][13][14]

3.2. Combinación de esquema embebido y cloud

Al diseñar una arquitectura descentralizada basada en un BPM *cloud* y uno embebido, se debe tener en cuenta en dónde desplegar los procesos, dónde ejecutar las actividades, como así también donde almacenar los datos producidos y consumidos por las aplicaciones. Para analizar la combinación entre un esquema embebido y uno de *cloud* se plantean tres aspectos: arquitectura, control de flujo y optimización de la distribución. [13] [14]

- **Arquitectura**: en la mayoría de las soluciones BPM el motor de procesos, las actividades y los datos del proceso se localizan en el mismo lado, tanto en un sistema embebido como en el *cloud*. Existen investigaciones que introducen el modelo PAD (Proceso-Actividad-Datos) de la Figura 3.1 como una posibilidad de distribución de BPM en el *cloud*. En el mismo el motor de procesos, las actividades involucradas en el mismo y sus datos están distribuidos.

El modelo PAD define cuatro posibilidades de distribución:

1. El primer patrón de solución es el enfoque tradicional de BPM donde todos los elementos están alojados en el usuario final.
2. El segundo patrón es útil cuando el usuario ya tiene un sistema BPM, pero las actividades con intensidad de cómputo se localizan en el *cloud* para incrementar su performance.
3. El tercer patrón es útil para los usuarios que aún no poseen un sistema BPM; en este caso podrían adoptar un esquema de *cloud* de manera de pago por uso, donde las actividades sin intensidad de cómputo y los datos sensibles se pueden localizar en el usuario final.
4. El cuarto patrón es el modelo basado en *cloud* donde todos los elementos se localizan en la nube.

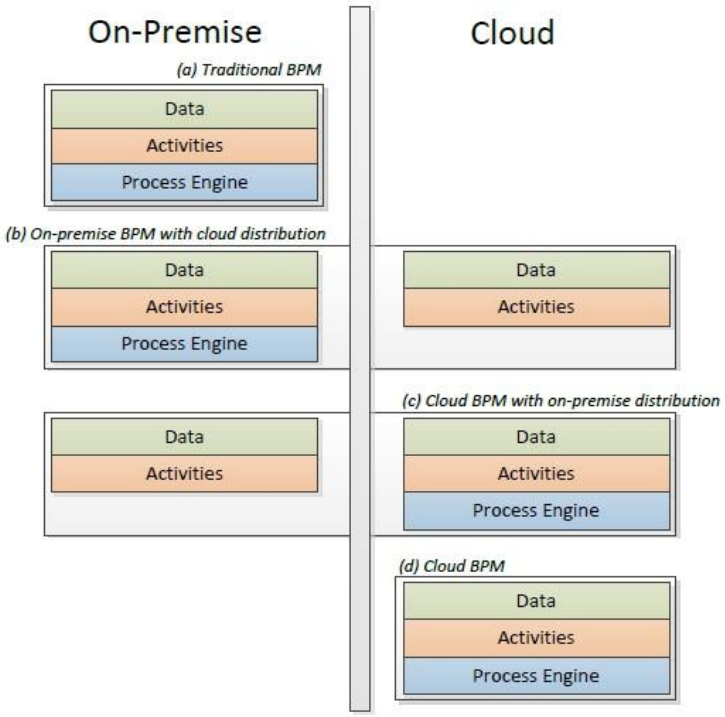


Figura 3.1. Esquema de distribución Proceso-Actividad-Datos [11][14]

- **Control de Flujo**: Los procesos de negocio consisten de dos tipos de flujos: de control y de datos. Los flujos de control regulan las actividades que se ejecutan y cuál es la siguiente actividad en ejecutarse, mientras que los flujos de datos determinan cómo los datos se envían de una actividad a la otra dentro del proceso, y cómo se realiza el mapeo de estos datos. Los motores de BPM deben lidiar con el control de ambos flujos. Un flujo de datos puede contener datos sensibles, por lo tanto, cuando se despliega un motor de BPM en el *cloud*, se debe proteger el contenido de los mismos.

Un ejemplo de arquitectura propuesta sería aquella en que el motor del lado del *cloud* solo lidia con flujos de datos usando identificadores de referencia en vez de datos reales. Cuando una actividad necesita datos sensibles, la transferencia de los datos a la actividad se maneja bajo supervisión del usuario dentro de un túnel de encriptación. Los datos sensibles se almacenan en el lado del usuario final, y los datos no sensibles se almacenan en el *cloud*. Este esquema permite que los datos sensibles no viajen indiscriminadamente a través de la web.

- **Distribución óptima**: los costos de un sistema de *cloud* han sido propósito de estudio en diversos artículos tales como [18] y [19], en los cuales se investiga a los proveedores de *cloud* como Amazon, Google, eBay, entre otros, y se plantea como encontrar el balance de costo y rendimiento utilizando una aplicación que se basa en *cloud computing* para la realización de las tareas que requieren mayor poder de procesamiento.

En el caso de los procesos de negocio existen distintas fórmulas para calcular la distribución óptima de las actividades, y determinar si las mismas deben ubicarse en la nube o en un sistema embebido. El cálculo toma en cuenta los costos de tiempo, los costos monetarios y los costos por el riesgo de privacidad. En la Sección 3.4, luego de realizar la introducción de la descomposición

de los procesos, se presentaran algunos casos de los tipos de distribución óptima de las actividades en un sistema embebido y un sistema *cloud*.

3.3. Descomposición de procesos

Es posible generalizar la distribución PAD e identificar un quinto patrón en el cual el motor de procesos, las actividades y los datos se despliegan en la nube y en el usuario final, tal como se puede observar en la Figura 3.2.

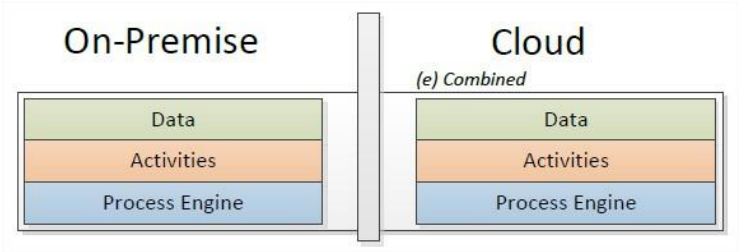


Figura 3.2. Quinto patrón de distribución PAD [11]

Esta solución presenta dos beneficios potenciales:

1. El motor de procesos regula el flujo de control y el flujo de datos. Una actividad recibe datos del motor de procesos y luego de su ejecución los datos que son producidos se pasan de nuevo al motor de procesos.
2. Cuando la nube no se encuentra accesible, los usuarios pueden ejecutar los procesos de negocio en forma completa en el sistema embebido hasta que el primero vuelva a estar disponible.

Para observar la diferencia entre un motor de procesos situado en un solo lugar, como por ejemplo en el patrón 2 del esquema de distribución PAD, y motores de procesos situados en ambos sitios, veamos el siguiente ejemplo.

Consideremos que una secuencia de actividades se ubica en la nube, mientras que el motor de procesos se despliega en el usuario final. Cada actividad utiliza los datos de salida de la actividad previa como entrada. Los datos no se envían directamente de una actividad a la otra sino que son enviados al motor de proceso primero (Figura 3.3a).

Debido a que la transferencia de datos es uno de los factores de facturación en el modelo de *cloud computing*, estas situaciones pueden volverse más caras cuando se transfieren grandes cantidades de datos entre actividades.

Para evitar este problema se puede agregar un motor de procesos al *cloud*, el cual regula el flujo de control y el flujo de datos entre las actividades situadas en él (Figura 3.3b). Cuando una secuencia de actividades se localiza en el *cloud*, los datos se regulan por el motor de procesos en el *cloud*, lo cual reduce la cantidad de datos a ser transferidos entre este y el sistema embebido.

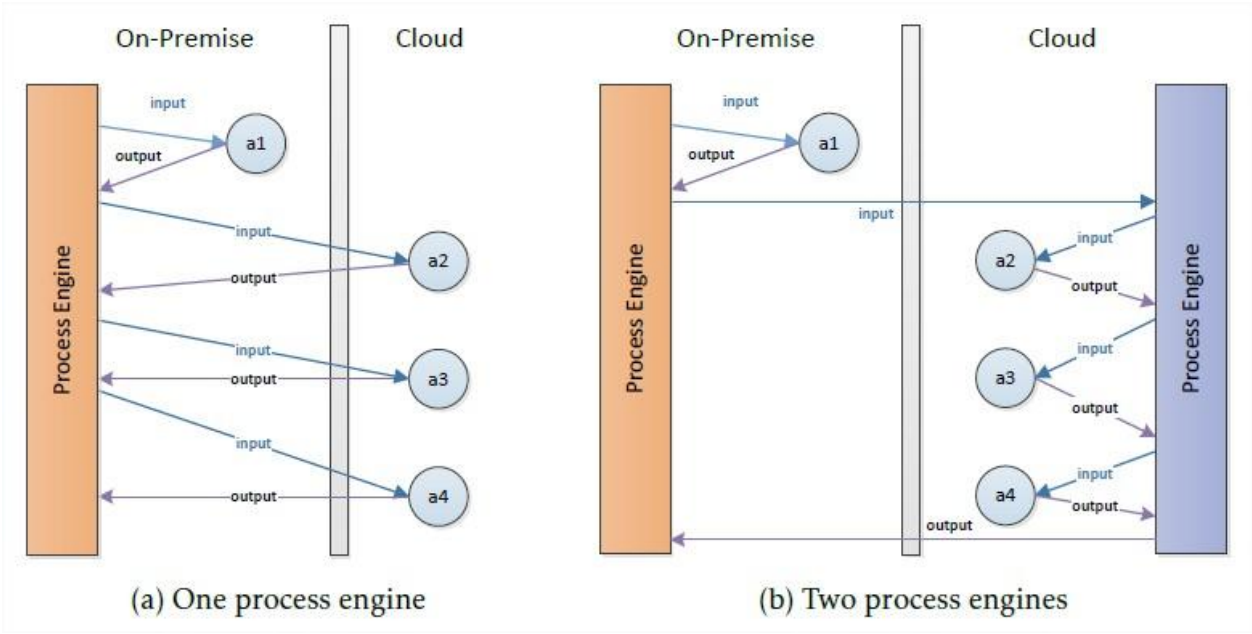


Figura 3.3. Datos enviados entre actividades coordinadas por motores de procesos [11]

Para poder correr un mismo proceso de negocio en dos motores de proceso separados, el mismo debe ser dividido en dos procesos individuales. Puede llegar a ser conveniente para los usuarios del BPMS tomar una lista de distribución del proceso de negocio y sus actividades, la cual puede ser automáticamente transformada en dos procesos de negocio, uno en el *cloud* y otro en el sistema embebido. [10][11][13]

Una aproximación posible para manejar la descomposición del proceso es identificar la estructura y la semántica del mismo (Figura 3.4). Esto significa que se deben identificar aquellas actividades del proceso de negocio en donde la ejecución de las mismas dan la pauta de atomicidad, y que su ejecución, ya sea en el sistema embebido o en el *cloud*, no modificaría el comportamiento del proceso de negocio original. Una estrategia para detectar aquellas actividades atómicas es tratar de observar las dependencias de control y de datos que tienen con otras actividades y con el motor de procesos en el cuál se ejecutan. Por ejemplo, si existe una actividad que utiliza información confidencial que obtiene del sistema embebido, lo más probable es que no sea recomendable ejecutar la actividad en el motor de procesos situado en el *cloud*, ya que correríamos peligro de exponer tales datos. Al identificar las dependencias de control y de datos, se pueden investigar las consecuencias de mover ciertas actividades del sistema embebido al *cloud* y viceversa.

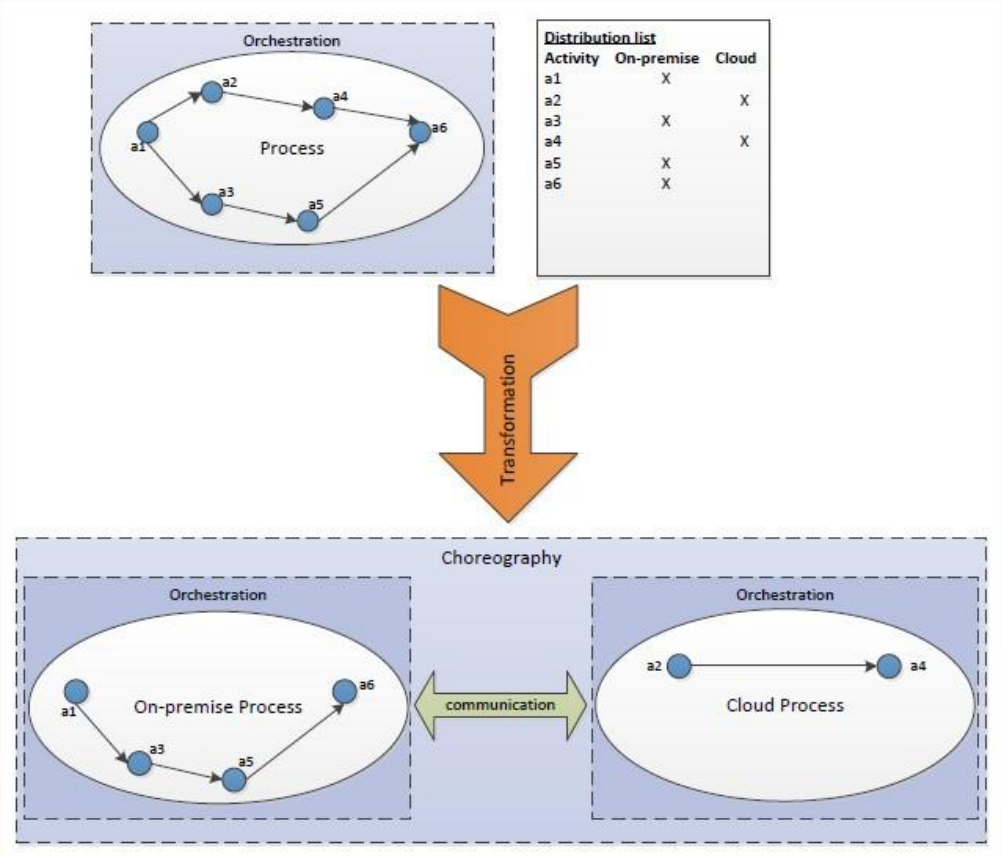


Figura 3.4. Descomposición de procesos[11]

Cuando se conocen las consecuencias de la distribución de actividades, se puede crear una transformación de modelo en la cual un proceso de negocio y una lista con marcas se usan para crear dos procesos individuales, uno para el *cloud* y otro para el usuario final. Además, se puede generar una descripción de la coreografía para describir la comunicación entre ambos procesos de negocio utilizando algún lenguaje estándar.

3.4. Distribución óptima de las actividades

Evitar que los datos sensibles se transfieran entre el sistema embebido y sistema de *cloud* es un método directo y eficaz para proteger la privacidad de los datos de los usuarios. Sin embargo, a veces, los usuarios quieren utilizar los recursos en ambos sitios para poder aprovechar el alto procesamiento y la capacidad de almacenamiento masivo de la nube, sobre todo si los datos que intervienen en el proceso no requieren un alto nivel de privacidad.

Existen varios esquemas acerca de la distribución de una actividad y los datos que maneja. En la Figura 3.5 se pueden ver ocho esquemas básicos de distribución de una actividad, y sus entradas y salidas de datos en ambos entornos. La parte superior de cada rectángulo representa la distribución en el *cloud*, y la inferior representa la distribución en el sistema embebido. El símbolo A representa a la actividad, mientras que los símbolos I y O representan los datos de entrada y salida respectivamente. Si bien las actividades pueden tener varias entradas y salidas de datos de diferentes

lugares, se presentan los casos básicos de ejemplo para lograr una mejor compresión de la distribución.

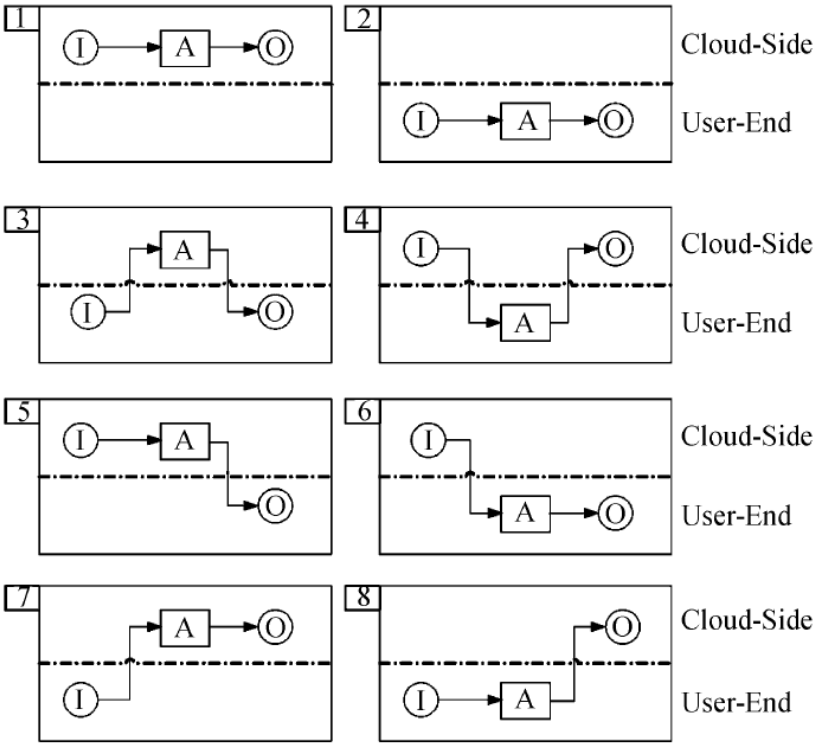


Figura 3.5 Esquemas de distribución de actividades en un sistema distribuido [14]

Para poder obtener el mayor beneficio correspondiente al despliegue de los subprocesos de negocios en ambos sistemas, es necesario determinar el costo que supondrá situar las diferentes actividades que componen al proceso.

Por lo general, un sistema embebido tiene un poder de cómputo relativamente menor aunque un alto nivel de privacidad. Un sistema de *cloud* tiene mejor rendimiento computacional pero un bajo nivel de privacidad. Mientras que el sistema embebido tiene un costo fijo, los usuarios deben pagar el servicio de *cloud* por los datos transferidos y las horas de CPU consumidas. Por esto, para determinar la distribución óptima de las actividades es necesario calcular los costos del sistema BPM en el *cloud* de acuerdo los costos de tiempo, los costos monetarios y los costos por el riesgo de privacidad.

- **Costo de Tiempo**: el tiempo de ejecución de una actividad en el *cloud* suele ser menor que en el sistema embebido, especialmente cuando se realizan tareas de procesamiento intensivo ante requerimientos concurrentes de gran tamaño. Sin embargo, en algunas circunstancias, cuando la ejecución en el *cloud* no requiere demasiado procesamiento, la transmisión de los datos podría ser mayor que el procesamiento en el *cloud*. Por esto, para lograr el mayor beneficio de costo de tiempo, se debe buscar un equilibrio entre los datos enviados y el procesamiento en el *cloud*, tomando como referencia los resultados que se lograrían al procesar la misma actividad en el sistema embebido.
- **Costo Monetario**: dado que el procesamiento, tanto como la transferencia desde y hacia el *cloud* son cobrados por los proveedores en término de horas de CPU utilizadas y GB (*GigaBytes*) de datos transferidos, hay que estar seguros de no perder dinero

ejecutando las actividades en el *cloud* cuando se podría ejecutar en el sistema embebido sin mucha diferencia en el procesamiento.

- **Costo de Privacidad**: la privacidad de los datos es otro tema a considerar. Las organizaciones, al situar datos sensibles o documentos privados en el *cloud*, deben estudiar el riesgo de impacto que supondrán la fuga de tal información, y aceptar o rechazar tomar dichos riesgos.

El estudio de las fórmulas para calcular los costos anteriormente mencionados exceden el alcance de esta tesina, pero si se desea profundizar en el tema, en el artículo de la referencia [14] se hace mención a cómo puede calcularse el costo mínimo global de un sistema BPM distribuido a partir de los costos de tiempo, monetario y de privacidad de los datos.

Mediante el uso de estas fórmulas los usuarios pueden hacer una estimación de los costos de desplegar partes de sus aplicaciones en un sistema embebido y en la nube. Debemos recordar que un criterio clave desde el punto de vista económico en un sistema de *cloud* es la tasa de transferencia de datos desde y hacia el servidor. [14]

3.5. Monitoreo de procesos de negocio distribuidos

En las soluciones orientadas a procesos de negocios existen dos objetivos principales. Primero, la capacidad de proveer flexibilidad en los procesos de negocio, permitiendo una rápida reacción a los cambios del mercado. Y segundo, proveer una buena visualización de los procesos y permitir una optimización eficiente de estos.

Para poder optimizar los procesos de negocios tenemos que saber qué es lo que está funcionando mal o es ineficiente, por lo tanto, debemos comenzar a medir los procesos. Es aquí donde se puede utilizar una herramienta de monitoreo llamada Monitoreo de Actividades Empresariales, o en inglés, Business Activity Monitoring (BAM). Esta herramienta que se encuentra dentro de muchos BPMS nos permite medir varios aspectos de los procesos de negocio tales como tiempo, costos, rendimiento, estados y resultados de operaciones, procesos y transacciones.

La optimización de los procesos de negocio es una tarea difícil y requiere de un buen conocimiento de los procesos de negocio dentro de la organización, por esto, aunque se cuente con la mejor herramienta de monitoreo, la optimización de los procesos de negocio dependen de las habilidades de la persona responsable de estos. Las personas encargadas de optimizar los procesos de negocio tienen que tener en cuenta varias tareas tales como identificar aquellas actividades que están teniendo poca eficiencia (ya sea en tiempo, costo o calidad); determinar los factores internos y externos que afectan la decisión de optimizar un proceso y como éstos influyen con el dueño del proceso, los dueños de las tareas y las áreas específicas de la organización; aplicar reingeniería, buenas prácticas o rediseño del proceso; definir nuevas medidas de rendimiento del nuevo proceso (*KPIs – Key Performance Indicator*), realizar simulaciones y monitorearlos para confirmar que ha sido mejorado.

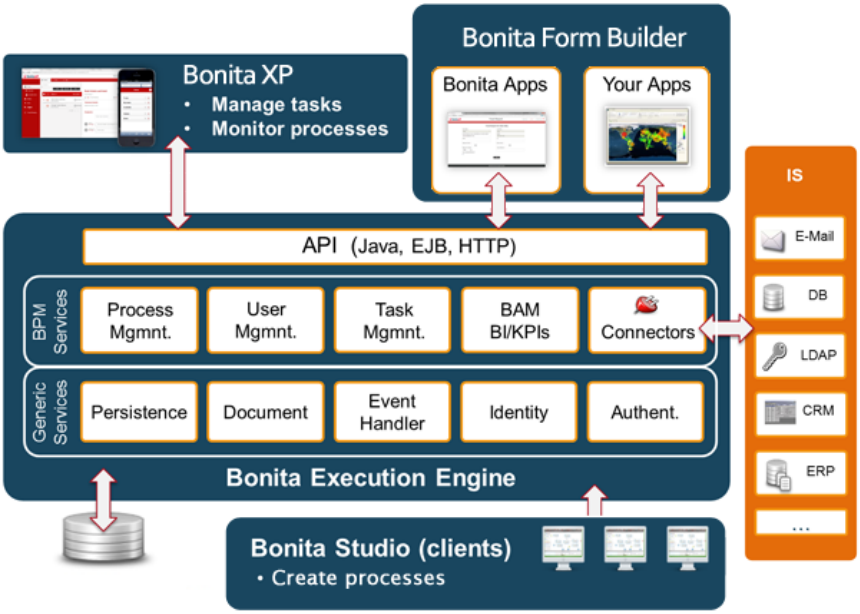


Figura 3.6. Arquitectura de Bonita OS – Version 5.x [16]

BAM provee acceso a información de los procesos en tiempo de ejecución, permite un análisis en tiempo real de los procesos de negocio, muestra los cuellos de botella en las tareas, mide el tiempo de cada tarea y provee herramientas para visualizar toda esa información [15]. Además, se utiliza para asegurar que los procesos de negocio funcionan como es esperado, simplificar la información compleja relacionada a los procesos y mostrarla oportunamente.

Como se ha dicho anteriormente, muchos BPMS contienen módulos BAM para realizar el monitoreo de los procesos de negocio que se están ejecutando en el motor de procesos. Por ejemplo, en la Figura 3.6, podemos ver la arquitectura de Bonita Open Solution [17], un BPMS libre y *open source* (que veremos en detalle en el Capítulo 4), en la cual observamos los distintos módulos que componen a la suite, entre ellos el módulo BAM de monitoreo. Sin embargo, estos módulos de monitoreo funcionan solo dentro del BPMS donde se están ejecutando los procesos. Por esto, en el caso de la descomposición de procesos, el mayor de los problemas de poseer un esquema de procesos particionados, es la recuperación y monitoreo de las distintas instancias distribuidas (ya sea en un sistema embebido o dentro del *cloud*), y a su vez lograr dar un esquema integrador de las mismas bajo la óptica del "proceso original" al cual pertenecen.

En la presente tesina se propone resolver este problema de monitoreo y presenta en el Capítulo 5 una solución que luego será respaldada por una aplicación de ejecución y monitoreo de los procesos de negocio distribuidos.

Capítulo 4 - **Bonita Open Solution y su capacidad de extensión**

En este capítulo se dará una introducción al BPMS Bonita Open Solution [21], una suite para la gestión de los procesos de negocio, profundizando en los componentes, APIs, y conectores que la componen. Dado que se caracteriza por ser un BPMS libre y *open source* basado en el lenguaje de programación Java [20], veremos cómo es posible utilizar a estos componentes para extender la funcionalidad de la misma y poder ejecutar procesos de negocio distribuidos.

4.1. Bonita Open Solution

Bonita Open Solution (BonitaOS) es una herramienta libre y *open source* que ayuda a los usuarios a crear aplicaciones basadas en procesos de negocio. Ha sido creada por BonitaSoft [21], una de las empresas que son líderes en el mercado de la gestión de procesos de negocio. Al estar completamente desarrollada en Java y ser *open source*, se puede hacer uso de esta herramienta para realizar nuestros propios desarrollos sin la necesidad de realizar contratos aparte con la empresa BonitaSoft.

Según Gartner Inc. [27], una de las empresas líder en consultoría e investigación de las tecnologías de la información, dice que "*Bonita BPM es el único producto de código abierto que cumple con la definición de solución BPMS de Gartner*". Además, la suite fue reconocida con los premios "*Cool Vendors in Business Process Management, 2011*", por Gartner, y "*The Best Modeling Tool*" por la comunidad de Eclipse. [28]

Por otra parte, BonitaSoft ofrece varias versiones de pago llamadas *Subscription Pack* (SP), las cuales poseen utilidades adicionales que agilizan la producción y permiten un control más exhaustivo de los procesos de negocio con el objetivo de ayudar a los desarrolladores en sus tareas (Figura 4.1). Entre algunas utilidades adicionales se pueden nombrar: duplicado de formularios, desarrollo colaborativo, funciones *Ajax* integradas, gestión del *Business Activity Monitoring* (BAM), control del *FrontEnd*, etc.

Además de ser una aplicación potente y ligera, se destaca sobre todo por la facilidad en su utilización debido al diseño intuitivo de los diferentes elementos que lo componen, y por el bajo costo en su implantación dentro de una organización, ya que se requiere solamente tiempo para aprender a utilizarla.

Este programa está dirigido a personas en la industria de los negocios, con el fin de mejorar el flujo de trabajo de los proyectos de la organización, y debido a sus características, es capaz de superar todos los huecos y errores que se pueden producir cuando existen varias personas involucradas en un proyecto.

BonitaOS tiene soporte para *Business Process Model and Notation* (BPMN) versión 2.0, lo cual ofrece a los usuarios la posibilidad de diseñar sus proyectos de una manera transparente, ágil y con la capacidad de expresar un proceso de negocio de una forma que sea comprensible para el público comercial y el público técnico de igual manera. Además, los usuarios tienen la posibilidad de

realizar diferentes ajustes a sus proyectos desde una perspectiva general, pudiendo modificar la apariencia de los procesos de negocio, y hasta incluso simular la ejecución de los mismos para poder obtener los costos de cantidad y costo por uso o por tiempo consumido.

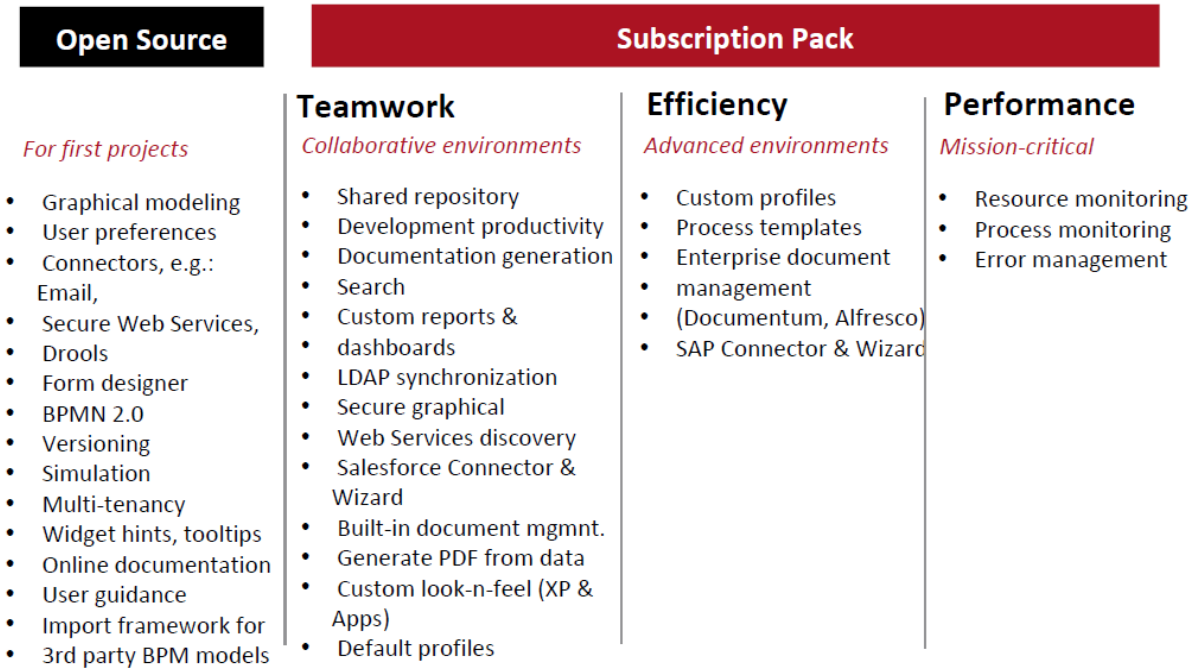


Figura 4.1. Diferencias entre la versión libre y los Subscription Pack de BonitaOS [23]

Una importante ventaja de la aplicación es que soporta varios formatos de archivos (proc, bar, BPMN, XPDL y jBPM) para la importación de procesos de negocio, pudiendo importar procesos de negocio incluso de aplicaciones BPMS similares a BonitaOS que utilicen la notación BPMN 2.0, como puede ser Bizagi [22]. Además permite realizar la exportación de los diseños de los procesos y guardarlos en diferentes formatos para poder traspasarlo de un sistema a otro o guardarlos como copias de seguridad, siendo cada proceso independiente a la plataforma, así como también exportar los procesos a formatos de archivo de documentos o imágenes tales como pdf, jpeg, png, bmp, gif y svg, para poder visualizar los procesos sin necesidad del uso de la suite.

Un BPMS como Bonita, además de ofrecer las ventajas propias de su orientación a procesos, presenta otras ventajas en cuanto al desarrollo de aplicaciones basadas en procesos. El hecho de poder modificar el modelo en BPMN y automáticamente modificar toda la aplicación con los nuevos cambios, de forma que cualquier cambio en la lógica de negocio no suponga modificar toda la aplicación, es una cualidad muy beneficiosa, ya que esto reduce enormemente los tiempos de desarrollo y testeo para el mantenimiento y adaptación de las aplicaciones. Con esta capacidad de los BPMS abandonamos la visión monolítica de las aplicaciones basadas en plataformas y abrimos el camino para trabajar en la mejora continua de los procesos. Además, es posible gestionar versiones de los procesos, pudiendo reutilizar procesos ya existentes para la generación de nuevos procesos adaptados o mejorados. [21][23]

4.2. Módulos de Bonita OS

La aplicación Bonita Open Solution en su versión 5.x está compuesta de varios módulos o componentes, como puede verse en la Figura 4.2, o más detalladamente en la Figura 3.6 del capítulo anterior. Los módulos que se destacan son Bonita Studio, Bonita Form Builder, Bonita Execution Engine (BEE), y Bonita User Experience (User XP).

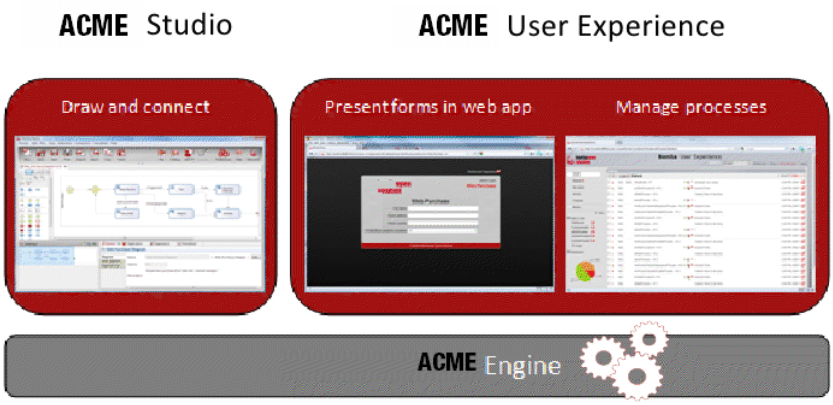


Figura 4.2. Módulos que componen a Bonita Open Solution [23]

4.2.1. Bonita Studio

Es una interfaz grafica cuya función es diseñar los procesos BPM usando la notación BPMN sobre un área de diseño de forma muy intuitiva basada en arrastrar los elementos y en su configuración específica mediante una o varias pestañas habilitadas para ello. Desde esta interfaz gráfica es posible crear nuevos procesos, abrir procesos existentes o importarlos. Dispone de varias barras de herramientas para la creación, ejecución y configuración de los procesos diseñados, paletas de diseño para la construcción de los procesos de negocio, un área de trabajo en donde se diseñan los procesos y paneles de vista y detalles.

4.2.2. Bonita Form Builder

Es un módulo que se encuentra dentro de Bonita Studio. Es el módulo en el que se definen los formularios que habrán de ser rellenados por los usuarios. Como muchos de los pasos que se producen en un proceso BPM requieren de la entrada de datos por parte del usuario implicado, es necesario contar con este componente para el ingreso de la información que requiere cada una de las actividades dentro del proceso. Una característica que se puede encontrar dentro de la construcción de los formularios es la posibilidad de declarar validadores para el formulario entero o para un conjunto de campos del mismo, para que de esta manera se controlen los datos que se ingresan.

4.2.3. Bonita Execution Engine (BEE)

Es el motor de procesos de Bonita y se encarga de la conexión de los procesos que existen en el sistema, así también como el despliegue y ejecución de los procesos. El módulo de Bonita Studio

está conectado directamente a este otro módulo para funcionar. Este motor es genérico y extensible por lo que siempre seremos capaces de añadir con mayor o menor dificultad nuevos estándares o bien servicios que puedan aparecer en el mundo de BPM con posterioridad.

El módulo *Bonita Execution Engine* (BEE) es uno de los tres principales módulos de Bonita OS. Es un módulo no intrusivo, ya que solo requiere una *Java Virtual Machine* (JVM) para ejecutarlo. Es la base de procesamiento, la cual se ejecuta en tiempo real en *background* y conecta a Bonita Studio, los formularios creados en el *Form Builder*, al módulo *Bonita User Experience* y las aplicaciones externas que utiliza la organización.

Bonita Open Solution se basa en la BEE para crear, acceder y procesar los datos a través del uso de las diferentes interfaces de programación de las aplicaciones (APIs), además de gestionar y ejecutar los procesos creados en Bonita Studio.

La BEE, también conocida como *Runtime*, fue diseñada para proveer la máxima flexibilidad a través de la inyección de servicios. La BEE es completamente configurable utilizando un archivo XML llamado bonita-server.xml, que puede encontrarse en la carpeta de configuración del servidor. Este archivo de configuración describe todos los servicios utilizados por defecto, aunque es posible cambiar cualquiera de ellos o reemplazarlos por los que se ajusten con la implementación de la aplicación.

4.2.4. Bonita User Experience (User XP)

Es una aplicación encargada de desplegar y gestionar los procesos ya desplegados así como las instancias de cada proceso. Es muy intuitiva ya que su interfaz es similar a una aplicación de gestión de correo.

Bonita User Experience (User XP) provee una interfaz similar a un cliente de correo electrónico para gestionar los pasos, actividades y procesos. Tiene una vista de usuario que puede ser utilizada, por ejemplo, por un empleado dentro de la compañía que tiene la responsabilidad de responder a cierta solicitud para completar una parte del proceso, o por algún cliente o comprador.

Los usuarios finales que tienen que realizar alguna acción en un proceso pueden utilizar el User XP para observar aquellas tareas que están esperando ser completadas, ingresar o mirar los datos que fueron introducidos en los formularios, e interactuar con la lista de actividades en las cuales se encuentra involucrado.

4.3. Capacidad de extensión de Bonita

Como ya se ha dicho antes, Bonita Open Solution es open source y libre, por lo que es posible crear un repositorio y realizar una nueva rama (*branch*) para modificar el código y crear nuestra propia aplicación Bonita, para incluir funcionalidad que quizás tengamos en una aplicación que ya disponemos. Sin embargo, esta es una tarea que requiere mucho trabajo, además del conocimiento necesario que se debe tener de la arquitectura de Bonita.

Para evitar tener que modificar el código del motor de procesos, Bonita nos permite utilizar conectores predefinidos que se encuentran incluidos en la suite, o crear los propios de acuerdo a las

necesidades del proceso de negocio. Los conectores son componentes básicos utilizados para modelar las interacciones entre dos o más componentes de software. Los conectores funcionan en tiempo de ejecución y por lo general se comunican entre dos objetos, o entre un cliente y un servidor, mediante mensajes asincrónicos, *multicast* o *streams* de datos. Principalmente se utilizan para la gestión de información y para la ejecución de funcionalidades específicas que contienen los sistemas de información.

El motor de bonita es extensible, esto significa que puede ser usado con la plataforma que tienen Bonita por defecto, es decir con Bonita User Experience, o ser consumido como un EJB (Enterprise JavaBeans) o por HTTP a través aplicaciones externas haciendo uso de la API REST que nos otorga. De esta manera, es posible integrar los procesos de negocio que gestiona Bonita hacia un sistema de información o a una aplicación propia, y ejecutar los métodos que se encuentran en las diferentes APIs que contiene el motor de procesos de Bonita (BEE). A partir de la Sección 4.3.3 se explicará cómo está integrada la API REST, su definición, obtención y ejecución de los métodos que la integran.

4.3.1. Conectores de Bonita OS

Bonita Open Solution provee conectores para conectar una tarea (o actividad) o un proceso (pool) hacia sistemas de información externos. Los conectores toman una entrada específica (directamente como un valor que el usuario ingresó en la actividad o construidos en una expresión) y ejecuta código Java. Algunos conectores también reciben el resultado de la llamada realizada, que es útil para informar a la actividad si el proceso de efectuar dicha llamada resultó exitoso, o por ejemplo, si lo que se desea es obtener datos de una base de datos externa, el conector se encarga de devolver los datos a la actividad que ejecutó el conector y realizó la consulta. (Figura 4.3)

Una gran biblioteca de conectores también es un elemento esencial para la viabilidad a largo plazo de una solución de BPM. Al agregar conectores a las actividades se permite que los procesos de negocio se integren con aplicaciones o herramientas de la organización. Los conectores permiten separar el proceso de definición del proceso de implementación, tal como se verá más adelante. Una de las ventajas que los caracterizan es que permiten reutilizar código ya sea, utilizando conectores que ya se encuentren en la biblioteca, creando conectores propios, o bien descargando e instalando conectores creados por otros desarrolladores de la comunidad.

Estos conectores se utilizan para operar dentro del flujo de trabajo de BPM con sistemas de terceros que tienen distintos tipos de servicios y aplicaciones (como bases de datos, mensajería, ERP´s, ECM´s, *data warehouse*, CRM´s, etc.). De esta manera podemos encontrar conectores para MySQL, Oracle, MSSQL Server, Jasper, SAP SalesForce, Alfresco, Sugar CRM, etc.

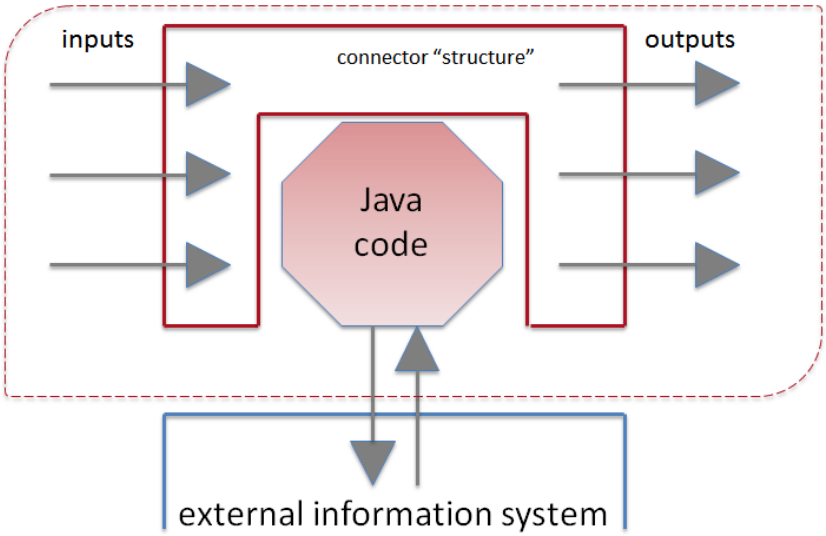


Figura 4.3. Estrucutra y ejecución de un conector en Bonita OS. [23]

Las empresas desarrolladoras de software que ofrecen sus productos con licencia propietaria, como por ejemplo Oracle BPM [29] o Intalio [30], introducen suscripciones pagas y otros obstáculos al proceso de creación de conectores de terceros debido a que desean maximizar sus ganancias. Esto contiene parcialmente a la comunidad de desarrolladores y, por consiguiente, limita el rango de las capacidades de la aplicación privada.

Por el contrario, los proveedores de BPM de código abierto como Bonita OS complementan las ofertas con las contribuciones de su base de usuarios y de la comunidad más grande para mejorar la versatilidad del producto. Los productos de BPM de código abierto como el conjunto de aplicaciones de BonitaSoft ya cuentan con una gran biblioteca de más de 100 conectores (Figura 4.4) listos para elegir y usar, entre los que podemos encontrar para:

- Base de datos como PostgreSQL, DB2, MySQL, MS SQL Server;
- Crear o eliminar ítems de un calendario de Google;
- Enviar correos a través de un servidor de correos como Gmail o Hotmail;
- Medios de comunicación social como Twitter y Facebook;
- Generación de reportes con la aplicación Jasper;
- Gestionar reglas de negocio con la aplicación Drools, entre muchos más.

Además esta selección está aún más enriquecida por las contribuciones que realizan los usuarios a la comunidad. Bastante a menudo, un usuario puede interactuar con la fuente de datos de la comunidad para la que no exista un conector actual. En lugar de solicitarle al proveedor del software que cree un conector para esa fuente de datos en una versión futura, el usuario tiene la opción de crear un conector y compartirlo con otros. [21]

Existen dos formas de incluir un conector en una actividad. Elegir un conector de los que ofrece Bonita OS que ya se encuentran instalados en la sección de conectores, o crear un conector propio y agregarlo.

Para agregar un conector de los que ofrece Bonita, es necesario seleccionarlo de una lista de conectores cargados que se encuentra separada por categorías. Para crear un conector nuevo, es

necesario indicarle a Bonita datos como, a qué categoría (nueva o existente) va a pertenecer el conector, el nombre que va a tener, la descripción del conector, el icono que va a tener, las entradas y salidas que se va a tener, e implementar el conector utilizando Java como lenguaje de programación.

Un conector es implementado en dos partes, la definición y la implementación. Esto permite modificar la implementación sin tener que modificar la definición, además, es posible crear varias implementaciones para una sola definición. La **definición** de un conector controla las interfaces externas del conector, es decir, las visibles a los usuarios y las visibles al motor de Bonita (las entradas y salidas). La **implementación** de un conector consiste en un archivo XML y una clase Java. Es posible crear varias implementaciones que correspondan a una definición, sin embargo, en un procesos solo existe una relación de uno a uno entre la definición y la implementación del conector.

El separar la definición de la implementación, permite que si el sistema externo cambia la interfaz a la que se está conectando el conector, lo único necesario para enlazar nuevamente el conector al sistema externo es cambiar la implementación del conector, sin tener que modificar los procesos de negocio, lo que ahorra muchos problemas y tiempo de redefinición de diseño.

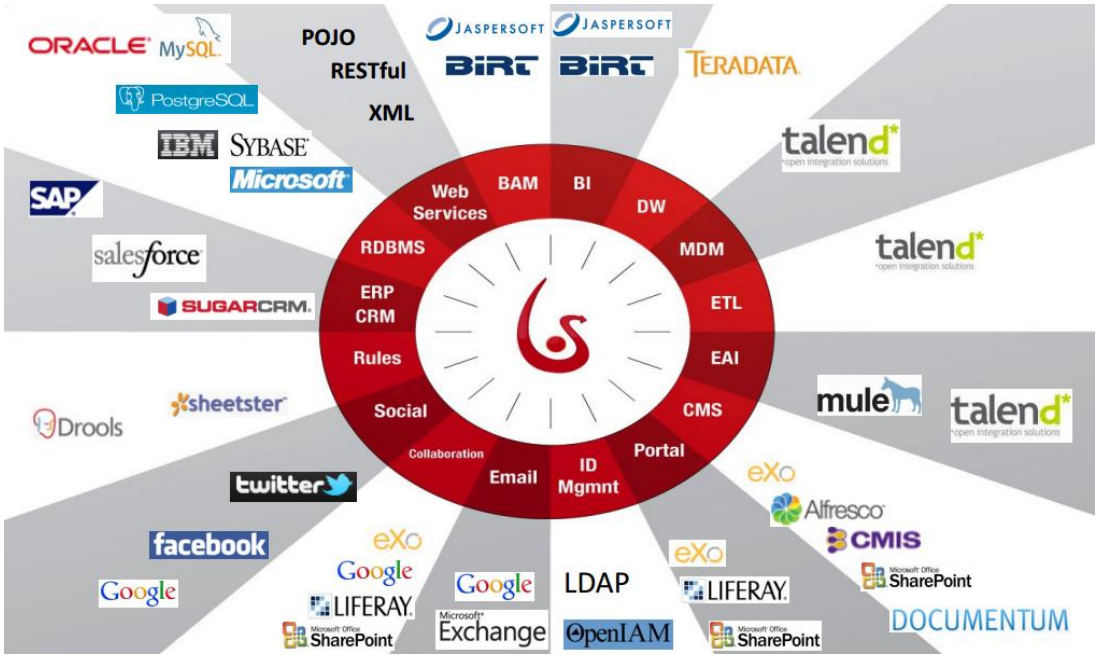


Figura 4.4. Diversidad de conectores en Bonita Open Solution [23]

Una vez creado el conector es posible agregarlo como si se tratase de un conector ya existente. Ambas acciones, agregar un conector como crearlo, se realiza a través de un asistente de configuración que dispone Bonita Studio.

Utilizando conectores, el proceso de negocio ya no tiene que adaptarse al sistema de información, sino, el sistema de información se debe adaptar al proceso de negocio. Con frecuencia la mayoría de procesos BPM enfrentan este problema de tener que forzar al sistema de información para evolucionar con respecto a los objetivos del proceso. Este es el resultado de la mejora continua de los procesos.

Con los distintos conectores que se disponen y con la facilidad para manejar los distintos módulos del BPMS de Bonita a la hora de diseñar e implementar los procesos, el desarrollar nuevas aplicaciones o adaptar los procesos actuales de una organización son proyectos totalmente realizables. Su adopción como herramienta de desarrollo, sólo dependerá de la implicación que la gente de TI tenga con los objetivos y metas de negocio. [21][23]

4.3.2. Creación de un conector en Bonita OS

Es bastante simple crear conectores en Bonita Studio. El único requisito es tener un conocimiento básico del lenguaje de programación Java, aunque con un poco de conocimiento de programación y el uso del asistente podría ser suficiente.

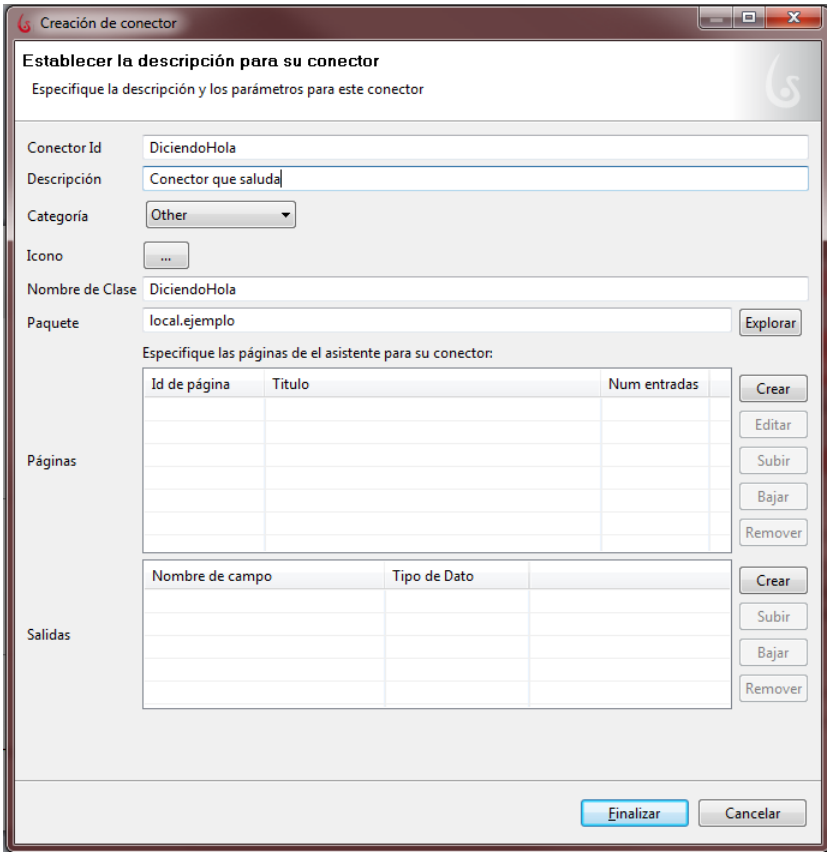


Figura 4.5. Asistente para la creación de un conector

Para crear un conector utilizando el asistente hay que ir al menú "Conector >> Nuevo conector…". Se abrirá una ventana donde se deberá ingresar información sobre el conector como: identificador, una descripción, categoría de conectores a la que va a pertenecer, y hasta es posible agregarle un icono para reconocerlo fácilmente en la lista de conectores. Luego, están las secciones de entrada ("Páginas") y salida ("Salidas") de datos. (Figura 4.5)

Para enviar datos al conector se deben crear "páginas". Al crear una "Página" se abre otro asistente para las entradas de información al conector (Figura 4.6). Las páginas sirven para separar los datos de entrada que tienen algún tipo de relación entre sí. Por ejemplo, si deseamos hacer una consulta a una base de datos, tenemos básicamente dos tipos de información, la conexión a la base de

datos (con datos como servidor, puerto, usuario y contraseña, etc.) y la consulta misma, por lo tanto, podemos crear dos páginas que contengan estos dos tipos de información y separar estos datos.

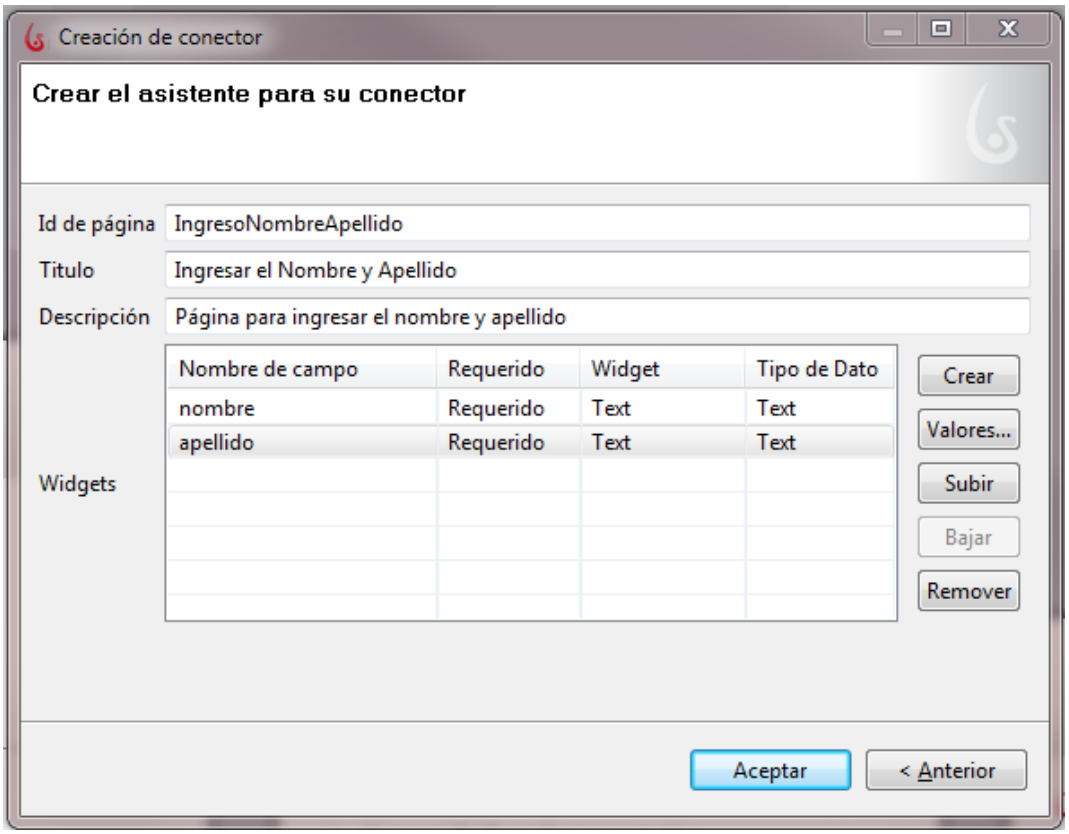


Figura 4.6. Asistente para la creación de una Página en un conector

Una vez que creamos las páginas, es necesario crear las variables de salida en donde se guardarán los datos que se generen luego de que se ejecute el código del conector.

En el ejemplo que se muestra en las figuras, el conector va a generar tan solo un texto de saludo. Las entradas serán el "nombre" y "apellido" de una persona, y el texto generado se guardará en una variable llamada "saludo". En la Figura 4.7 podemos ver como quedaría el asistente luego de crear el conector.

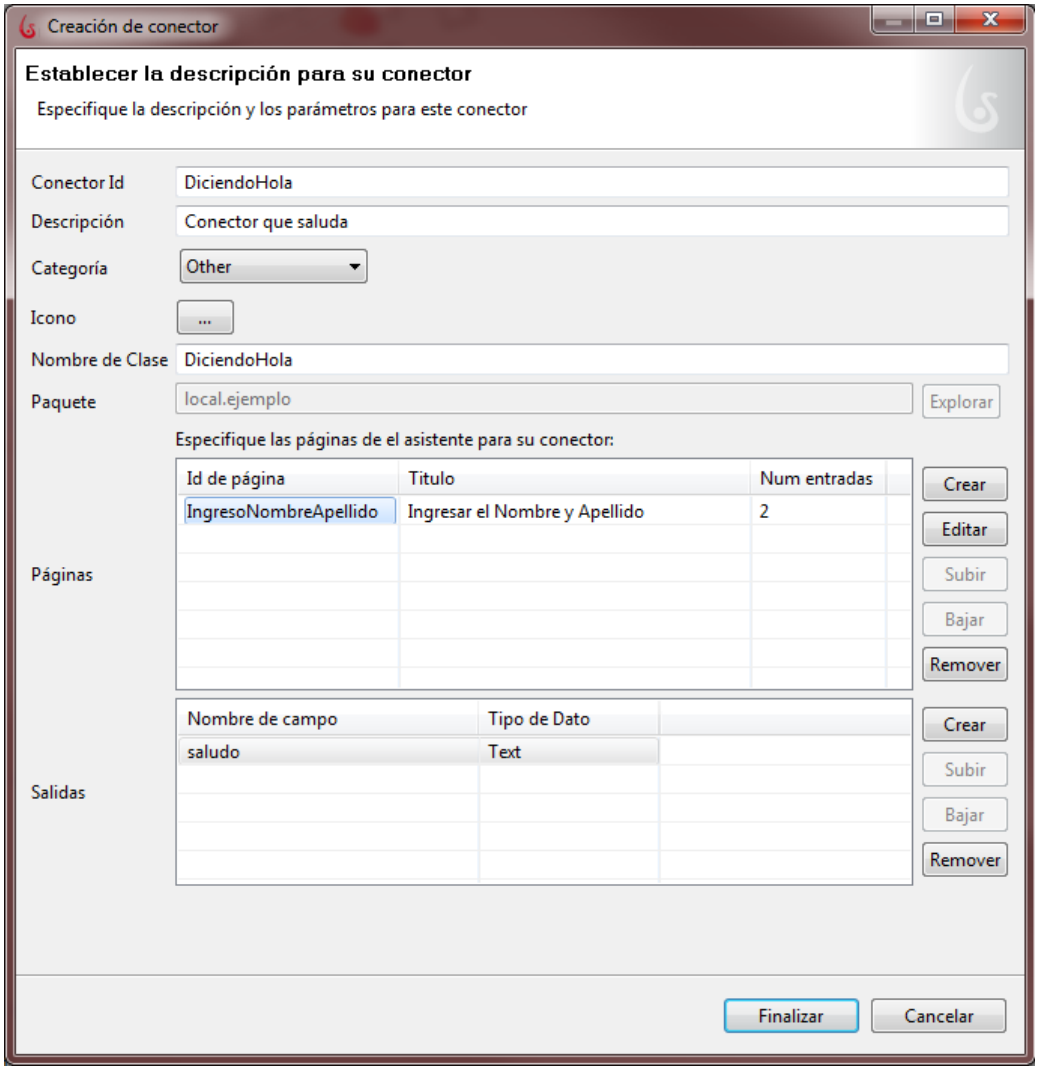


Figura 4.7. Creando el conector de ejemplo "DiciendoHola"

Luego de apretar en el botón "Finalizar", Bonita genera un bloque de código Java basado en las entradas y salidas ingresadas en el asistente, en donde incluye los *getters* y *setters* de estas variables. Cada conector extiende de la clase *ProcessConnector* y tiene al menos dos métodos que son los que se debemos sobrescribir, estos son *executeConnector* y *valídateValues*.

ExecuteConnector es el método que realiza el trabajo del conector. En el caso del ejemplo, debemos concatenar las variables apellido y nombre y agregarle un "Hola" al inicio de esta.

ValidateValues es un método que debería validar las entradas. Por ejemplo, se podría validar que el nombre y el apellido contengan sólo letras.

En la Figura 4.8 es posible visualizar el código fuente del conector, el cuál contiene los métodos anteriormente mencionados, junto con los *getters* y *setters* de las variables.

Luego de salvar el archivo ya es posible probarlo yendo al menú "Conectores >> Probar un Conector", o utilizarlo dentro de cualquier actividad del proceso de negocio.

DiciendoHola.java

```java
package local.ejemplo;

import java.util.List;
import org.ow2.bonita.connector.core.ConnectorError;
import org.ow2.bonita.connector.core.ProcessConnector;

public class DiciendoHola extends ProcessConnector {
    // DO NOT REMOVE NOR RENAME THIS FIELD
    private java.lang.String apellido;
    // DO NOT REMOVE NOR RENAME THIS FIELD
    private java.lang.String nombre;
    private java.lang.String saludo;

    @Override
    protected void executeConnector() throws Exception {
        this.saludo = "Hola " + this.nombre + " " + this.apellido + ", cómo estás?";
    }

    @Override
    protected List<ConnectorError> validateValues() {
        // TODO Auto-generated method stub
        return null;
    }

    /**
     * Getter for output argument 'saludo'
     * DO NOT REMOVE NOR RENAME THIS GETTER, unless you also change the related entry in the X
     */
    public java.lang.String getSaludo() {
        // TODO Add return value for the output here
        return this.saludo;
    }

    /**
     * Setter for input argument 'apellido'
     * DO NOT REMOVE NOR RENAME THIS SETTER, unless you also change the related entry in the X
     */
    public void setApellido(java.lang.String apellido) {
        this.apellido = apellido;
```

Figura 4.8. Código Java del conector "DiciendoHola"

4.3.3. APIs de Bonita

El motor de procesos de Bonita, es decir Bonita Execution Engine (BEE), es el responsable de la ejecución de los procesos. Todas las interfaces de usuario (User Experience, Formularios de las aplicaciones web, aplicaciones web personalizadas y las herramientas de administración) interactúan con el motor de procesos a través de sus interfaces de programación de aplicación (API) para gestionar los procesos de negocio.

Las diferentes interfaces que brinda la Bonita y el propósito de cada una dentro de la BEE son las siguientes:

1. **ManagementAPI**: responsable de la gestión de la definición de los procesos. Se utiliza para:
   - Instalar un proceso (*deploy* de un proceso).
   - Definir qué usuario puede comenzar un proceso
   - Ocultar un proceso del panel de usuario

2. **QueryDefinitionAPI**: se utiliza para acceder en sólo lectura a las definiciones de los procesos. Esta interface puede verse como un complemento de la interface QueryRuntimeAPI, y se ocupa de la parte estática de los datos gestionados por el *workflow*. Los datos del flujo de trabajo se pueden recuperar con los IDs o nombres de las entidades. Se utiliza para:
   - Listar todos los procesos desplegados en el servidor
   - Obtener las definiciones de todas las tareas de un proceso
3. **RuntimeAPI**: esta clase gestiona las definiciones de los procesos, las instancias de procesos ejecutados y el ciclo de vida de las tareas, así como también insertar, agregar y actualizar las variables dentro de la actividad o la instancia. Sirve para:
   - Comenzar un proceso. Es decir, iniciar una instancia.
   - Validar una tarea
   - Cambiar el valor de una variable de un proceso
4. **QueryRuntimeAPI**: utilizada para obtener información de las instancias de los procesos ejecutados. Las operaciones se utilizan en procesos, instancias de procesos, tareas. Entre lo que puede realizar se encuentra:
   - Obtener instancias ejecutadas
   - Obtener tareas afectadas a un usuario
   - Obtener el valor de una variable de un proceso
5. **IdentityAPI**: es la encargada de gestionar el directorio de usuarios de Bonita. Las funciones básicas son:
   - Crear un nuevo usuario
   - Agregar un rol y grupo a un usuario
   - Validar las credenciales de un usuario
6. **BAMAPI**: utiliza llamadas a indicadores claves de performance para obtener información sobre la ejecución de los procesos. El propósito de esta API es proporcionar una breve reseña de Bonita Business Activity Monitoring (monitoreo de las actividades de negocio de Bonita). El uso de esta API podría afectar a rendimiento del motor.
   - Obtener el tiempo de ejecución de una tarea
   - Obtener el tiempo necesario para finalizar una instancia
7. **CommandAPI**: es utilizada para inyectar código al motor de bonita. Utilizando esta interface se tiene acceso completo a todos los objetos utilizados por el motor de bonita. También puede ser utilizado para agregar características al motor. Otro uso que tiene es el de permitir a los desarrolladores escribir y ejecutar comandos dentro de la aplicación.
8. **WebAPI**: es utilizada solo para uso interno de Bonita User Experience. Nunca debería utilizarse.
9. **RepairApi**: es utilizada con el propósito de reparar y ajustar las instancias de los procesos.

De la misma forma que las interfaces de usuario interactúan (utilizando Java como lenguaje) con el motor de procesos (BEE) de Bonita a través de estas APIs, es posible también que una aplicación externa interactúe con el motor a través de HTTP utilizando una API REST que Bonita nos brinda.

4.3.4. Definición de la API REST de Bonita Open Solution

REST (*REpresentational State Transfer*) es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. Este tipo de arquitectura permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que utilice HTTP. Por esto es que REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

Sin importar si la aplicación externa utiliza Java o cualquier otro lenguaje de programación, es posible integrarla con Bonita OS utilizando la API REST. Esta API provee acceso a todos los objetos de Bonita (como procesos, tareas, usuarios, conectores, etc.), para ejecutar operaciones sobre ellos (crearlos, consultarlo, actualizarlos o eliminarlos). A partir de esta API, es posible utilizar estas operaciones para crear un *workflow* con el motor de procesos de Bonita ya que sigue siendo el responsable de ejecutar la lógica del *workflow*, mientras que los usuarios siguen siendo los que llevan a cabo las tareas y realizan las actividades administrativas del BPMS.

Existen 3 pasos básicos para que una aplicación externa pueda integrarse y operar con Bonita a través de la API REST.

- Autenticación: para acceder a los métodos de las API, es necesario autenticarse con un módulo de acceso (*LoginModule*) como un usuario registrado en Bonita.
- Ejecutar llamadas REST: una vez que el acceso al motor de procesos de Bonita es exitoso, es posible ejecutar métodos de la API para, por ejemplo, obtener una lista de los procesos desplegados, obtener una lista de las tareas pendientes de un usuario en particular, comenzar una instancia de un proceso, ejecutar tareas, etc.
- Cierre de Sesión: cuando ya se han realizado las tareas deseadas, es necesario cerrar la sesión para evitar errores en el motor de procesos.

Cabe aclarar que para que se pueda utilizar la API REST de Bonita en un ambiente de producción, es necesario contar con un servidor Tomcat o JBoss que fácilmente se puede descargar desde el sitio de Bonitasoft, que contiene al motor de procesos de Bonita junto con uno de estos servidores. [21]

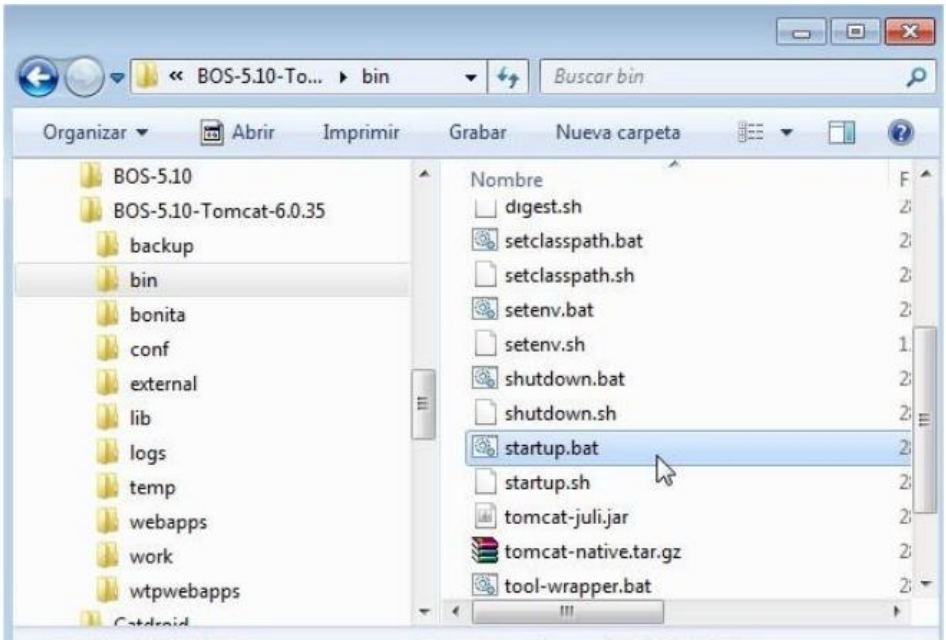


Figura 4.9. Servidor Tomcat con Bonita Open Solution integrado

En la Figura 4.9 se puede ver el árbol de directorios del servidor Tomcat que contiene a Bonita luego de haber descomprimido el archivo descargado. Es necesario tener con el paquete de Java JDK (*Java Development Kit*) [24] instalado e iniciar el servidor desde esta carpeta, utilizando el archivo `startup.bat`, ya que es el archivo que se encarga de asignar las variables de entorno para que sea posible el acceso desde la red.

4.3.5. Obtención de la API REST y configuración de despliegue

Para poder utilizar la API REST de Bonita lo primero que hay que realizar es obtener el archivo `bonita-server-rest.war` y desplegarlo en el servidor Tomcat. Una forma de hacer esto es exportándolo desde Bonita Studio, accediendo al menú "*Proceso/Exportar Avanzado*", cliquear en el botón "*Pasar al último paso>>*", marcar la casilla "*Exportar Runtime*", seleccionar "*REST*" y clickear en el botón "*Exportar*".

El archivo exportado `bonita-server-rest.war` hay que copiarlo a la carpeta "*webapps*" del servidor Tomcat, e iniciar el servidor, que se encargará automáticamente de construir las carpetas correspondientes a la API REST de Bonita.

4.3.6. Autentificación y Autorización en la API REST

Casi todas las aplicaciones o sistemas necesitan de los procesos de autentificación y autorización. Bonita Open Solution utiliza por defecto una configuración JAAS para realizar la autentificación y autorización.

JAAS son las siglas de *Java Authentication and Authorization*. Se trata de una especificación integrada en la máquina virtual Java a partir de la versión 1.4 y cuya finalidad es la de definir un estándar para los procesos de autentificación y autorización.

Ambos procesos, autentificación y autorización están directamente relacionados con la seguridad de las aplicaciones. La **autentificación** es el proceso por el cual un usuario o servicio tiene que autentificarse para poder acceder a ciertos servicios que ofrece nuestro sistema. La **autorización** es el proceso por el cual se controlan las acciones que tiene un usuario o servicio normalmente ya autenticado puede realizar, para ello se le conceden o deniegan permisos.

JAAS es un *framework* de seguridad de Java que define la capa de abstracción entre una aplicación y un mecanismo de autenticación, permitiendo conectar el mecanismo deseado sin cambiar el código de la aplicación. Configurando unas cuantas propiedades de agente, es posible conectar cualquier servicio de autenticación compatible con JAAS y actualizarlo sin interrumpir ni alterar el código del agente.

La Figura 4.10 muestra los elementos básicos de JAAS: un cliente JAAS, un servicio de autenticación compatible con JAAS y un archivo de configuración de JAAS.

El **cliente JAAS** es una aplicación que busca realizar la autenticación con un servicio compatible con JAAS. Se comunica con este servicio mediante módulos de registro (*LoginModule)* y se encarga de proporcionar un controlador de devolución de llamadas que puede utilizar el *LoginModule* para obtener el nombre del usuario, su contraseña y otra información importante.

El **servicio de autenticación** compatible con JAAS consiste en uno o más módulos de registro y en los módulos de autenticación adicionales que realizan la autenticación necesaria. El *LoginModule* puede incluir la lógica de autenticación o puede utilizar un protocolo privado o API para comunicarse con un módulo que proporcione el proceso lógico. Los módulos de registro se ejecutan en la misma máquina virtual Java que el agente.

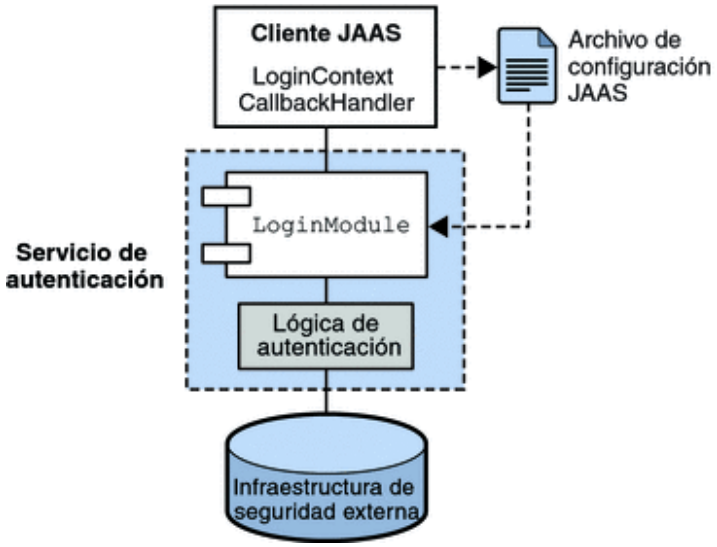


Figura 4.10. Elementos de la arquitectura JAAS [25]

El **archivo de configuración** de JAAS es un archivo de texto que utiliza el cliente JAAS para encontrar los *LoginModules* necesarios para comunicarse con el servicio compatible con JAAS. En Bonita, este archivo de configuración que contiene los diferentes módulos de registro se llama `jaas-estandar.cfg`.

La clase `javax.security.auth.login.LoginContext` de Java provee los métodos básicos utilizados para autentificar a los usuarios, y permite separar a la aplicación del proceso de

auteticación. El *LoginContext* consulta al archivo de configuración `jaas-standard.cfg` para determinar los módulos de acceso (*LoginModules*) que están permitidos en la aplicación, y actúa de acuerdo al proceso de identificación que se encuentra desarrollado dentro de la aplicación. [25]

Por ejemplo, en Bonita, el archivo `jaas-standard.cfg` se compone de los siguientes *LoginModules* principales:

```
BonitaAuth {
  org.ow2.bonita.identity.auth.BonitaIdentityLoginModule required;
};

BonitaStore {
  org.ow2.bonita.identity.auth.LocalStorageLoginModule required;
};

BonitaRESTServer {
  org.ow2.bonita.identity.auth.BonitaRESTServerLoginModule optional logins="restuser"
passwords="restbpm" roles="restuser";
};
```

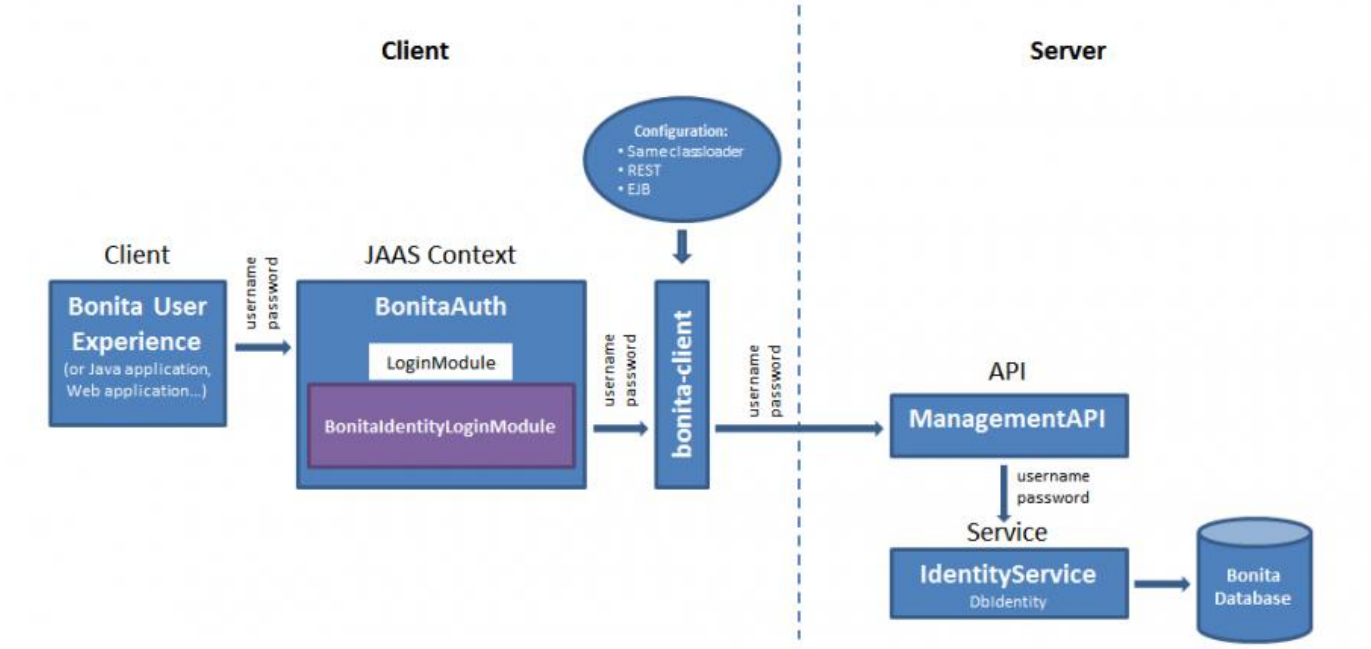


Figura 4.11. Acceso local a Bonita User Experience [23]

En el caso del acceso local a Bonita, es decir, al acceder al *User Experience*, los pasos que se realizan, y que pueden observarse en la Figura 4.11, son los siguientes:

- El cliente realiza el acceso frente al contexto de *BonitaAuth* JAAS.
- *BonitaAuth* delega el modulo de acceso a *BonitaIdentityLoginModule*
- *BonitaIdentityLoginModule* llama al servicio *checkUserCredentials* que es otorgado por *BEE Management API*

- *Management API* se basa en el servicio de identidad (*IdentityService*) que provee la BEE, la cual obtiene los datos de autenticación del usuario de la base de datos de Bonita.

Ahora, cuando se expone a Bonita al acceso REST (HTTP), es necesario adaptar la configuración en la autenticación, ya que para poder utilizar la API REST se debe cambiar el contexto que se mencionó anteriormente. Para chequear el acceso HTTP, Bonita utiliza por defecto el módulo de acceso *BonitaRESTServer*.

En este caso se utiliza el módulo de acceso *BonitaRESTLoginModule* que es el que verifica que el usuario y contraseña suministrados por la petición HTTP sean los mismos que se encuentran dentro del módulo *BonitaRESTServerLoginModule*, que es el encargado de otorgar el acceso a la API REST, la cual gestiona las peticiones HTTP para poder acceder al motor de procesos de Bonita (BEE). En la Figura 4.12 se puede ver el funcionamiento de los distintos componentes en el acceso mediante REST.

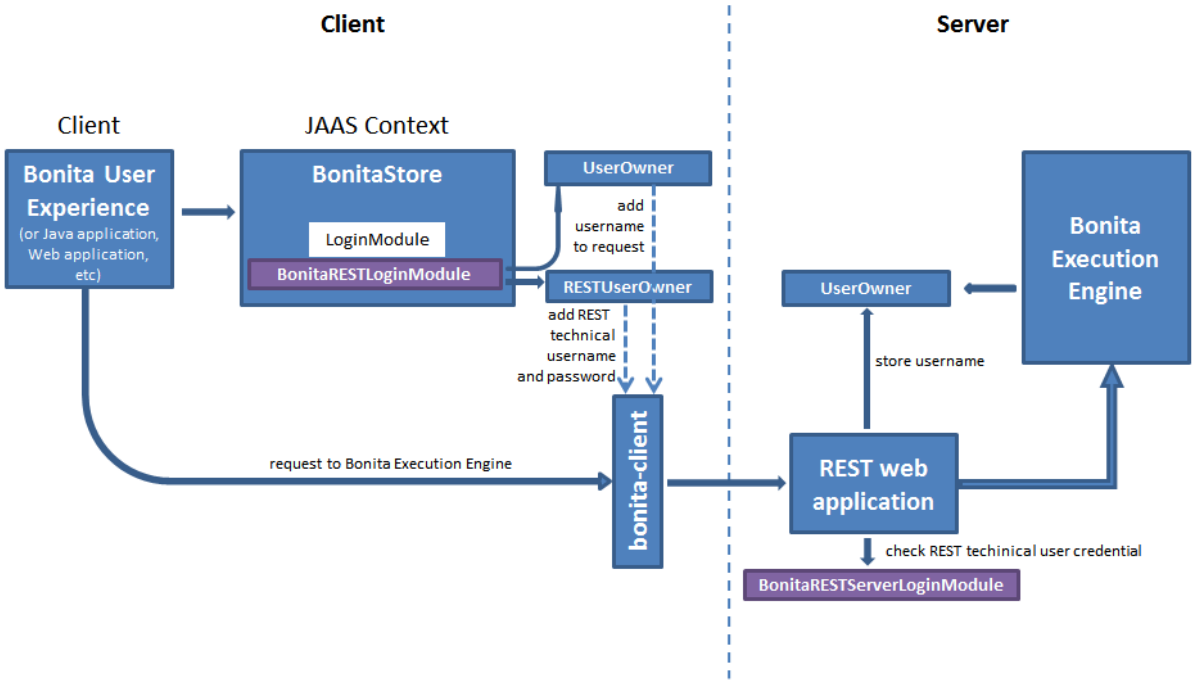


Figura 4.12. Acceso a Bonita a través de REST [23]

Es posible utilizar a Bonita para que se ejecute en ambos contextos, es decir, local y externo mediante REST, sin embargo, ambos contextos no se pueden ejecutar de forma simultánea. Gracias a que el motor de Bonita es transaccional, se debe cambiar de contexto cada vez que se desea acceder mediante REST, ejecutar las acciones necesarias y volver al contexto local (*BonitaAuth*) para que no se produzcan errores en la ejecución de las actividades.

Una vez que se ha configurado correctamente el acceso a la API REST, es posible invocar los métodos que conforman a esta interfaz.

4.3.7. Ejecución de métodos de la API REST

Para poder ejecutar los métodos que componen a la *Bonita Execution Engine* (BEE), primero se debe consultar conocer los métodos disponibles que otorga esta API. Para realizar esto es posible consultar a la documentación de la API REST que se encuentra en el sitio de Bonita en [26]. Aquí es se puede encontrar la documentación de las clases que componen a la API, sus métodos y los parámetros necesarios para ejecutarlos. Las clases son las mismas que en la sección 4.3.3, pero con la diferencia que la API REST es una interfaz que sirve de adaptador HTTP para la BEE. Al ingresar dentro de la documentación de cada clase, es posible visualizar los métodos disponibles que se pueden invocar, junto con una descripción de lo que realiza cada uno, y los parámetros que deben enviarse cuando se genera dicha petición.

La API REST de Bonita, para el acceso mediante peticiones HTTP, utiliza 3 tipos de parámetros que puede utilizar para la ejecución de los métodos que componen a la BEE:

- **Parámetro Path**: los parámetros Path (o Ruta) están identificados por los símbolos "`{`" y "`}`" en la ruta URL. Para utilizarlos hay que reemplazar "`{nombre parámetro}`" por el valor que se desea ejectar. Por ejemplo, para el método `getProcessesByState`:

  ```
  /API/queryDefinitionAPI/getProcessesByState/{processState}
  ```
  se transforma en
  ```
  /API/queryDefinitionAPI/getProcessesByState/READY
  ```

- **Parámetro Query**: los parámetros Query (o de consulta) se pone luego del símbolo "?" en la URL. Se puede observar en el detalle de los métodos cómo utilizarlos. Para definir un valor para un parámetro de consulta, se debe poner luego del símbolo "=". Por ejemplo, para el método `getLightProcessesByIndexAndPageSize, :`

  ```
  /API/queryDefinitionAPI/getLightProcessesByIndexAndPageSize?fromIndex=?&pageSize=?
  ```
  que luego reemplazando los símbolos se convierte en,
  ```
  /API/queryDefinitionAPI/getLightProcessesByIndexAndPageSize?fromIndex=0&pageSize=20
  ```

- **Parámetro Form**: a diferencia de los parámetros Query y Path, un parámetro Form no es enviado por la URL, sino que se debe enviar dentro del cuerpo (*body*) de la petición HTTP. Para utilizarlos, se debe agregar el parámetro al contenido de la petición HTTP y utilizar el encabezado "`Content-Type`" con el valor "`application/x-www-form-urlencoded`".

Para ejecutar los métodos se debe conocer la dirección IP en la cual se encuentra el servidor REST. Si se ha desplegado el servidor REST como se indicó en la sección 4.3.5, la URL en la cual se enviarán las peticiones suele tener la forma:

```
http://{direccionIP}:{puertoUtilizado}/bonita-server-rest/
```

Luego es necesario indicar los métodos que se desean ejecutar. Por ejemplo, para ejecutar el método `getProcessesByState` en `localhost` en el puerto 8080, se debe realizar una petición a la siguiente URL:

`http://localhost:8080/bonita-server-rest/API/queryDefinitionAPI/getProcessesByState/READY`

El resultado de esta petición, devolverá todos aquellos procesos que estén en un estado de `READY`.

Capítulo 5 - **Solución propuesta**

En este capítulo se presenta una solución propuesta para la ejecución y monitoreo de procesos de negocios distribuidos. Se comenzará introduciendo al lector sobre los temas generales en la ejecución de los procesos distribuidos, empezando con el diseño de los subprocesos a partir de los criterios que se deben tomar en la descomposición de los procesos. Luego, se presentará el conector de Bonita desarrollado capaz de iniciar instancias de procesos desplegados en otros motores de procesos de este BPMS como punto de partida para la ejecución distribuida de los procesos.

Para introducir en la idea de la solución de monitoreo, se define de la infraestructura y arquitectura necesaria para llevar a cabo la fase de ejecución y monitoreo, y se presentará un ejemplo sencillo en el cuál se le permitirá ver al lector de manera práctica como se puede utilizar varios motores de procesos situados en diferentes servidores para realizar la ejecución.

Por último, se mostrarán las posibilidades que ofrece Bonita para la recolección de los datos de las instancias que están, o han sido ejecutadas en el motor de procesos, y se presentarán algunas herramientas que posibilitan esta recolección de información a través del lenguaje PHP con el que se desarrollará la aplicación de monitoreo.

5.1. Introducción

La solución que se propone para la ejecución y monitoreo de los procesos de negocio distribuidos puede separarse en tres grandes fases: el diseño del proceso de negocio distribuido, la ejecución de cada uno de los subprocesos en los servidores en los que se encuentran, y la recolección de la información perteneciente a los subprocesos o a las instancias, que servirán para realizar el monitoreo.

El concepto de “subproceso” en la notación BPMN 2.0 tiene un significado diferente al que se utilizará en esta parte del informe. Mientras que en la bibliografía actual se utiliza el término para referirse a una actividad compuesta que es incluida dentro de un proceso y que contiene un conjunto de actividades, compuertas, eventos y flujos de secuencia [31], en este capítulo se utilizará dicho término para referirse a las partes del proceso de negocio original que ha sido dividido a raíz de la descomposición del proceso.

El diseño del proceso de negocio distribuido, al igual que en el diseño de los procesos de negocio que se conocen comúnmente, es la fase previa para la ejecución de los procesos de negocio. Sin embargo para realizar la ejecución de procesos de negocio de forma distribuida es necesario descomponerlos para crear los subprocesos que serán situados en los distintos motores de procesos que se encuentran en los servidores distribuidos.

Como se ha visto en el Capítulo 3, los factores que hacen que se lleve a cabo la descomposición de los procesos de negocio están dados principalmente por las características que tienen los sistemas en los que serán ejecutadas las actividades que componen a los procesos, esto es, la alta capacidad de procesamiento, el pago por uso y la facilidad en la escalabilidad de sistemas

basados en *cloud*, y por la privacidad de los datos en los sistemas embebidos. De esta manera, el diseñador de procesos debe ser capaz de determinar aquellas actividades que serán adecuadas para cada servidor y descomponer el proceso de acuerdo a estas actividades.

Como producto del proceso de negocio descompuesto se tendrán dos o más subprocesos que se deberán ejecutar en distintos motores de procesos. La ejecución del proceso de negocio distribuido comenzará con la generación de una instancia de uno de los subprocesos, y luego éste será el encargado de iniciar la ejecución del subproceso siguiente, o de los diferentes subprocesos en caso de que existiera más de uno. Para llevar a cabo la continuación de la secuencia en la ejecución de los subprocesos, se hará uso de un conector de Bonita que utiliza la API REST del motor de procesos remoto para iniciar una instancia del subproceso que se encuentra en dicho motor.

Una vez que se encuentran desplegados los subprocesos en cada uno de los servidores intervinientes, se hará uso de una aplicación web que utilizará la API REST de Bonita para recolectar toda la información referida a los subprocesos y visualizar al proceso de negocio distribuido como si se tratase de un proceso único. Además, en dicha aplicación será posible observar información de cada una de las actividades del proceso, lo que servirá para monitorear las instancias y obtener valores de desempeño de cada una de las actividades.

Si bien se ha hecho hincapié en la combinación del uso de un sistema cloud y uno embebido, en la siguiente solución se pretende generalizar este modelo para lograr una visión en la cual es posible adaptar la ejecución de procesos descompuestos a cualquier esquema de servidores distribuidos.

5.2. Criterios para la descomposición de procesos

Las decisiones en la descomposición de los procesos de negocio que son llevadas por los diseñadores de los procesos, estarían dadas principalmente por las características de la infraestructura en la que se van a ejecutar las actividades de estos procesos.

Por ejemplo, si se quiere descomponer un proceso en el cual existe una actividad que consulta una base de datos que contiene información confidencial, esa actividad debería ejecutarse en un servidor en el cual se tenga el control de esos datos confidenciales, y si bien es posible realizar las consultas de forma remota a una base de datos, es posible que haya una restricción en el sistema que no permita este tipo de consultas debido a normas de seguridad, por lo tanto, sería obligatorio que la actividad se ejecute en un motor de procesos que se encuentre en un servidor propio.

Por otro lado, si existen actividades que no tienen este tipo de restricciones de seguridad, ejecutar todas las actividades en un servidor propio requeriría que se disponga de una gran infraestructura de hardware para que se puedan procesar todas las solicitudes en un tiempo relativamente aceptable. Para evitar el elevado costo en hardware es posible contratar los servicios de cloud, que se ha mencionado en la *Sección 2.2*, de una manera de pago por uso, y que permite escalar la infraestructura de hardware según las necesidades de capacidad de procesamiento que vayan surgiendo.

Al realizar la descomposición, el esquema de ejecución de los procesos descompuestos consiste en encadenar el flujo de las instancias correspondientes a los subprocesos particionados.

Así, al finalizar una instancia en un servidor, esta inicia automáticamente una nueva instancia de la partición de proceso siguiente en el servidor que corresponda de acuerdo a la arquitectura de distribución. Para esto, cada servidor del esquema distribuido debe ser capaz de comunicarse con el servidor siguiente para poder iniciar instancias y continuar con la ejecución del proceso original. [10] [13]

5.3. Creación de un conector iniciador de instancias

Para conseguir que los subprocesos distribuidos se ejecuten automáticamente según el flujo de control de las actividades que residen en diferentes servidores, se hace uso de un conector de Bonita que se encarga de iniciar una instancia en el motor de procesos remoto. Esta solución, si bien no es totalmente automática, ya en la etapa de diseño se deben asignar parámetros correspondientes a la ubicación del servidor en el cual se encuentra el motor de procesos y el nombre del proceso en el servidor remoto, se puede observar la automatización en tiempo de ejecución, esto quiere decir que una vez instalados los procesos en los motores de procesos, las llamadas son totalmente invisibles al usuario.

El conector está desarrollado para realizar dos tareas, la primera es la de instanciar procesos en otros motores de procesos de Bonita, y la segunda, la de guardar en una base de datos el identificador de las instancias que han instanciado procesos en otro motor, como así también el identificador de la nueva instancia en el servidor remoto. Con estos datos, la aplicación de monitoreo será capaz de interactuar con los diferentes servidores para recolectar la información perteneciente a las instancias ejecutadas en otros motores de procesos.

Una imagen del conector se puede ver en la Figura 5.1, en la cual se puede ver la descripción del conector. En la descripción se observan las páginas necesarias que deben completarse según la configuración del motor de procesos remoto, las variables que se desean enviar al proceso remoto y los datos de acceso a la base de datos local en la cual se almacenará el identificador de la instancia que ejecuta el conector como así también el identificador de la instancia remota.

El conector también genera datos de salida que servirán para controlar que se haya ejecutado correctamente o verificar cual es el problema que causó que no se haya podido ejecutar, y mostrar un mensaje de acuerdo a cual sea el caso. El uso de estos datos de salida servirá para controlar el flujo de trabajo del proceso, haciendo que la instancia no finalice hasta que se complete la instanciación del proceso remoto (en la Sección 5.5 se presentará un ejemplo simple en la ejecución de procesos en diferentes motores en donde se verá cómo se puede controlar el flujo de trabajo).

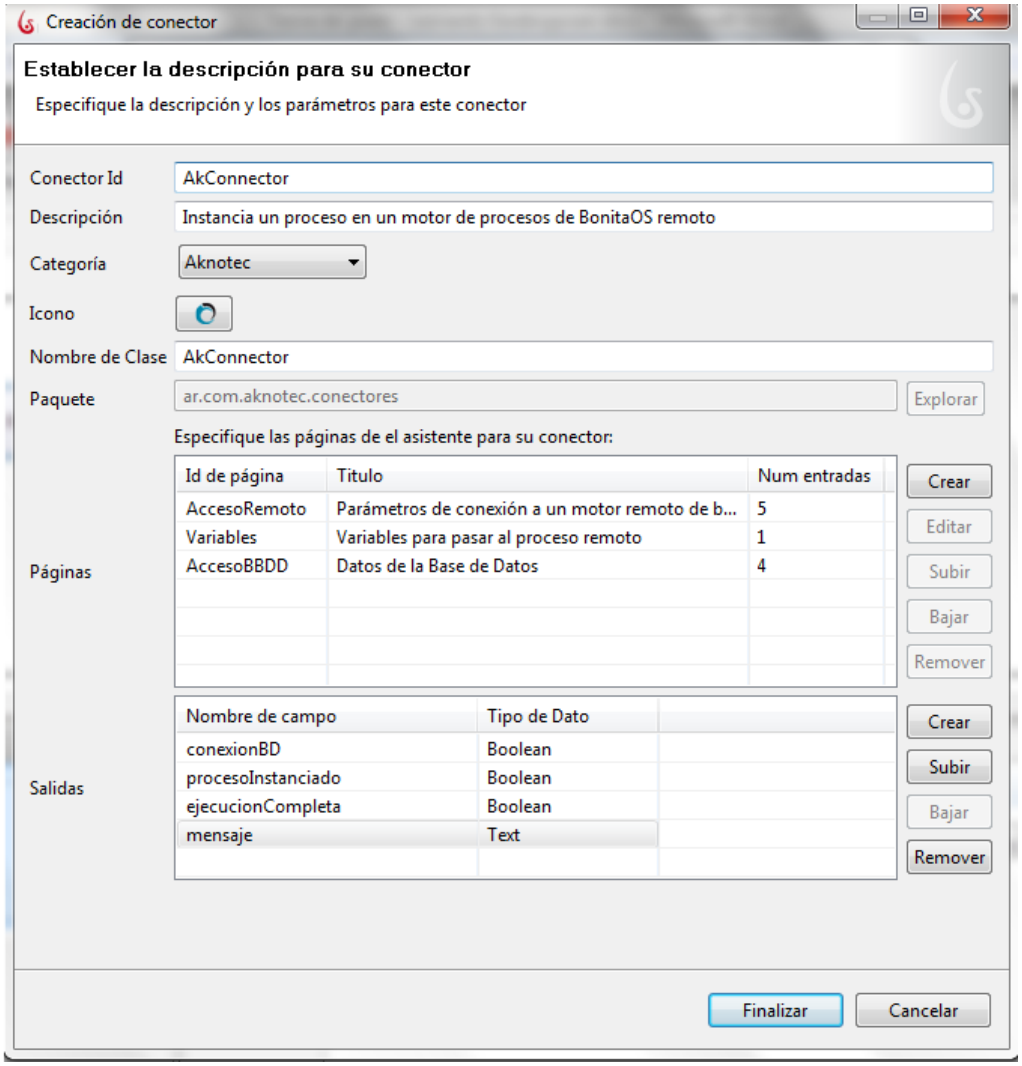


Figura 5.1. Descripción del conector para ejecutar instancias remotas

El conector funciona de la siguiente manera:

1. Primero verifica si hay acceso a la base de datos local donde se guardará la información de la instancia que se está ejecutando, junto con los datos de acceso al servidor remoto y la fecha y hora actual, esperando que se instancie el proceso en el servidor remoto para actualizar la información correspondiente al identificador de la instancia remota. En caso de que no se consiga el acceso a la base de datos, se supone que existe un problema con la misma, por lo tanto no será posible instanciar el proceso remoto ya que se perdería el rastro de los procesos desplegados en ambos sitios.
2. Una vez que se obtiene acceso a la base de datos local y se inserta la fila correspondiente a la instancia en curso, se intenta conectar a la API REST del servidor remoto. Para realizar esto es necesario cambiar el motor de procesos local al contexto "REST", donde se autenticará al servidor mediante el usuario y contraseña proporcionado al asistente del conector en el momento del diseño del proceso. En el caso de que el servidor remoto no se encuentre disponible o los datos proporcionados de usuario y contraseña no sean los adecuados, no será posible instanciar el proceso

remoto, por lo tanto, se deberá borrar la fila insertada en el paso anterior, en donde se verifica y se inserta en la base de datos.

3. Si los datos de acceso al servidor remoto son correctos y se ha instanciado correctamente el proceso remoto, se llevará a cabo la actualización de la fila insertada en la primera parte con el identificador del proceso remoto. De esta manera, se tendrán todos los datos necesarios para utilizarlos posteriormente en la etapa de monitoreo. Al finalizar, se debe volver al contexto Standard.

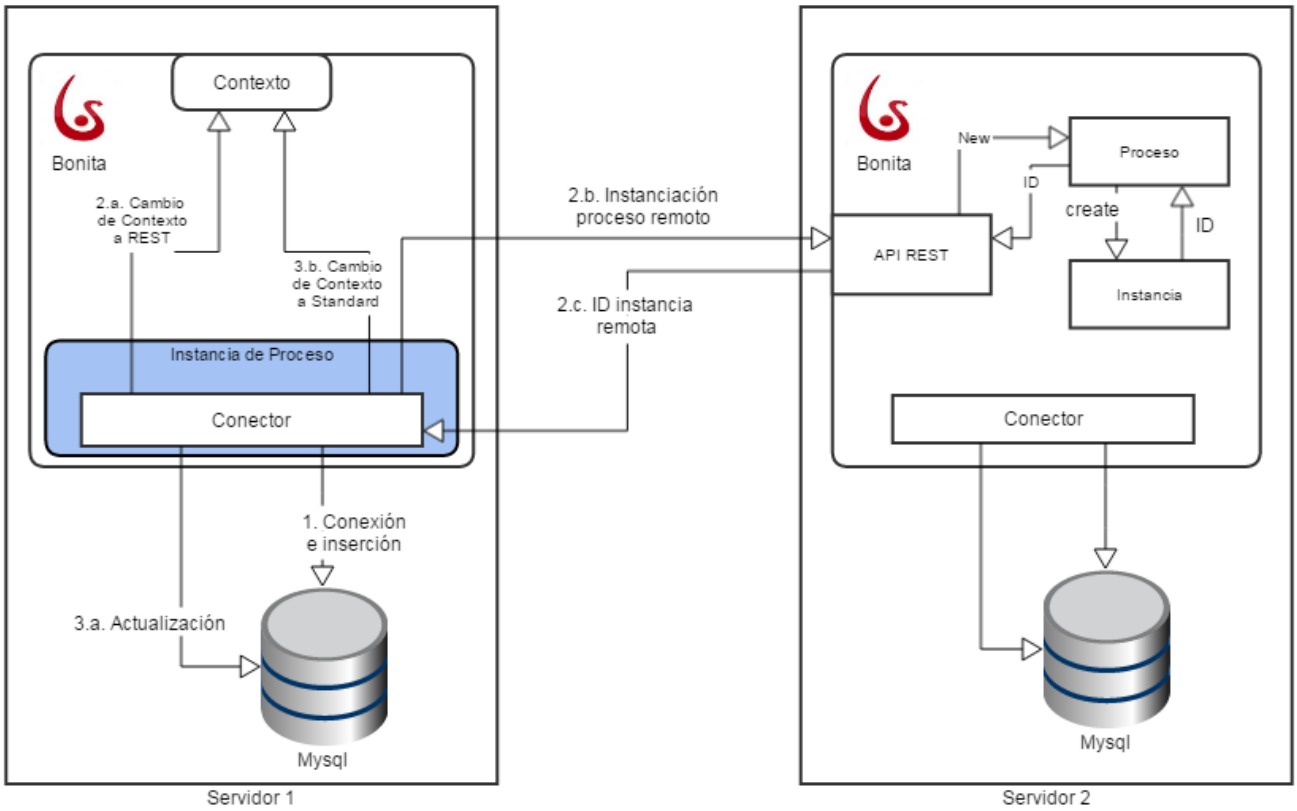


Figura 5.2. Diagrama de flujo del funcionamiento del conector

A continuación se presenta el código fuente del conector de Bonita Open Solution escrito en Java, en el cual se puede ver el comportamiento descrito anteriormente.

```
package ar.com.aknotec.conectores;

import java.sql.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import javax.security.auth.login.LoginContext;
import org.ow2.bonita.connector.core.ConnectorError;
import org.ow2.bonita.connector.core.ProcessConnector;
import org.ow2.bonita.facade.RuntimeAPI;
import org.ow2.bonita.facade.def.majorElement.ProcessDefinition;
import org.ow2.bonita.facade.uuid.ProcessDefinitionUUID;
import org.ow2.bonita.facade.uuid.ProcessInstanceUUID;
import org.ow2.bonita.util.AccessorUtil;
import org.ow2.bonita.util.BonitaConstants;
import org.ow2.bonita.util.SimpleCallbackHandler;
import org.slf4j.Logger;

public class AkConnector extends ProcessConnector {
   private Boolean ejecucionCompleta = false;
```

```
   private Boolean procesoInstanciado = false;
   private Boolean conexionBD = false;
   private String mensaje = "";
   private java.lang.String servidor;
   private java.lang.String proceso;
   private java.lang.String passwordBD;
   private java.lang.String usuario;
   private java.lang.String usuarioBD;
   private java.lang.String nombreBD;
   private java.lang.String password;
   private java.lang.String url;
   private java.util.List<java.util.List<Object>> variables;
   private java.lang.String version;
   private Connection conexion;
   private Statement statement;

   private ProcessInstanceUUID instanciaLocal;
   private ProcessInstanceUUID instanciaRemota;

   @Override
   protected void executeConnector() throws Exception {
      LoginContext loginContext = null;
      final String LOGIN = this.usuario;
      final String PSSWD = this.password;
      Logger logger = org.slf4j.LoggerFactory.getLogger(this.getClass());

      try {
         logger.info("Comienza el conector \n");
         try {
            // Verificación del acceso e inserción en la Base de Datos Local
            Class.forName("com.mysql.jdbc.Driver");
            java.util.Date dt = new java.util.Date();
            java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            String currentTime = sdf.format(dt);
            this.conexion = DriverManager.getConnection("jdbc:mysql://" + this.servidor + "/" +
this.nombreBD, this.usuarioBD, this.passwordBD);
            this.statement = this.conexion.createStatement();
            this.instanciaLocal = this.getProcessInstanceUUID();//Nombre de la instancia local

            String query = "INSERT INTO instancias (instancia_local, instancia_remoto, url_remoto,
                              usuario_remoto, password_remoto, fecha) " +
                        "VALUES ('" + this.instanciaLocal + "', '" + "obteniendoInstancia" + "', '" +
                                 this.url + "', '" + this.usuario + "', '" + this.password + "', '" +
                                 currentTime + "')";

            int result = this.statement.executeUpdate(query);
            if (result == 0){
               this.mensaje = "No se pudo insertar los datos de la instancia en la base de datos.
                              La sentencia SQL es: " + query;
            }
            else {
               this.conexionBD = true;
            }
         }
         catch (Exception e) {
            logger.error("Problemas para acceder a la base de datos: " + e.getMessage());
            this.mensaje = "Problemas para acceder a la base de datos: " + e.getMessage();
         }

         if (this.conexionBD){
            try {
               //Chequea el logueo a la API REST del servidor remoto e instancia el proceso remoto
               System.setProperty(BonitaConstants.API_TYPE_PROPERTY, "REST");
               System.setProperty(BonitaConstants.REST_SERVER_ADDRESS_PROPERTY, this.url);
               System.setProperty(BonitaConstants.REST_SERVER_EXCEPTION, "");
               //Reseteo el contexto para usar la API REST
               AccessorUtil.resetContext();

               //Verificacion del usuario en el servidor remoto
               loginContext = new LoginContext("BonitaAuth", new SimpleCallbackHandler(LOGIN,
PSSWD));
               loginContext.login();
               loginContext.logout();

               //Acceso a la API REST del servidor remoto
               loginContext = new LoginContext("BonitaStore", new SimpleCallbackHandler(LOGIN,
```

```
PSSWD));
                loginContext.login();
                this.procesoInstanciado = true;

                logger.info("Logueo Correcto");
                RuntimeAPI runtimeAPI = AccessorUtil.getRuntimeAPI();
                ProcessDefinition process =
                         AccessorUtil.getQueryDefinitionAPI().getProcess(this.proceso,this.version);
                ProcessDefinitionUUID processUUID = process.getUUID();
                HashMap<String, Object> vars = new HashMap<String, Object>();
                for (List<Object> o : this.variables) {
                   vars.put(o.get(0).toString(), o.get(1));
                }
                this.instanciaRemota = runtimeAPI.instantiateProcess(processUUID,vars);
                logger.info("Se ejecutó el proceso. \nEl UUID de la instancia es: " +
instanciaRemota);

             } catch (Exception e) {
                logger.error(e.getMessage());
                this.mensaje = "Problemas en el acceso al servidor remoto: " + e.getMessage();
             }
             finally {
                //Vuelve al contexto anterior para que se sigan ejecutando los procesos normalmente
                if (loginContext != null) loginContext.logout();
                System.setProperty(BonitaConstants.API_TYPE_PROPERTY, "Standard");
                System.setProperty(BonitaConstants.REST_SERVER_ADDRESS_PROPERTY, "");
                AccessorUtil.resetContext();
             }

             if (this.procesoInstanciado){
                try {
                   //Se actualiza la fila con el identificador del proceso remoto
                   String query = "UPDATE instancias " +
                               "SET instancia_remoto = '" + this.instanciaRemota + "'" +
                               "WHERE instancia_local = '" + this.instanciaLocal + "'";
                   int result = this.statement.executeUpdate(query);
                   if (result == 0){
                      this.conexionBD = false;
                      this.mensaje = "No se pudo actualizar los datos de la instancia en la base
                                  de datos. La sentencia SQL es: " + query;
                   }
                   else {
                      this.ejecucionCompleta = true;
                   }
                   this.mensaje = "Se instanció el proceso remoto correctamente.";
                } catch (Exception e) {
                   logger.error(e.getMessage());
                   this.mensaje = "Problemas para actualizar la fila: " + e.getMessage();
                }
             }
             else {
                try {
                   //Si hay problemas ejecutando el proceso remoto, se borra la fila de la BBDD
                   String query = "DELETE FROM instancias " +
                                  "WHERE instancia_local = '" + this.instanciaLocal + "'";
                   int result = this.statement.executeUpdate(query);
                   if (result == 0){
                      this.mensaje = "No se pudo borrar la fila de la instancia en la base de datos.
                                   La sentencia SQL es: " + query;
                   }
                   else {
                      this.conexionBD = false;
                   }
                   this.ejecucionCompleta = false;
                } catch (Exception e) {
                   logger.error(e.getMessage());
                   this.mensaje = "Problemas para borrar la fila: " + e.getMessage();
                }
             }
          }
       }
        catch (Exception e) {
          logger.error(e.getMessage());
          this.mensaje = "Problemas al inicio del conector: " + e.getMessage();
       }
    }
```

```
    /**
     * Validador de los datos de entrada. Función por defecto de un conector de Bonita.
     */
    @Override
    protected List<ConnectorError> validateValues() {
        List<ConnectorError> errorsList = new ArrayList<ConnectorError>();
        …
        …
        return errorsList;
    }

    /**
     * Setter y Getters que deben estar en el conector para el intercambio de información entre Bonita
     * y el conector
     */
    public void setServidor(java.lang.String servidor) {…}
    public void setProceso(java.lang.String proceso) {…}
    public void setPasswordBD(java.lang.String passwordBD) {…}
    public void setUsuario(java.lang.String usuario) {…}
    public void setUsuarioBD(java.lang.String usuarioBD) {…}
    public void setNombreBD(java.lang.String nombreBD) {…}
    public void setPassword(java.lang.String password) {…}
    public void setUrl(java.lang.String url) {…}
    public void setVariables(java.util.List<java.util.List<Object>> variables) {…}
    public void setVersion(java.lang.String version) {…}
    public java.lang.Boolean getConexionBD() {…}
    public java.lang.Boolean getProcesoInstanciado() {…}
    public java.lang.Boolean getEjecucionCompleta() {…}
    public java.lang.String getMensaje() {…}
}
```

5.4. Arquitectura del sistema de ejecución y monitoreo

La arquitectura del sistema para la ejecución y monitoreo de procesos de negocio distribuidos estará basada principalmente en tres componentes encargados de la ejecución y monitoreo de los procesos de negocio distribuidos, estos son:

- El motor de procesos de Bonita Open Solution, que se ejecuta sobre un servidor Tomcat
- La aplicación de monitoreo distribuido, que al estar desarrollada con el lenguaje de programación PHP se ejecuta en un servidor Apache
- Un servidor de base de datos (MySQL) para almacenar la información de las instancias de los procesos de negocio

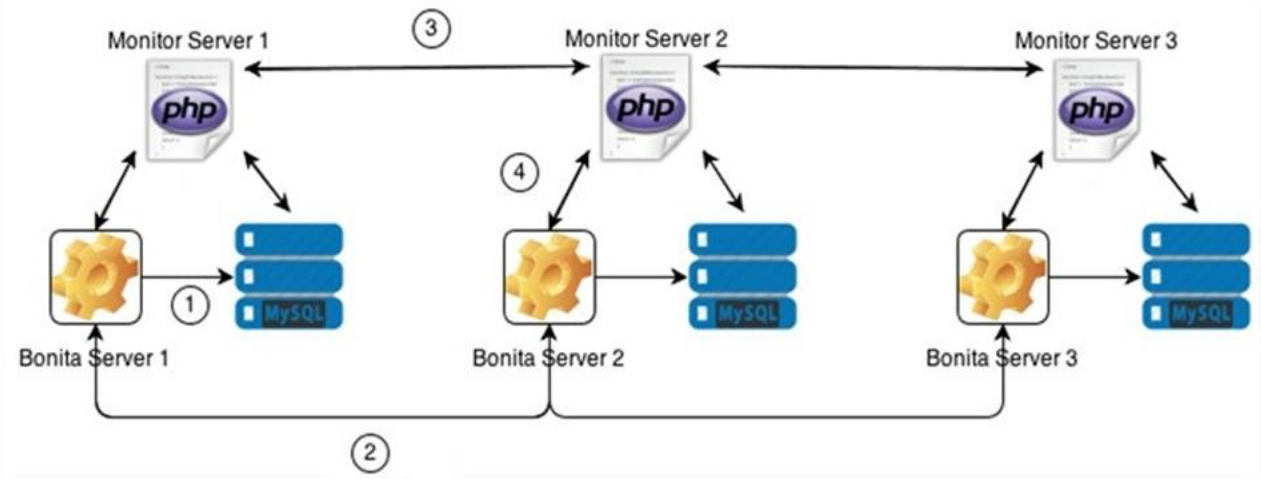


Figura 5.3. Arquitectura del Sistema de Ejecución y Monitoreo [10]

La arquitectura del sistema de monitoreo puede verse en la Figura 5.3, en la cual podemos observar el flujo de la información entre los distintos componentes del sistema, y si bien son visibles solo 3 nodos, la arquitectura es extensible a la cantidad de nodos que sean necesarios. Sin embargo, en caso de existir un encadenamiento de procesos entre los nodos 1, 2 y 3, el nodo 3 sería totalmente invisible al nodo 1, siendo el 2 su intermediario.

Observando esta Figura, en los puntos 1 y 2 el conector hace uso del driver de MySQL para almacenar la información de las instancias, y de la API REST (que en este caso se ejecuta utilizando Java) de Bonita para crear la instancia del proceso en el servidor remoto; en el punto 3 la aplicación utiliza servicios web localizados en otra aplicación remota para obtener la información correspondiente a las actividades de las instancias. En 4, dicha aplicación remota hace uso de la API REST (que aquí se ejecuta utilizando PHP debido a que la aplicación de monitoreo está desarrollada en este lenguaje de programación) de Bonita para poder responder a la solicitud.

5.4.1. Motor de procesos de Bonita Open Solution

El motor de procesos de Bonita OS, situados en cada uno de los nodos de la arquitectura, es el encargado de la ejecución normal de los procesos de negocio. En cada uno de los nodos es necesario desplegar un servidor Tomcat en donde se ejecutará el motor de procesos.

En cada motor de procesos se ejecutarán las actividades que podrían tener el conector que inicia las instancias de los procesos localizados en otros motores de Bonita.

5.4.2. Aplicación de monitoreo distribuido

La aplicación de monitoreo será la encargada de recolectar la información de los subprocesos situados en los diferentes nodos para unificar el proceso original particionado y mostrar el proceso como si se tratase de uno solo.

Se deberá comunicar con las aplicaciones de monitoreo que residen en los otros nodos a través del uso de servicios web (WebServices), donde las aplicaciones de estos nodos serán las encargadas de recolectar la información de los subprocesos que son ejecutados en el motor de procesos que residen en ese sitio y que conocen si el subproceso contiene alguna referencia a otro nodo. En caso de existir estas referencias a otros nodos, la aplicación será la responsable de comunicarse con la aplicación de monitoreo situada en el nodo donde se encuentra el subproceso y obtener la información del subproceso del tercer nodo que hace referencia, para luego devolver toda la información al primer nodo.

Obsérvese que este tipo de arquitectura delega la recolección de información a las aplicaciones de monitoreo que se encuentran en cada uno de los nodos. Debido a esto, la cantidad de nodos involucrados es extensible a cualquier número de éstos, sólo es necesario identificar a los nodos que se encuentran involucrados en el encadenamiento de la ejecución de los subprocesos.

5.4.3. Servidor de base de datos

El servidor de base de datos (en este caso se utilizará MySQL) es el encargado de almacenar la información de las instancias que realizan la ejecución de los subprocesos en los demás nodos.

Para poder recolectar la información de las diferentes instancias, los datos básicos que se necesitan para obtener la visualización y monitoreo de las instancias de los procesos de negocio son tres: el identificador de la instancia local que realizó la ejecución del conector encargado de instanciar un proceso en un motor de procesos remoto, la localización del servidor remoto en la que se ha realizado la ejecución del subproceso, y el identificador de la instancia en el motor remoto que se ha creado producto del conector ejecutado en el motor de procesos local.

5.5. Diseño y Ejecución de los procesos de negocio distribuidos

Luego de realizar la descomposición del proceso se lleva a cabo el diseño de los subprocesos de forma tal que cada subproceso sea un proceso separado.

Para comprender mejor el diseño y ejecución de un proceso de negocio distribuido se presentará el siguiente ejemplo. Suponga un proceso situado en un solo motor de procesos, en el cual una persona envía un mensaje y otra persona lo recibe. El proceso tendrá la siguiente forma:

El diseño extremadamente simple se puede ver en la Figura 5.4. En este ejemplo, si el proceso se ejecuta en un motor de procesos, solo será necesario tener dos tareas en *lanes* (o sendas) diferentes, ya que por sentido común, el que envía el mensaje no debería ser el que lo reciba. Esto se puede configurar fácilmente en Bonita OS asignando los actores encargados de completar cada una de las tareas.

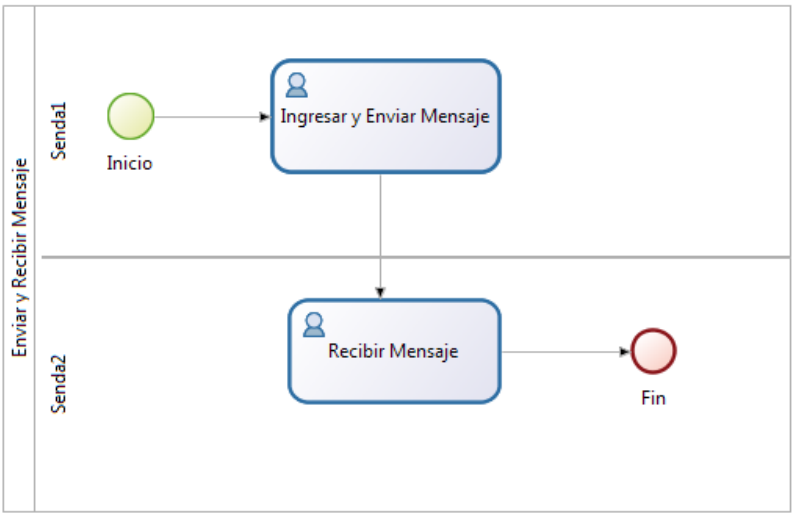


Figura 5.4. Proceso de negocio "Enviar y Recibir Mensaje" en un solo motor de procesos

Si bien en procesos más complejos se debería estudiar la descomposición con mayor análisis, en este proceso de ejemplo es evidente que se debe descomponer en dos subprocesos. El primer subproceso será la parte que envía el mensaje, es decir, la senda superior (Senda1) con la tarea

"Ingresar y Enviar Mensaje", y el segundo el que reciba el mensaje enviado por el primero, que vendría a ser la Senda 2 con la tarea "Recibir Mensaje".

Al descomponer el proceso, las tareas continúan realizando el comportamiento que realizaban en el proceso original, e inclusive utilizan las mismas variables que compartían. Sin embargo, al estar en dos motores de procesos separados, es necesario que la tarea "Recibir Mensaje" sepa de alguna manera que la tarea "Ingresar y Enviar Mensaje" se ha ejecutado. En este momento es cuando el diseñador de los subprocesos debe hacer uso del conector descrito en la Sección 5.3, y en lugar de seguir el flujo de trabajo hacia la tarea "Recibir Mensaje" (que se encuentra en otro subproceso), dirige el flujo hacia una "tarea de servicio" que contiene el conector encargado de instanciar al subproceso "Recibir Mensaje" en otro motor de procesos localizado en otro servidor.

El diseñador del proceso debe configurar el conector para que se conecte al motor de procesos remoto, indicando la URL en la que se encuentra, el nombre del subproceso (que en el motor de procesos remoto sería un proceso más), la versión, el usuario y contraseña para acceder a su API REST.

Como segundo paso debe declarar los datos que le serán pasados al subproceso, en este caso el mensaje que quiere enviarle, ya que en el proceso original utiliza este dato para mostrarle al receptor el mensaje que le enviaron.

Luego, debido a que Bonita no mantiene la relación de las instancias entre motores de procesos, se debe indicar la configuración de la base de datos local para guardar la información de las instancias que intervienen en la ejecución distribuida para que luego, al utilizar la aplicación web de monitoreo, se pueda realizar el seguimiento de las instancias.

Por último, se deben crear las variables donde se guardaran los resultados de la ejecución del conector, y asignar cada dato de salida del conector a estas variables. El uso de estas variables le permite al diseñador manejar el flujo de control del proceso en caso de que ocurra un error en la ejecución del conector.

En la Figura 5.5 se puede ver como quedaría el subproceso "Enviar Mensaje", el cual contiene la tarea "Ingresar y Enviar Mensaje" del proceso original, la tarea de servicio "Enviar Mensaje a través del conector" que contiene el conector, y también otra tarea que muestra el resultado de la ejecución del conector llamada "Resultado Ejecución Remota" que sirve para informar el resultado de la ejecución del conector. En el caso de éxito en la ejecución del conector, el proceso terminaría ya que el mensaje ha sido enviado correctamente, por otro lado, si ha fallado la ejecución (por ejemplo porque no tiene acceso a la base de datos, o porque no ha podido acceder al motor de procesos remoto) se muestra un mensaje de error y no permite la finalización de la instancia hasta que se haya corregido el error que no permite la ejecución. Esto sirve para evitar que queden instancias locales sin una relación con una instancia en un motor de procesos remoto.

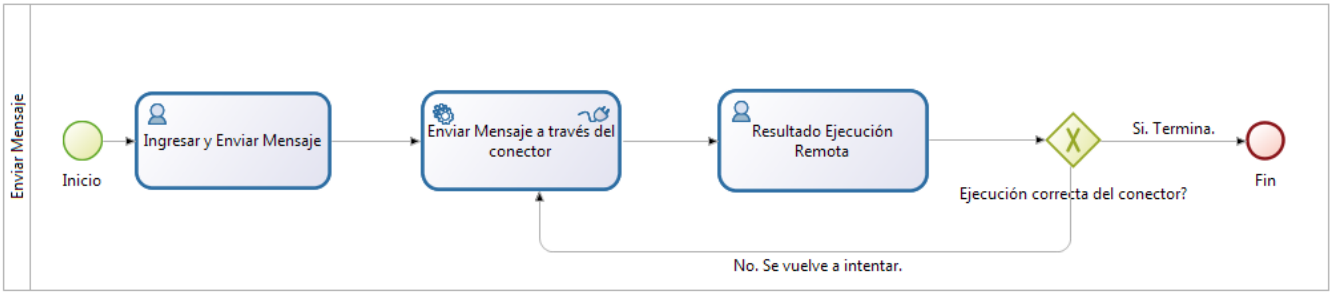


Figura 5.5. Subproceso "Enviar Mensaje" producto de la descomposición

Siguiendo con la descomposición del proceso, solo falta crear un subproceso que contenga la tarea "Recibir Mensaje", el cual se localizará en un motor de procesos diferente al del subproceso "Enviar Mensaje". El subproceso se puede ver en la Figura 5.6.

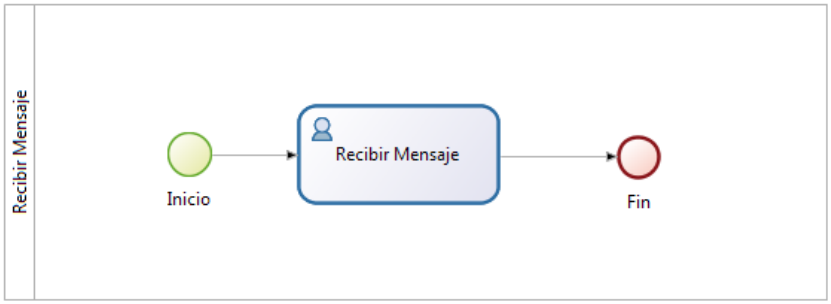


Figura 5.6. Subproceso "Recibir Mensaje" producto de la descomposición

Este subproceso necesita tener la tarea del proceso original y una variable (o la misma cantidad de variables) con el mismo nombre y tipo de la que fue configurada en el segundo paso dentro del conector, para que una vez instanciado, este valor se cargue automáticamente dentro de la variable del subproceso.

La ejecución de este proceso distribuido, desde que se instancia el subproceso "Enviar Mensaje" hasta que se recibe el mensaje en el subproceso "Recibir Mensaje" se puede ver en las siguientes imágenes.

Figura 5.7.1. Tarea "Ingresar y Enviar Mensaje" del subproceso "Enviar Mensaje"

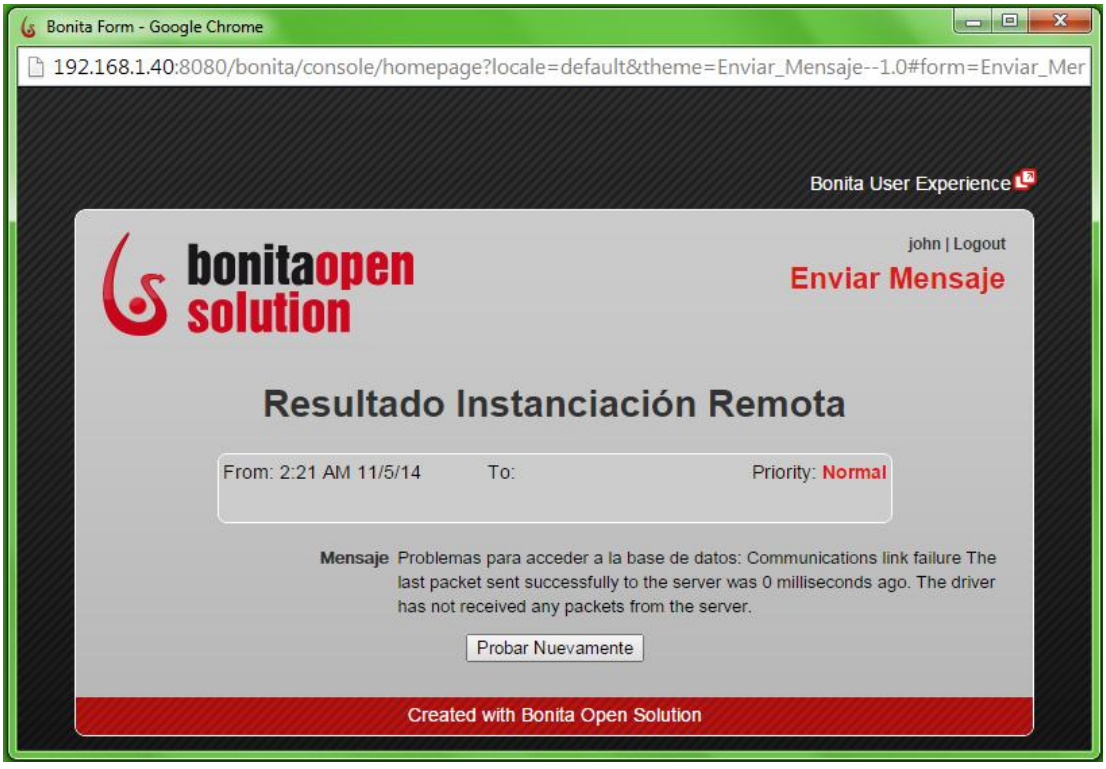


Figura 5.7.2. Tarea "Resultado Ejecución Remota" del subproceso "Enviar Mensaje". Problema con la Base de Datos

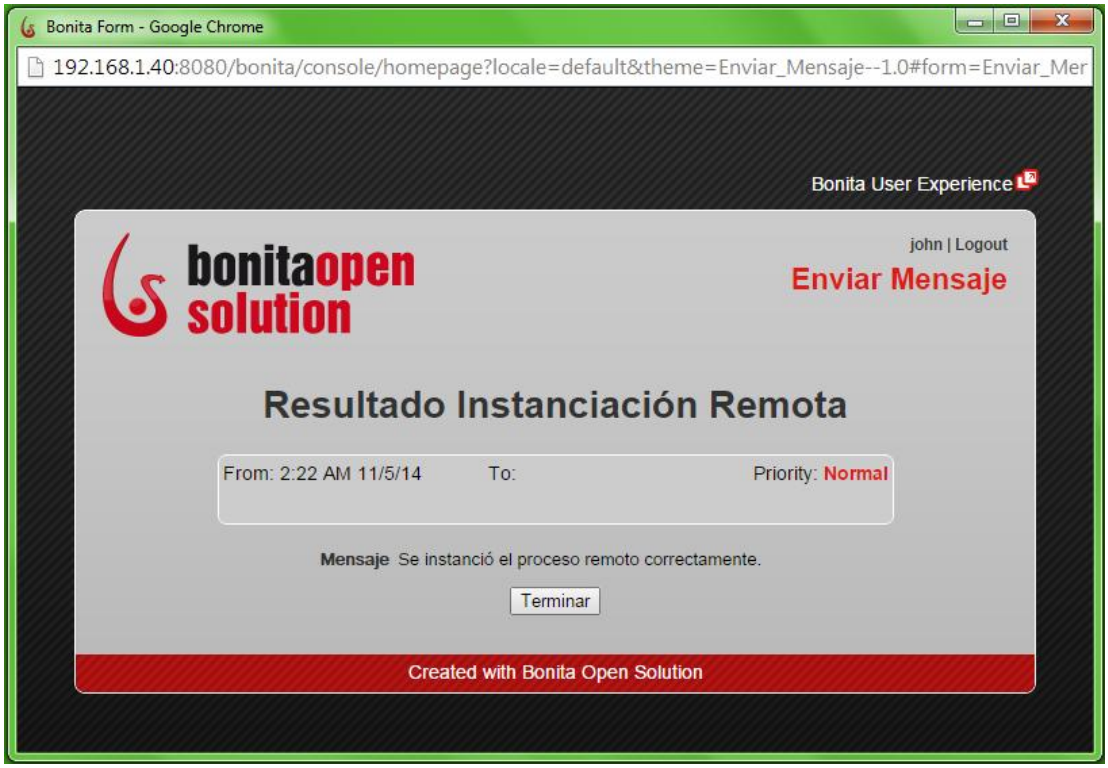


Figura 5.7.3. Tarea "Resultado Ejecución Remota" del subproceso "Enviar Mensaje". Se arregló el problema con la Base de Datos .

Figura 5.7.4. Tarea "Recibir Mensaje" del subproceso "Recibir Mensaje".

Nótese que las tres primeras imágenes la dirección IP del motor de procesos de Bonita es 192.168.1.40, que corresponden al Servidor 1, mientras que en la última es 192.168.1.41, donde se ejecuta el Servidor 2, y que en la Figura 5.7.2 hubo un problema con la base de datos ya que el servicio no se encontraba en ejecución, por lo que no se podía guardar la información de las

instancias. Una vez que se ejecutó el servicio de MySQL en el Servidor 1, el conector permitió que se instanciara el subproceso "Recibir Mensaje" en el Servidor 2.

5.6. Recopilación de los datos de las instancias

Una vez que el proceso de negocio es ejecutado o está siendo ejecutado en los motores de procesos del esquema, dada una instancia iniciada en un servidor de la arquitectura, debemos ser capaces de obtener, no solo los datos propios de la misma, sino también de cualquier instancia que esta haya desencadenado en otro servidor. Para poder realizar esto, será necesario asociar las distintas instancias de proceso iniciadas de manera encadenada, a los fines de poder recuperar información sobre las mismas accediendo a los distintos servidores intervinientes [10] [13].

De esta manera, cuando se obtiene toda la información del estado de los procesos y sus actividades en todo el camino de ejecución de los subprocesos de negocio en los diferentes servidores, es posible unificar los datos para visualizarlos como si se tratase de un solo proceso, y darle al analista de procesos una visión integradora del proceso original, con el fin de que pueda determinar las posibles fallas o mejoras que deban realizarse.

Utilizando la API REST de Bonita, se puede obtener la definición de un proceso de negocio que se encuentra instalado dentro del motor de procesos, como así también la información de las instancias desplegadas dentro del motor.

Al realizar una petición REST de consulta sobre el motor de procesos donde se encuentra instalado el proceso "Enviar Mensaje", podemos obtener una respuesta en formato XML que contiene toda la información del proceso. Por ejemplo, si realizamos la siguiente petición a la API REST del motor de procesos donde se encuentra el proceso "Enviar Mensaje",

`http://192.168.1.40:8080/bonita-server-rest/API/queryDefinitionAPI/getLightProcesses`

obtendremos la siguiente descripción:

```
<set>
      <LightProcessDefinition>
         <description></description>
         <name>Enviar_Mensaje</name>
         <label>Enviar  Mensaje</label>
         <uuid>
          <value>Enviar_Mensaje--1.0</value>
         </uuid>
         <version>1.0</version>
         <state>ENABLED</state>
         <type>PROCESS</type>
         <deployedDate>1415164657578</deployedDate>
         <undeployedDate>0</undeployedDate>
         <deployedBy>admin</deployedBy>
         <categories/>
         <migrationDate>0</migrationDate>
      </LightProcessDefinition>
</set>
```

La información contenida en esta respuesta XML muestra un conjunto de los procesos desplegados en el motor de procesos. En este caso se observa sólo un proceso, el proceso "Enviar Mensaje" del ejemplo anterior.

De la misma manera que se puede obtener la definición de los procesos instalados en el motor de Bonita, también es posible recuperar la información de las instancias que se encuentran ejecutándose o ya terminadas dentro del motor de procesos.

Por ejemplo, algunos de los datos que podemos encontrar al realizar la siguiente solicitud,

```
http://192.168.1.40:8080/bonita-server-rest/API/queryRuntimeAPI/getProcessInstance/Enviar_Mensaje--1.0--1
```

que pertenece a la instancia número 1 del proceso "Enviar Mensaje", son:

- **ProcessUUID**: el identificador del proceso
- **State**: el estado del proceso (ready, finished, aborted, cancelled, failed, etc)
- **StartedBy**, **EndedBy**: los actores que comenzaron y terminaron la instancia.
- **Activities**: las tareas que se encuentran dentro del proceso, como por ejemplo "Ingresar y Enviar mensaje", o "Enviar mensaje a través del conector", con sus respectivas variables de identificación, estado, actores, fecha de inicio y fin, conectores utilizados, etc.
- **InvolvedUsers**: actores que participaron en la instancia.
- **ClientVariables**: las variables utilizadas dentro del proceso.
- **VariableUpdates**: los valores que fueron actualizando las variables de la instancia a través de su ejecución en el tiempo, por qué actores, y en qué momento.

Al disponer de esta información en XML, y con el uso de un analizador sintáctico (o parser) como por ejemplo SimpleXML [32], que es utilizado dentro del lenguaje PHP, es posible recuperar los datos de cada propiedad que se encuentra dentro de la respuesta de la petición a la API REST de Bonita. SimpleXML es una extensión de PHP que proporciona un conjunto de herramientas muy simple y fácil de usar para convertir XML a un objeto que pueda ser procesado con selectores de propiedades normales e iteradores de arreglos.

La obtención del acceso a la API REST de Bonita y la consulta al motor de procesos desde la aplicación web de monitoreo con el que se obtendrán las respuestas en formato XML, se lleva a cabo con la utilización de un componente de *PEAR* [33] llamado *HTTP_Request2* [34].

PEAR es un *framework* y un sistema de distribución de componentes que está orientado a la comunidad de desarrolladores en PHP para promover la reutilización de código que realiza tareas comunes, y que tiene como metas promover una biblioteca de código fuente estructurada, mantener un sistema de distribución y mantenimiento de paquetes de código, y promover un estilo de codificación estándar. Cada componente de PEAR es un proyecto independiente dentro del framework, en donde se pueden encontrar algunos como:

- Autenticación
- Base de datos

- Encriptación
- Sistemas de archivos
- HTTP
- Imágenes
- Logs
- Matemática
- Sistemas
- WebServices
- XML

En nuestro caso, para realizar las peticiones al motor de procesos de Bonita a través de la API REST, se utilizará el componente HTTP_Request2. Este componente le provee a las aplicaciones PHP una manera simple de realizar peticiones HTTP, que servirá para conectare, autenticarse y realizar las peticiones necesarias a la API REST de Bonita de la siguiente manera utilizando PHP:

```
$query = 'http://192.168.1.41:8080/bonita-rest-api/API/queryRuntimeAPI/getProcessInstance/Enviar_Mensaje--1.0--1');
$request = new HTTP_Request2($query, HTTP_Request2::METHOD_POST);
$request->setAuth('usuario', 'contraseña', HTTP_Request2::AUTH_BASIC);
$request->setBody('options=user:monitor');
$response = $request->send();
$sxe = new SimpleXMLElement($response->getBody());
```

Con este ejemplo podemos ver la simplicidad con la que se pueden obtener los datos en formato XML de la instancia "Enviar Mensaje" que se encuentra instalado dentro del motor de procesos de Bonita mediante una aplicación desarrollada en PHP. En la última línea del código del ejemplo, en la que se realiza un *new* de la clase *SimpleXMLElement*, se utiliza para asignar la respuesta de la petición HTTP enviada en formato XML del motor de procesos, a un objeto encargado de analizar sintácticamente esta respuesta y simplificar el acceso a las propiedades de la instancia del proceso.

Con el uso de estas herramientas, es posible desarrollar una aplicación web para obtener la información de las diferentes instancias de los subprocesos y unificar los datos de cada uno de estas para realizar la etapa de monitoreo de los procesos distribuidos.

Capítulo 6 - **Validación con un ejemplo**

En este capítulo se presentará un ejemplo de proceso de negocio y se aplicarán los pasos del ciclo de vida de los procesos, como el diseño y modelado. Luego procederá la descomposición, el monitoreo y unificación de los subprocesos.

Al comienzo se hará una introducción al ejemplo, donde se introducirá al lector acerca de las tecnologías utilizadas, y las posibilidades de utilizar estas tecnologías en conjunto con BPM.

Luego se llevará a cabo la descomposición del proceso original en varios subprocesos, y se realizará una arquitectura de 3 servidores en donde se llevarán a cabo la ejecución de los subprocesos. Se hablará de la configuración de la arquitectura y el acceso seguro a Bonita OS.

Se presentará una aplicación web de monitoreo, junto con las herramientas y tecnologías utilizadas para el desarrollo de dicha aplicación, la comunicación entre los diferentes componentes de la arquitectura, y se mostrará el proceso unificado a partir de los subprocesos desplegados en los diferentes servidores.

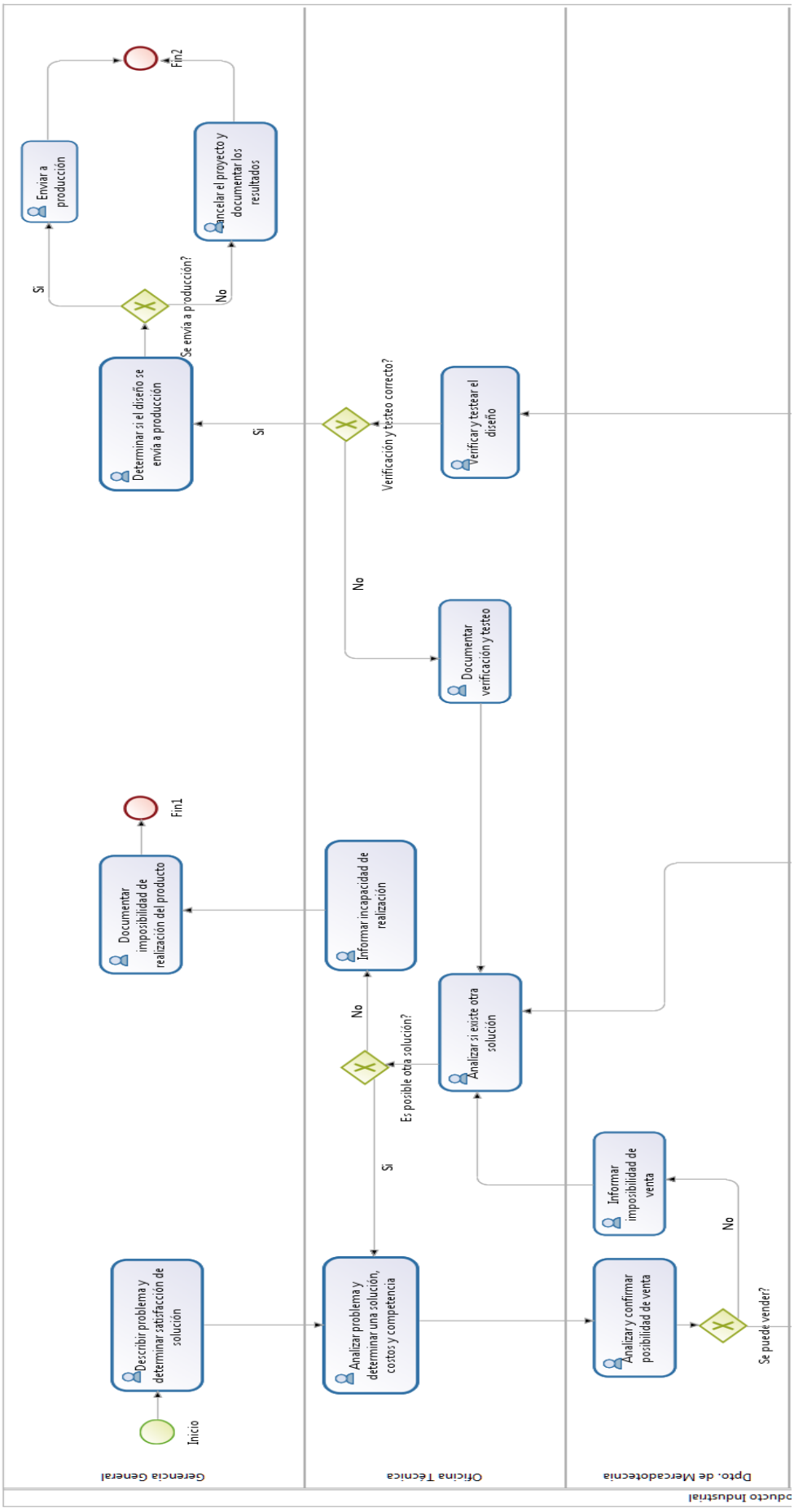
6.1. Elección del ejemplo: Diseño 3D de Producto Industrial

A lo largo del capítulo se irá comentando sobre de la arquitectura, la disposición y configuración de los servidores, las herramientas utilizadas en el desarrollo, la manera en que se recolecta la información y se realiza el monitoreo del proceso de negocio distribuido, entre otras cosas. Sin embargo, me es transcendental explicar una parte importante del proceso de negocio distribuido, en la cual hago elección de una herramienta en la que es necesaria una alta capacidad de procesamiento, inclusive para computadoras personales de última generación, en las que la conclusión de un trabajo cualquiera podría requerir varias horas de procesamiento de forma exclusiva en el uso del procesador.

El ejemplo que abarcará la mayor parte del capítulo estará basado en el Diseño 3D de un producto que se da habitualmente en el ámbito industrial. Es indistinto si el proceso de negocio de realiza en una empresa grande, mediana, o pequeña, ya que el tiempo de procesamiento del diseño en 3D del producto está dado por el detalle que contiene el modelo, por lo que el tiempo para efectuar la conversión del modelo al diseño 3D es el mismo, independientemente del tipo de empresa en la que se realiza.

En este ejemplo consideraremos que la empresa se compone de una Gerencia General, una oficina técnica, un departamento de mercadotecnia, un departamento de ingeniería, y un equipo de diseño.

El proceso de negocio de ejemplo puede verse en la Figura 6.1.


Gerencia General
Oficina Técnica
Dpto. de Mercadotecnia
Inicio
Describir problema y determinar satisfacción de solución
Analizar problema y determinar una solución, costos y competencia
Analizar y confirmar posibilidad de venta
Se puede vender?
No
Si
Informar imposibilidad de venta
Analizar si existe otra solución
Es posible otra solución?
No
Si
Informar incapacidad de realización
Documentar imposibilidad de realización del producto
Fin1
Documentar verificación y testeo
Verificar y testear el diseño
Verificación y testeo correcto?
No
Si
Determinar si el diseño se envía a producción
Se envía a producción?
No
Si
Cancelar el proyecto y documentar los resultados
Enviar a producción
Fin2

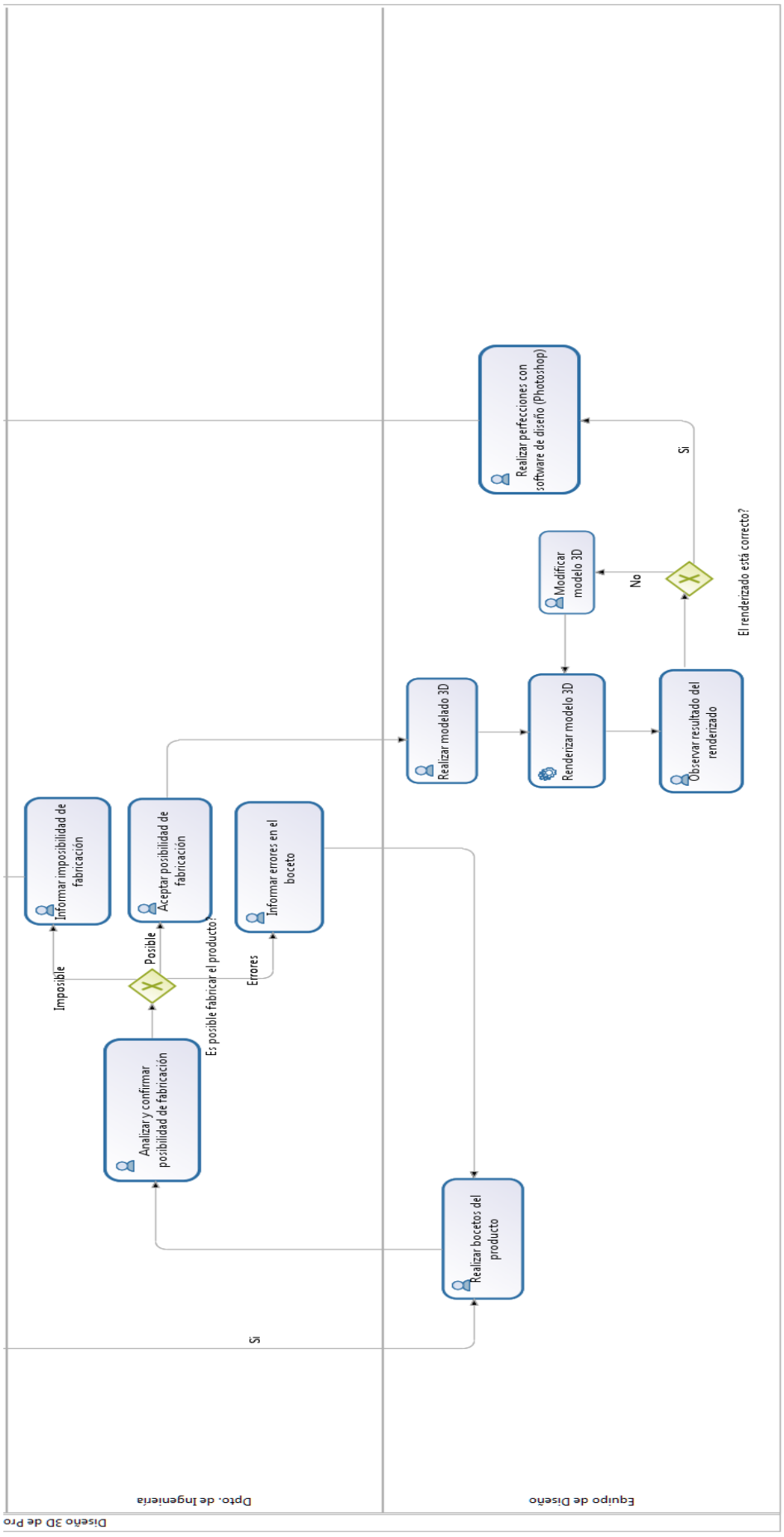


Figura 6.1. Diseño 3D de un producto industrial

En este ejemplo, suponemos que la empresa se encarga de realizar los productos para un cliente, o bien, para la misma empresa que fabrica y vende los productos.

Esta empresa tiene una Gerencia General, en lo más alto de la pirámide organizacional, encargada de plantear los problemas a solucionar, y tomar las decisiones de fondo de la empresa como así también de inicio y finalización de cada uno de los proyectos.

Debajo de la Gerencia General, se encuentra la Oficina Técnica, que se encarga de analizar el problema y determinar posibles soluciones de acuerdo a factores como costos, competencia, entre otros. Además es el nexo entre la Gerencia General y los equipos de Ingenieros, Mercadotecnia, y Diseñadores, y verifica que se realicen las tareas que conducirán al resultado esperado del producto.

El Equipo de Mercadotecnia se encarga de analizar si el producto es capaz de comercializarse en el mercado, o si es necesario realizar cambios en las especificaciones que ha definido la Of. Técnica para obtener una mejor respuesta comercial.

Una vez que Mercadotecnia ha acordado la posibilidad de inserción del producto al mercado, el Equipo de Diseño realiza los bocetos del producto como una aproximación a la idea de modelo que tendrá dicho producto.

Debido a que los diseñadores pueden llegar a realizar bocetos que vayan más allá de la posibilidad de fabricación, el Equipo de Ingenieros se encarga de aprobar, enviar modificaciones o rechazar tales bocetos. En caso de que sean aprobados, los diseñadores deben realizar el modelo en 3D del producto, utilizando programas especializados basados en herramientas CAD (*Computer-aided design* – o en español Diseño Asistido por Computadora), para obtener una perspectiva real de producto final.

Una vez obtenido el modelo 3D del producto, es enviado la Oficina Técnica para que se le realicen las pruebas necesarias y se apruebe de acuerdo a las especificaciones requeridas por la Gerencia General, que es la encargada de rechazar o aprobar el proyecto, y enviarlo a producción o al cliente que lo solicitó.

Obsérvese que este ejemplo no pretende explicar el proceso real de un diseño en 3D de un producto industrial, sino una aproximación a éste. Para lograr un proceso de negocio completo, con todos los detalles que se merece, sería necesario consultar a especialistas dedicados al tema del ejemplo, pero esto extendería el tema principal de la presente tesina, perdiendo el foco de lo importante de la misma, es decir, la descomposición y el monitoreo de procesos de negocio distribuidos. Por esto es que es fundamental que el lector no se centre en el ejemplo en sí, sino en las potenciales posibilidades que se pueden aprovechar del mismo.

6.1.1. Modelado 3D, renderización y software 3D

En computación, las tres dimensiones son el largo, el ancho y la profundidad de una imagen. Técnicamente hablando el único mundo en 3D es el real, la computadora sólo simula gráficos en 3D, pues, en definitiva toda imagen de computadora sólo tiene dos dimensiones, alto y ancho (resolución). En la computación se utilizan los gráficos en 3D para crear animaciones, gráficos, películas, juegos, realidad virtual, diseño, etc.

Modelado 3D

El modelado es una técnica que se utiliza para ir dando forma a objetos. Por lo general, el modelo visual suele ser el modelo 3D que los diseñadores manejan, dejando las fórmulas a procesos computacionales. Esto quiere decir el modelado 3D visual se acerca más a la imagen 3D final que se mostrará al renderizarse.

En la actualidad es utilizada para la producción de películas para cine y televisión, publicidad, juegos y efectos especiales. También se utiliza en los sectores profesionales de la medicina, la ingeniería o la arquitectura es utilizado para el diseño de piezas industriales, maquetas en la construcción de edificios, entre otras muchas más opciones, este programa además de estas opciones permite realizar renderizados espectaculares. Es decir que permite ambientar modelos en 3D para hacerlo realistas y crear animaciones.

Renderización

El proceso de transformación de un modelo en 3D hacia una imagen 3D es llamado renderización (*rendering*).

Renderizado es un término usado en informática para referirse al proceso de generar una imagen desde un modelo. Este término técnico es utilizado por los animadores o productores audiovisuales y en programas de diseño en 3D. La imagen resultado de la renderización es una imagen digital (raster).

En el caso de los gráficos en 3D, el renderizado puede hacerse lentamente (prerenderizado) o en tiempo real. Son millones los cálculos matemáticos que deben realizarse para procesar un modelo en 3D y resultar en una imagen renderizada.

En general, en el proceso de cálculo se pueden tener en cuenta tonalidades, texturas, sombras, reflejos, transparencias, translucidez, iluminación (directa, indirecta y global), profundidad de campo, desenfoques por movimiento, ambiente, etc.

Además a todo eso hay que agregarle los distintos objetos poligonales en 3D de la escena. Todos estos cálculos producen una simple imagen final. Por esta razón el proceso de creación de películas en 3D, **necesita mucho tiempo y gran capacidad de procesamiento computacional**. Un sólo segundo de película suele estar constituido por 24 cuadros de imagen, por lo que si pensamos que tenemos un capítulo de una serie animada de solo 25 minutos, tendríamos que renderizar 36.000 imágenes. El software encargado de esta renderización es llamado motor de renderizado.

El render es importante porque es el resultado final en forma de imagen que entrega el computador, específicamente, a través del motor de render dentro de un programa 3D. [35]

Programas de diseño 3D

Los últimos años han visto la aparición del concepto BIM (*Building Integrated Model*), un modelo de intercambio e interoperatividad entre los programas de diseño asistido por computadora de tipo general (AutoCAD, ArchiCAD, Sketchup) y programas específicos de las especialidades.

El software CAD (Computer-aided design – o en español Diseño Asistido por Computadora) está en continua evolución, adaptándose cada vez más a los nuevos tiempos, nuevas tecnologías, y formas de trabajo. El uso de las tres dimensiones es cada vez más frecuente, y por ello ese es un aspecto que se mejora en cada versión de los programas, ganando en estabilidad, velocidad y prestaciones. [36]

Algunos de los programas que se utilizan hoy en día son: Autodesk Revit [37], Autodesk 3D Studio Max [38], Autodesk Inventor [39] SketchUp [40], Rhinoceros 3D [41], Blender [42], entre muchos más.

La elección de cada uno de estos programas está fundamentada en el proyecto que se desea realizar, esto es, por ejemplo, Revit se utiliza como un software de construcción y diseño de edificios; 3ds Max es utilizado para el modelado, animación y renderización en 3D para creadores de juegos, cine y gráficos en movimiento.

Otro software de diseño es Rhinoceros 3D, que es comúnmente usado para el diseño industrial, la arquitectura, el diseño naval, el diseño de joyas, el diseño automotriz, CAD/CAM, prototipado rápidos, ingeniería inversa, así como en la industria del diseño gráfico y multimedia. [43]

6.1.2. Renderizado en Cloud Computing

El propósito del ejemplo, es utilizar *Cloud Compunting* para realizar el renderizado del modelo 3D en la nube. Hoy en día existe un servicio que brinda la empresa Autodesk llamado "*Autodesk 360 Renderin*", en el cuál, al utilizar algunos de sus productos tales como AutoCAD, 3ds Max o Revit, le permite al usuario subir a la nube sus modelos y renderizarlos en ella.

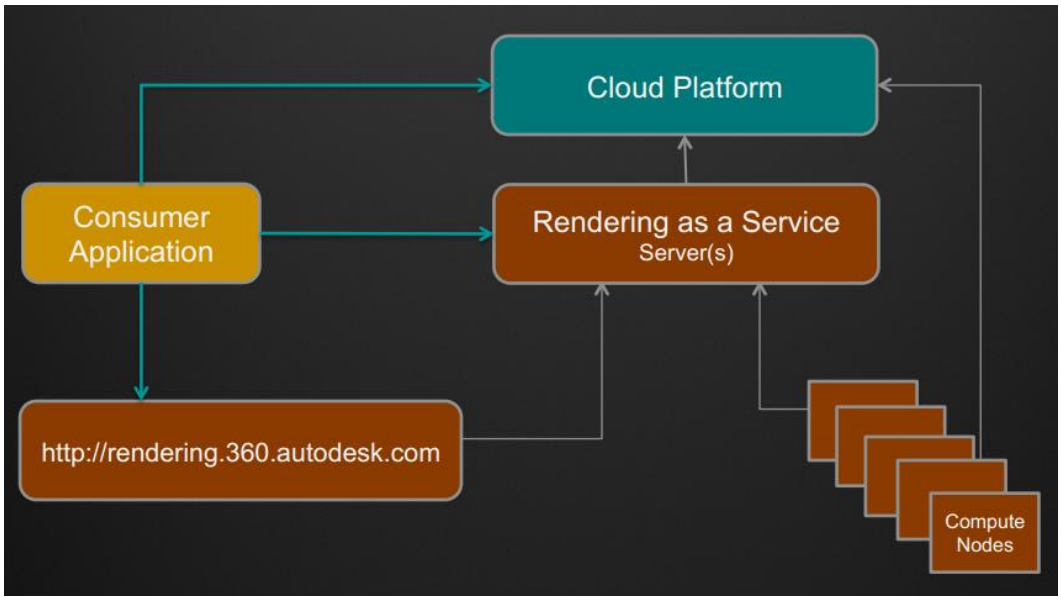


Figura 6.2. Rendering como Servicio en productos Autodesk [44]

Este servicio, además de ofrecer un renderizado rápido y eficiente, incluso de mejor manera que en una computadora personal, también brinda el servicio de almacenamiento de los modelos y de los renderizados, utilizando una metodología de *hash* que minimiza la duplicación de los datos en el cloud.

Al almacenar los modelos 3D, se puede llevar a cabo diferentes tipos de renderizado directamente en la nube. El usuario puede ir modificando las diferentes configuraciones del renderizado para obtener la mejor imagen posible a su gusto.

Por otra parte, a través del modelo 3D, y como parte de una de las características principales que le da el nombre al servicio (360º), es posible realizar una visión denominada "panorama", que permite al usuario visualizar el modelo renderizado en 360º a través del portal de Autodesk, dando una perspectiva de realidad virtual al modelo 3D.

Al comparar el renderizado en términos de performance y calidad de imagen entre un programa como Revit y el servicio de renderizado en el *cloud*, se observa que la imagen final renderizada en el *cloud* tiene una mejor definición, respetando las texturas y la iluminación que han sido definidas, y además puede tardar de un 75% a un 90% menos en el tiempo de renderizado que en un programa situado en una computadora personal. Esto quiere decir que, por ejemplo, si en una computadora personal, el renderizado de un modelo 3D tarda 9 horas en finalizar, utilizando el servicio de renderizado en la nube, tardaría 1 hora. [44] [45]

La comparación puede verse en la Figura 6.3.

Figura 6.3. Renderizado en Revit vs Renderizado en Cloud [44]

Claramente se puede ver que la utilización del servicio de renderizado en la nube tiene varias ventajas, aunque cabe destacar, que como todo servicio de la nube, sigue el concepto de pago por uso. Si bien en el ejemplo presentado en la tesina se va a hacer uso de este servicio de una manera simulada, se debe analizar si es la mejor elección de acuerdo a la infraestructura de hardware que se dispone en la empresa, y a la carga de trabajo que existe en la misma.

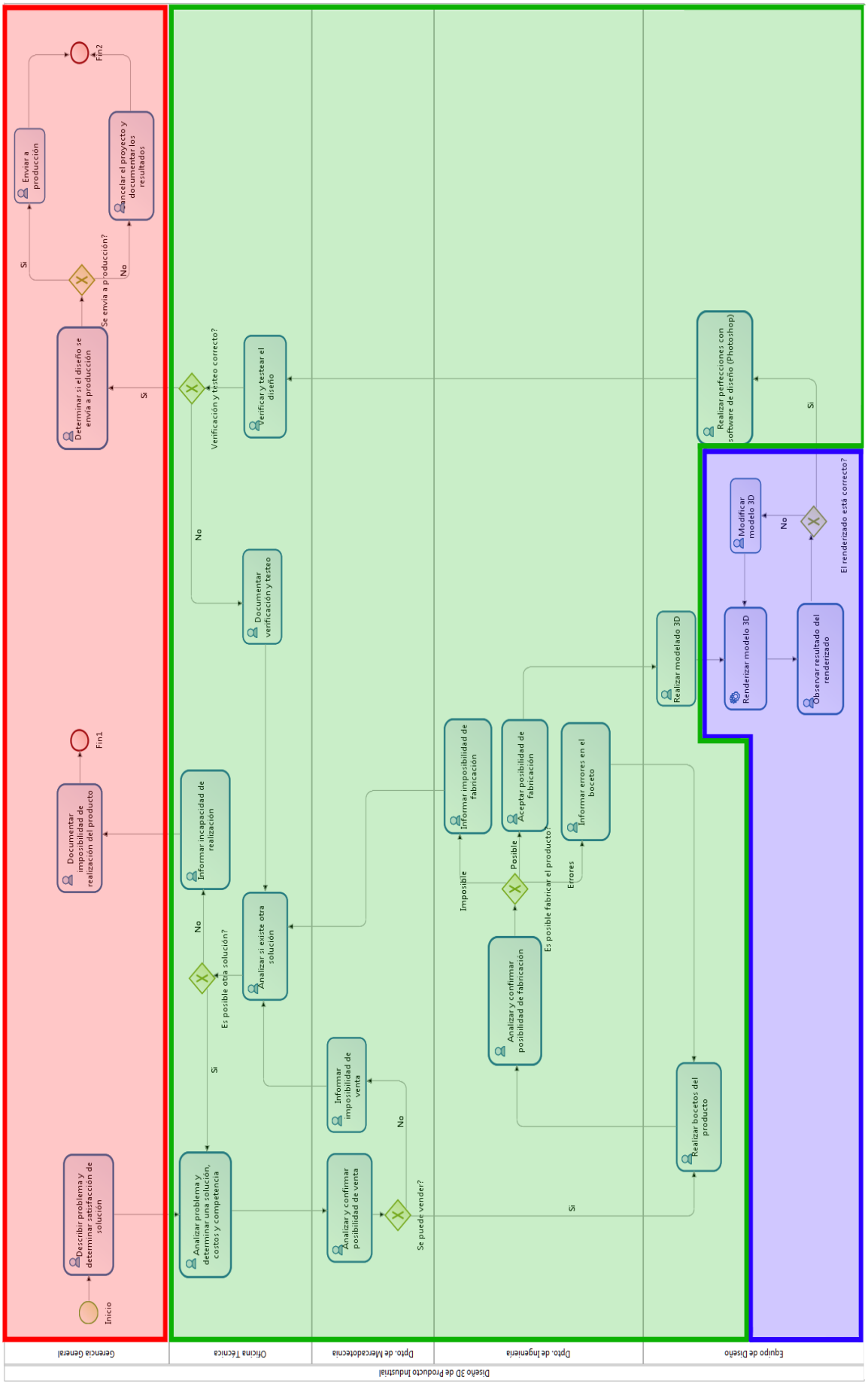


Figura 6.4. División de las actividades en los servidores

6.2. Descomposición y distribución de las actividades

Para llevar a cabo la descomposición del proceso del ejemplo, vamos a utilizar 3 servidores con sus respectivos BPMS, en donde se situarán las actividades de acuerdo al servidor que pertenezcan.

En la Figura 6.4 vemos como estarán divididas las actividades en cada servidor de acuerdo a esta división, en donde se pueden observar 3 conjuntos de actividades:

- El **primer servidor**, coloreado en rojo, es el que va a manejar los datos sensibles de la empresa, el que puede estar detrás de un firewall o tener una base de datos con acceso restringido, para asegurar la seguridad y confidencialidad de los proyectos. Vamos a hacer que sea exclusivo de la Gerencia General, que es la que debería manejar estos datos.
- El **segundo servidor**, coloreado en verde, va a ser al que vayan a acceder los diferentes equipos de Ingenieros, Mercadotecnia, Diseñadores y la Oficina Técnica, para llevar a cabo los proyectos.
- Y el **tercer servidor**, coloreado en azul, va a estar situado en el *Cloud*, el cuál será encargado de renderizar los modelos 3D que realicen y carguen los diseñadores en la nube.

Al momento de realizar la descomposición es necesario identificar aquellas actividades que se encuentran conectadas a través de flujos de control con actividades que pertenecen a otros servidores. Al realizar esto seremos capaces de determinar en dónde se utilizará el conector que inicia las instancias (visto en el Capítulo 5), que marca la finalización del subproceso, y cual actividad será la que marque el punto inicial del siguiente subproceso.

Una manera simple de identificar esto es observando los flujos de control de las actividades y la orientación que éstos tienen. Por ejemplo, si sabemos que las actividades del *lane* de la Gerencia General van situados en el Servidor 1, y las actividades del *lane* de la Oficina Técnica van en el Servidor 2, en el caso de la actividad "Describir el problema y determinar la satisfacción de la solución" (Gcia. Gral.) que mantiene un flujo de control con la actividad "Analizar el problema y determinar una solución, costos y competencia" (Of. Téc.), al ver la orientación del flujo de control que va desde la actividad de la Gcia. Gral. a la actividad de la Of. Téc., podemos determinar que la actividad "Describir el problema y determinar la satisfacción de la solución" marca la finalización del subproceso, por lo tanto, aquí es donde debemos hacer uso del conector, mientras que la actividad "Analizar el problema y determinar una solución, costos y competencia" será la primer actividad del siguiente subproceso.

En las siguientes secciones se identificarán cada uno de estos casos y se presentarán cada uno de los subprocesos, llevando a cabo la descomposición del proceso de negocio original, y nombrando a cada subproceso con una letra (A, B, C, etc), manteniendo el orden del flujo de control del proceso general.

6.2.1. Descomposición de las actividades del Servidor 1

Al observar el *lane* de la Gerencia General, el cual va a estar enteramente en el Servidor 1, se pueden identificar 3 flujos de control que provienen o se dirigen a actividades de otros servidores (o *lanes* en este caso). Estas actividades de la Gerencia General son:

1. Describir el problema y determinar la satisfacción de la solución
2. Documentar la imposibilidad de realización del producto
3. Determinar si el diseño se envía a producción

Podemos ver que el flujo de control de la primera actividad se dirige hacia una actividad en otro servidor, mientras que las otras dos reciben el flujo de control. Por lo tanto, la actividad 1 será parte del subproceso que haga uso del conector de ejecución distribuida (CED) para instanciar un subproceso en otro servidor, mientras que las actividades 2 y 3 serán el inicio de dos subprocesos individuales.

Al realizar la descomposición, los subprocesos quedarían de la siguiente manera:

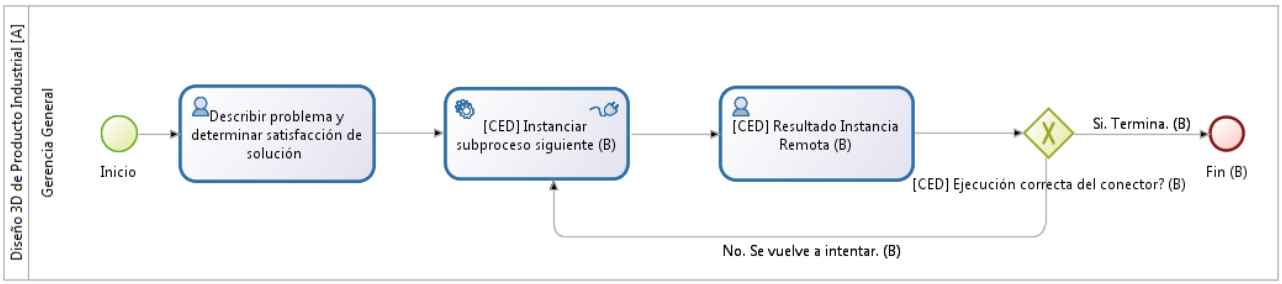


Figura 6.5.1. Subproceso A en el Servidor 1

En la figura 6.5.1 podemos observar al subproceso A, que es donde comienza el proceso de negocio original. Debido a que la siguiente actividad corresponde a la senda del servidor 2, se debe hacer uso del conector para comenzar con el subproceso B.

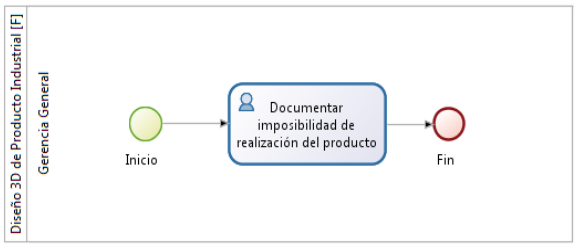


Figura 6.5.2. Subproceso F en el Servidor 1

Luego nos encontramos con los subprocesos F y G, en las figuras 6.5.2 y 6.5.3 respectivamente. Estos dos subprocesos tienen actividades que conducen a la finalización del proceso original, por lo tanto no es necesario utilizar el conector para instanciar otro subproceso.

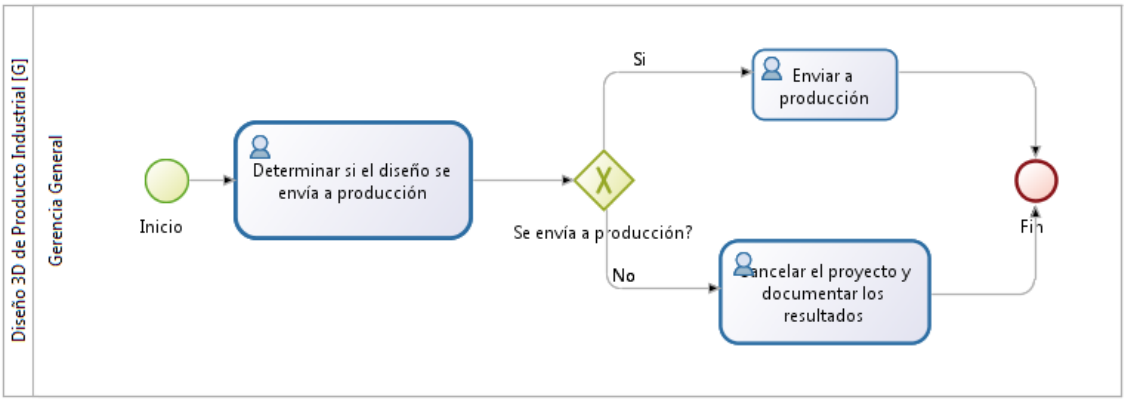


Figura 6.5.3. Subproceso G en el Servidor 1

6.2.2. Descomposición de las actividades del Servidor Cloud

Antes de comentar cómo se van a conformar los subprocesos del servidor 2, vamos a ver cuál es el subproceso que estará situado en el servidor de la nube, ya que al tener un subproceso de otro servidor en el medio de sus actividades (a continuación se mostrará este subproceso), será necesario dividir las actividades para formar subprocesos intercomunicados en el mismo servidor.

Como dijimos anteriormente, vamos a utilizar el servidor de la nube para realizar el renderizado de los modelos 3D, por lo tanto, el subproceso en este servidor tendrá que comenzar con la actividad de "Renderizar modelo 3D", tal como se encuentra en la Figura 6.4 sombreado con el color azul.

En la Figura 6.6 podemos ver como quedaría el subproceso.

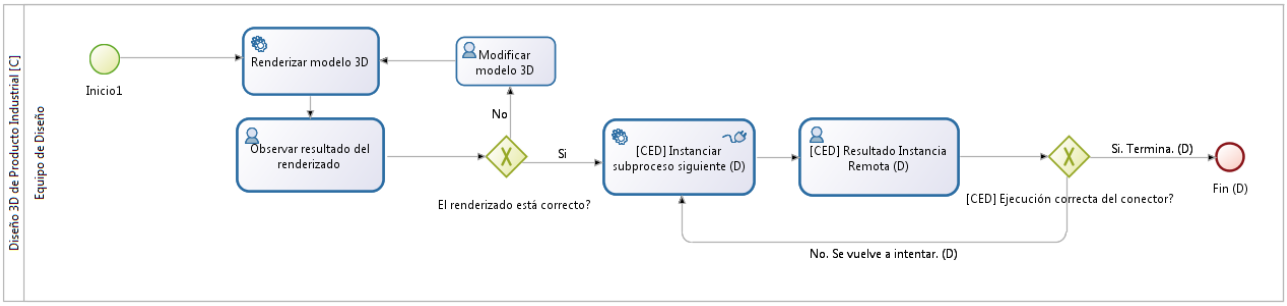


Figura 6.6. Subproceso C. Renderizado del modelo 3D situado en el servidor del Cloud

El subproceso comienza recibiendo el modelo 3D, al cual se le realiza el renderizado y finaliza enviando el resultado al subproceso D para que el equipo de diseño le haga los ajustes necesarios.

6.2.3. Descomposición de las actividades del Servidor 2

La disposición de los subprocesos del servidor 2 no es tan simple como las de los servidores 1 y el servidor en la nube. Al encontrarse el subproceso del servidor de la nube en el medio del flujo de control es necesario subdividir al servidor 2 en por lo menos 2 subprocesos (aunque más adelante veremos que en realidad son 3 subprocesos).

En este servidor, el cual es utilizado por la Oficina Técnica, el Depto. de Mercadotecnia, el Dpto. de Ingeniería y el Equipo de Diseño, el inicio del primer subproceso, de acuerdo al flujo de control, lo marca la actividad "Analizar problema y determinar una solución, costos y competencia" (Oficina Técnica), y el nexo con el subproceso en la nube es la actividad "Realizar modelo 3D" (Eq. Diseño). Por lo tanto, luego de esta actividad va a ser necesario utilizar el conector de ejecución distribuida (CED) para que se pueda renderizar el modelo 3D en el servidor de la nube.

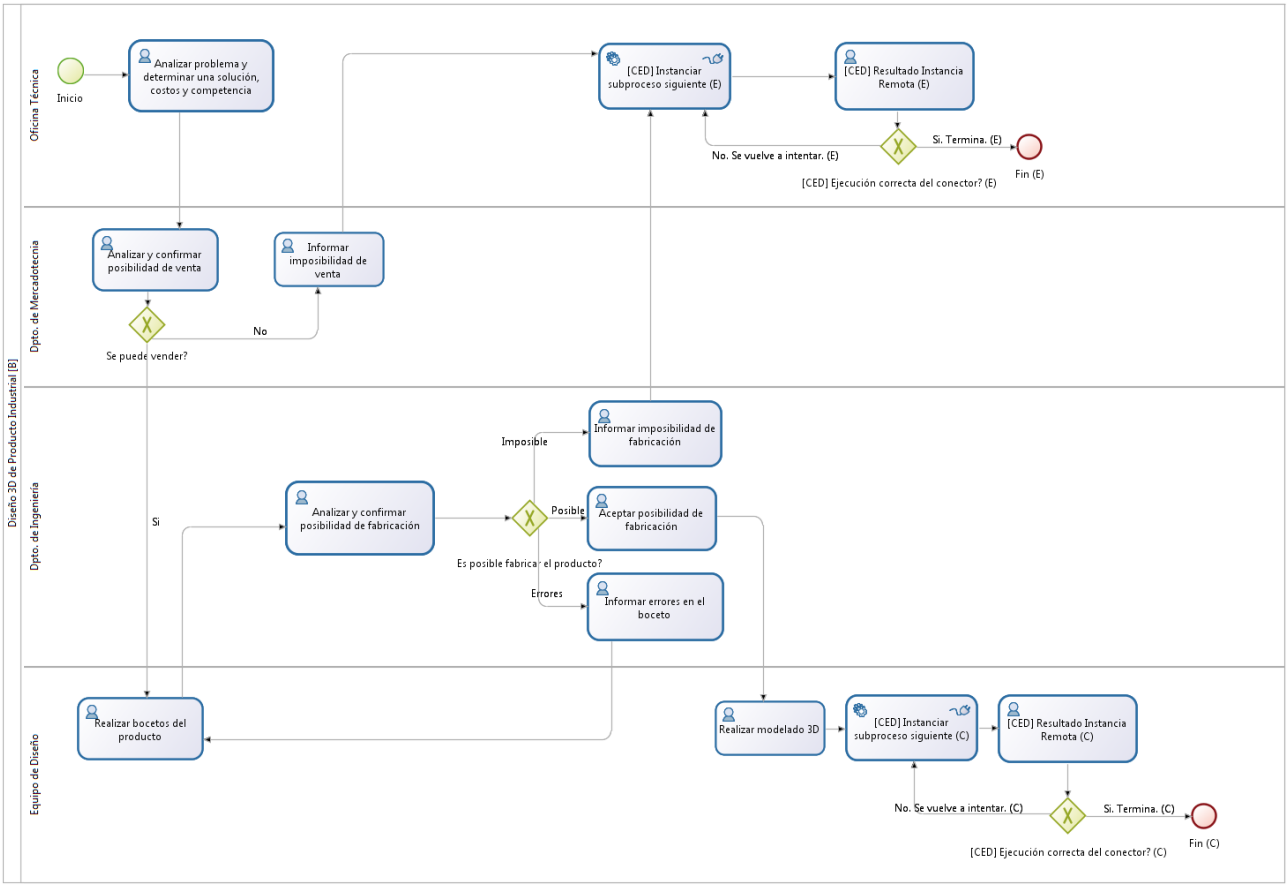


Figura 6.7.1. Subproceso B en el Servidor 2

Para construir los subprocesos del servidor 2, se puede tomar el subproceso en el servidor de la nube (Subproceso C) y utilizarlo como división de las actividades del servidor 2 que hay antes de este subproceso y las que hay después. De esta manera se puede descomponer las actividades del servidor 2 en dos subprocesos (uno antes del subproceso C y otro después). Sin embargo, esto sería incorrecto ya que hay una actividad entre las actividades que están situadas en el servidor 2 que es utilizada antes y después del subproceso situado en el servidor de la nube, esta actividad es "Analizar si existe otra solución".

Debido a esto, es necesario crear un subproceso adicional, que comience a partir de la actividad "Analizar si existe otra solución", que puede ser instanciado por los dos subprocesos que se encuentran antes y después del subproceso situado en el servidor de la nube.

Los subprocesos que derivan de las actividades del servidor 2 pueden verse en las Figuras 6.7.1, 6.7.2 y 6.7.3 (subproceso B, D y E respectivamente), en donde el último es utilizado por los subprocesos B y D.

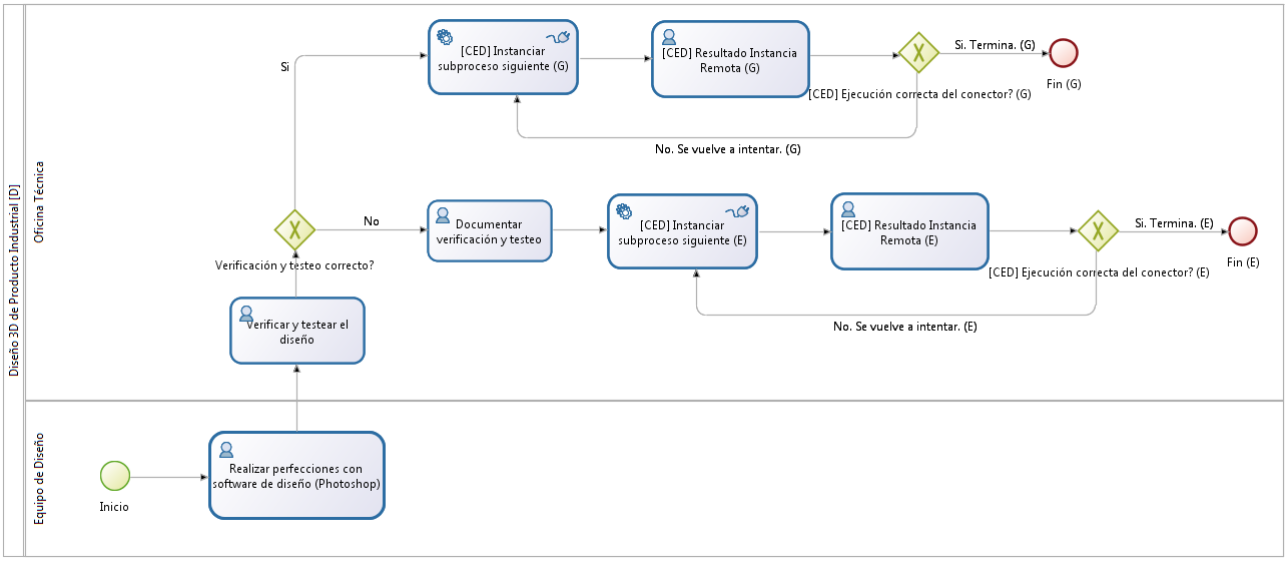
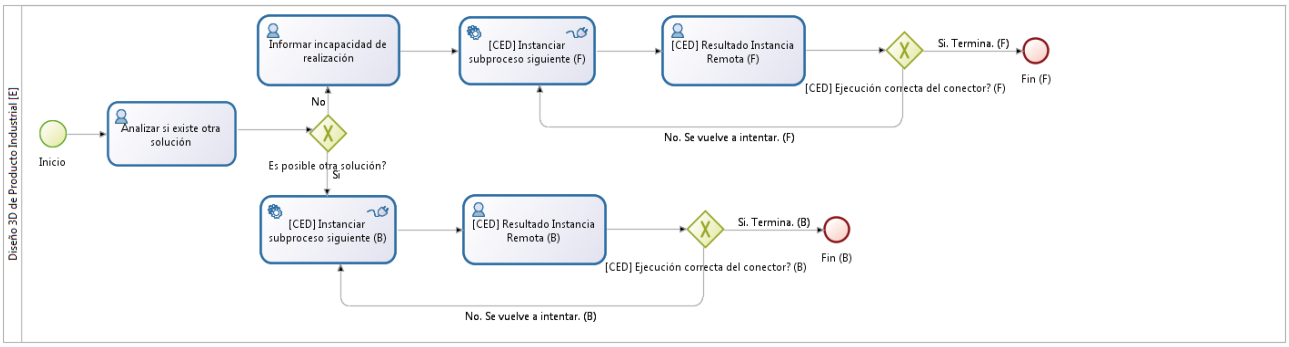


Figura 6.7.2. Subproceso D en el Servidor 2.



Figuta 6.7.3. Subproceso E en el Servidor 2.

6.3. Configuración de los servidores

Para la simulación de la solución de ejecución y monitoreo de los procesos de negocio se van a desplegar tres máquinas virtuales (una para cada servidor) utilizando Oracle VM VirtualBox [46], que simulan ser los servidores que ejecutan y monitorean los subprocesos.

Vale aclarar que se habla de una simulación, debido a que por cuestiones prácticas y económicas, no se contrató un servidor en el *cloud* para realizar las pruebas. Sin embargo, debido a que los espacios que son contratados en la nube de manera IaaS (*Infraestructure as a Service*) son máquinas virtuales en grandes servidores, podemos aproximarnos a la utilización de un servidor en la nube de una manera similar.

En cada uno de los nodos de la arquitectura se utiliza como sistema operativo Lubuntu 14.04, ya que es una distribución liviana de Linux que servirá para realizar las pruebas necesarias.

Para la ejecución de los procesos de negocio, se despliega un servidor Tomcat el cual contendrá el motor de procesos de Bonita Open Solution en su versión 5.10.

Además, cada nodo cuenta con un servidor Apache 2.4.7 y con MySQL 5.5.40 como motor de base de datos, que servirá para ejecutar la aplicación web de monitoreo y almacenar la información correspondiente a la ejecución distribuida de los subprocesos.

La configuración de red y del acceso a cada servicio de cada uno de los servidores es la siguiente:

- **Servidor 1**

  IP: 192.168.1.40

  Bonita Open Solution: http://192.168.1.40:8080/bonita

  Usuarios Bonita: gerencia, servidor1, servidor2

  Bonita Server Rest: http://192.168.1.40:8080/bonita-server-rest/

  Monitor de Procesos: http://192.168.1.40

- **Servidor 2**

  IP: 192.168.1.41

  Bonita Open Solution: http://192.168.1.41:8080/bonita

  Usuarios Bonita: ofitecnica, mercado, ingeniería, disenio, server1, server2, serverCloud

  Bonita Server Rest: http://192.168.1.41:8080/bonita-server-rest/

  Monitor de Procesos: http://192.168.1.41

- **Servidor Cloud**

  IP: 192.168.1.42

  Bonita Open Solution: http://192.168.1.42:8080/bonita

  Usuarios Bonita: servidor1, servidor2, serverCloud, disenio

  Bonita Server Rest: http://192.168.1.42:8080/bonita-server-rest/

  Monitor de Procesos: http://192.168.1.42

6.3.1. Configuración y seguridad en el acceso a la API REST

En la Sección 4.3.6 se comentó acerca de la Autentificación y Autorización para poder utilizar la API REST, sin embargo, en la configuración del archivo jaas-stadard.cfg, el acceso a la interfaz de la aplicación se lleva a cabo sin otro chequeo que el usuario y contraseña que se encuentra contenido en el archivo de configuración.

Para agregarle un nivel más de seguridad a la aplicación, es posible configurar este archivo para que chequee el usuario y contraseña contra la base de datos de usuarios de Bonita. De esta manera sería necesario disponer del usuario y contraseña tanto cuando se ingresa al User Experience, o se utiliza la API REST ya sea a través del conector desarrollado o a través de los servicios web que recolectan la información para el monitoreo del proceso distribuido.

A continuación se puede ver la configuración que se utiliza los archivos de seguridad de Bonita para otorgarle mayor seguridad a la aplicación:

```
BonitaAuth {
 org.ow2.bonita.identity.auth.BonitaIdentityLoginModule required;
};

BonitaStore {
/** Se utiliza para autenticarse contra un servidor de Bonita. En el ejemplo puede ser server 1,
server2 o cloudServer */
/** Sería quien es este servidor para el resto de los servidores */
 org.ow2.bonita.identity.auth.LocalStorageLoginModule required;
 org.ow2.bonita.identity.auth.BonitaRESTLoginModule required restUser="server1" restPassword="bpm";
};

/** Used by the REST server */
BonitaRESTServer {
/** Se utiliza para chequear a los clientes que quieren utilizar la API REST del servidor */
org.ow2.bonita.identity.auth.BonitaRESTServerLoginModule required
logins="server1,server2,cloudServer" passwords="bpm,bpm,bpm" roles="restuser,restuser,restuser";
};
```

6.4. Ejecución de los subprocesos de negocio

Una vez que se han configurado todas las variables de la instanciación remota, los subprocesos de negocio se ejecutan de la misma manera de que si se tratase de un único proceso de negocio, ya que de hecho, si se mira desde un punto de vista simplista, son un único proceso de negocio que tienen un conector que permite instanciar a otros procesos de negocio.

De hecho, Bonita nos brinda un conector para realizar la acción de instanciar un proceso (que se puede ver en [47] y [48]), pero sólo es posible hacerlo dentro del mismo servidor Bonita en el que se ejecuta el proceso, por lo que no sería posible llevar a cabo la instanciación entre servidores. En el código fuente de nuestro conector que instancia procesos remotos, presentado en la Sección 5.3, se puede ver que se hace uso de la función "*instantiateProcess(processUUID,vars)*" que es la que utiliza el conector que nos brinda Bonita para la instanciación de procesos, pero al realizar el cambio de contexto, la función se ejecuta en el servidor remoto y por lo tanto, se instancian procesos de ese servidor.

El proceso de negocio distribuido comienza con el subproceso A, a partir de la actividad "Describir problema y determinar satisfacción de solución" (Figura 6.8). Si observamos la distribución de las actividades de este subproceso en la Figura 6.5.1, podemos ver que existe además de esta actividad, otras actividades que corresponden a la instanciación del subproceso siguiente situado en el servidor 2.

Figura 6.8. Inicio del Subproceso A.

Una vez que se completan estos datos y se envía el formulario, el flujo de control continúa a la actividad encargada de instanciar el subproceso B, que en este caso es el siguiente subproceso. Pero para que pueda realizarse esto, el conector debe estar previamente configurado (en tiempo de diseño de los subprocesos) para que pueda conectarse con el servidor 2, ejecutar el subproceso B, enviar las variables de entrada al subproceso B y guardar el identificador de la instancia que se ejecuta en el servidor 2 en la base de datos local para el futuro monitoreo del proceso.

Con la configuración de los servidores indicada en la Sección 6.3 y sabiendo que el identificador único del subproceso B es "Diseno_3D_de_Producto_Industrial__B_", podemos configurar el conector tal como se ve en la Figura 6.9, y pasar las variables que sean necesarias de entrada al subproceso B.

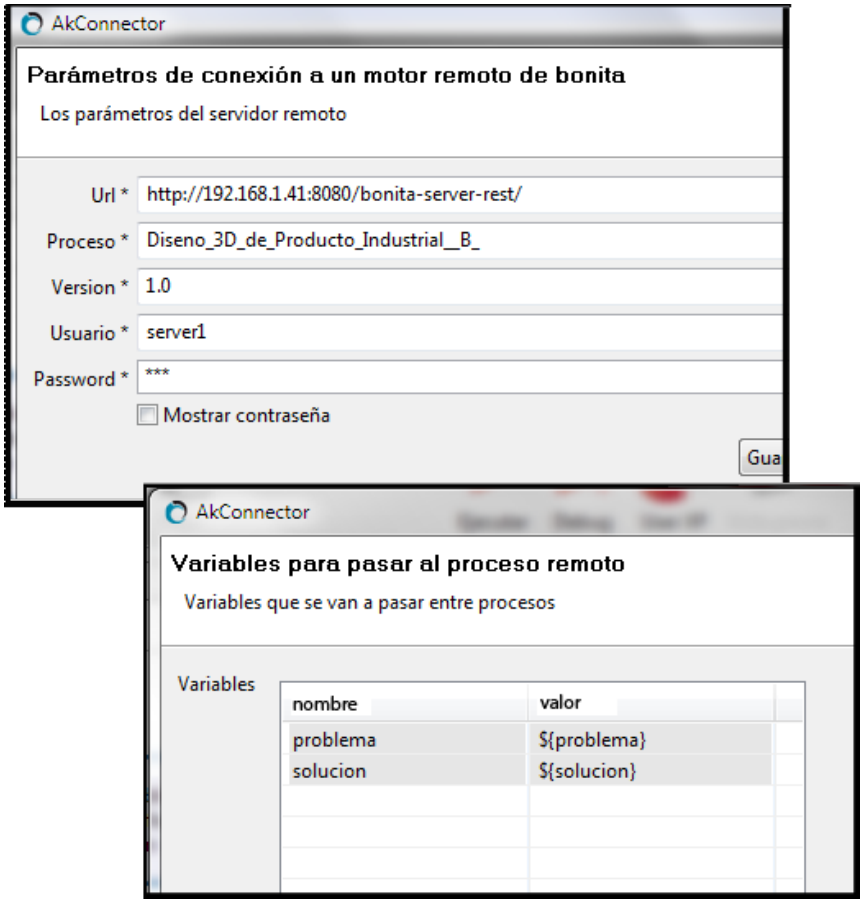


Figura 6.9. Configuración para instanciar el Subproceso B y variables enviadas al subproceso.

Una vez que se ejecuta la actividad que contiene al conector, es decir, "[CED] Instanciar subproceso siguiente (B)", se muestra el resultado de la ejecución del proceso remoto en la siguiente actividad (Figura 6.10), en donde se mostrará si se ha realizado correctamente la ejecución o se ha producido algún error (tal como se vio en la Sección 5.5).

Figura 6.10. Resultado de la Instanciación del Subproceso B.

Si bien sería recomendable que la gerencia, de la misma forma que ocurre con los demás actores cuando realizan la instanciación remota, no se involucre en el resultado de la ejecución del conector, estos resultados se muestran para facilitar las pruebas que se han realizado.

Lo ideal sería que el resultado de la instanciación remota sea transparente a estos actores, y en caso de la ocurrencia de algún error en la ejecución remota del subproceso, exista alguna forma de avisar al administrador del sistema (algo así como enviar un correo electrónico a través de un conector), para que el mismo solucione el problema y que logre continuar con la ejecución del proceso distribuido.

De la misma forma anteriormente mencionada se llevan a cabo todas las ejecuciones remotas entre los demás subprocesos.

6.5. Aplicación de monitoreo de procesos de negocios distribuidos

La aplicación de monitoreo está compuesta de dos secciones que si bien corresponden a la misma aplicación, se pueden separar por cuestiones de funcionalidad y visualización.

En una de las secciones se pueden observar las propiedades de los subprocesos, las propiedades de las instancias ejecutadas en Bonita y de las actividades de cada uno de los subprocesos, mientras que la otra sección consiste en visualizar el proceso de negocio unificado tal como se ha mostrado cuando se presentó el ejemplo, en la Figura 6.1.

Para el desarrollo de dicha aplicación, se utilizaron diferentes tecnologías que combinándolas, ayudan a conseguir un análisis de los subprocesos y sus actividades. Estas tecnologías y su uso dentro de la aplicación son, en su mayoría, tecnologías muy conocidas y usadas dentro del área de desarrollo de aplicaciones web, por esto es que sería innecesario extenderme demasiado en las características de las mismas. Sin embargo, a continuación describiré algunas de sus características y su uso dentro de la aplicación de monitoreo.

6.5.1. PHP, HTML y CSS

La aplicación de monitoreo es una aplicación web desarrollada en su mayoría con PHP, un lenguaje de programación que es uno de los pioneros en este tipo de aplicaciones y que tiene una amplia popularidad en el mundo.

Se utilizó este lenguaje ya que puede ser desplegado en la mayoría de los servidores web, en casi todos los sistemas operativos y plataformas, que al encontrarse dentro de licencias de software libre, es posible su instalación sin costo alguno. Además posee una amplia comunidad que desarrolla librerías o paquetes que ayudan al desarrollo de las aplicaciones, y que las ceden al servicio de cualquier desarrollador de manera libre. Se utiliza PHP para darle la lógica del negocio a la aplicación.

Existen otras alternativas a esta tecnología tales como ASP.NET, Java o PERL, pero algunas de estas son pagas, no tienen una licencia libre o no cuentan con tanta documentación y librerías. Por este motivo se optó por el uso de PHP.

Para la visualización de la aplicación se utiliza HTML y CSS. HTML y CSS no son lenguajes de programación, ambos pertenecen al área de diseño gráfico y maquetación, por lo tanto podemos categorizarlos como lenguajes de maquetación o marcado. Con el uso de estos dos lenguajes, se genera la interfaz de usuario para la aplicación de monitoreo.

Estas tres tecnologías en conjunto hacen que sea posible realizar una aplicación web que sea accedida por un navegador web desde cualquier parte del mundo a través de Internet.

6.5.2. MySQL

MySQL es un sistema de administración de bases de datos (*Database Management System, DBMS*) para bases de datos relacionales que proporciona un servidor de base de datos SQL (*Structured Query Language*) muy rápido, multi-threaded, multi usuario y robusto. El servidor MySQL está diseñado para entornos de producción críticos, con alta carga de trabajo así como para integrarse en software para ser distribuido.

Al ser un sistema de gestión de bases de datos relacionales, MySQL almacena datos en tablas separadas en lugar de poner todos los datos juntos en un archivo.

MySQL tiene una doble licencia. Los usuarios pueden elegir entre usar el software MySQL como un producto *Open Source* (cualquiera puede usar y modificar el software) bajo los términos de la licencia GNU GPL o, en caso de realizar una aplicación comercial, pueden adquirir una licencia comercial estándar. [49]

En esta tesina se utiliza MySQL como motor de bases de datos en dos sitios:

- **En la ejecución del conector**: para guardar los identificadores de los procesos que son instanciados. Un detalle no menos importante es que se debe incluir en las dependencias del proceso (o subproceso) diseñado en Bonita la librería correspondiente a MySQL, de otra forma el proceso no se podrá comunicar con la base de datos.
- **En la aplicación de monitoreo**: se utiliza para conseguir los identificadores de los procesos instanciados. De esta manera es posible conseguir la información de los procesos desplegados en Bonita a través de la API REST.

6.5.3. jQuery y jTable

jQuery no es un lenguaje, sino una serie de funciones y métodos de Javascript. Por tanto, Javascript es el lenguaje y jQuery es una librería que podemos usar opcionalmente si queremos facilitar nuestra vida cuando programamos en Javascript. A veces nos podemos referir a jQuery como *framework* o incluso como un API de funciones, útiles en la mayoría de proyectos web.

Es uno de los complementos más usados en el desarrollo web, que se encuentra en millones de sitios en toda la web, debido a que facilita mucho el desarrollo de aplicaciones enriquecidas del lado del cliente (en Javascript), compatibles con todos los navegadores.

Además, es un producto serio, estable, bien documentado y con un gran equipo de desarrolladores a cargo de la mejora y actualización del *framework*. Otra cosa muy interesante es la dilatada comunidad de creadores de *plugins* o componentes, lo que hace fácil encontrar soluciones ya creadas en jQuery para implementar asuntos como interfaces de usuario, galerías, votaciones, efectos diversos, etc. [50]

Por otro lado, jTable es un componente de jQuery, tal como se ha descrito recientemente. Este componente está desarrollado para crear tablas CRUD (**C**reate, **R**ead, **U**pdate and **D**elete – o en español Crear, Obtener, Actualizar y Borrar) basadas en la comunicación mediante AJAX (Asynchronous JavaScript And XML – o en español JavaScript asíncrono y XML).

Al utilizar este *plugin* podemos manejar tablas en nuestra interfaz web sin tener que escribir HTML o Javascript, tan solo es necesario incluir una etiqueta *div*, luego el complemento se encarga de crear las tablas automáticamente y cargar los datos provenientes del servidor mediante AJAX. Solo es necesario escribir la lógica de las acciones CRUD anteriormente mencionadas. [51]

Una captura de pantalla de la aplicación de monitoreo puede verse en la Figura 6.11, en donde se observan las propiedades de una instancia del Subproceso B tales como, el identificador (*ProcessUUID*), el estado (*state*), quien inició (*startedBy*) y quién terminó (*endedBy*) la instancia, y las fechas de estos sucesos.

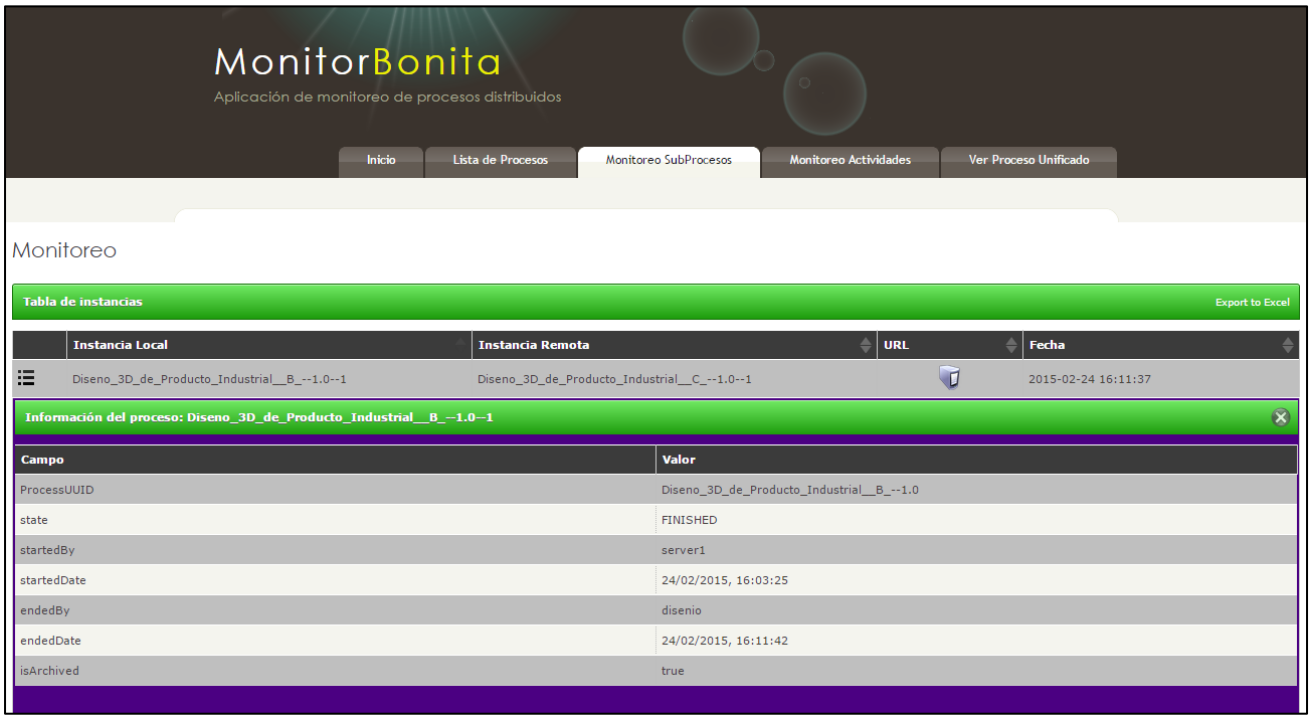


Figura 6.11. Aplicación de monitoreo de los subprocesos de negocios.

Además de monitorear los subprocesos es posible monitorear las actividades tal como se muestra en la Figura 6.12. En esta figura se pueden ver las actividades que componen al subproceso B, si ya han sido completadas (*FINISHED*) o están completándose (*READY*), el tipo de tarea y lo que tardaron en completarse cada una de éstas.

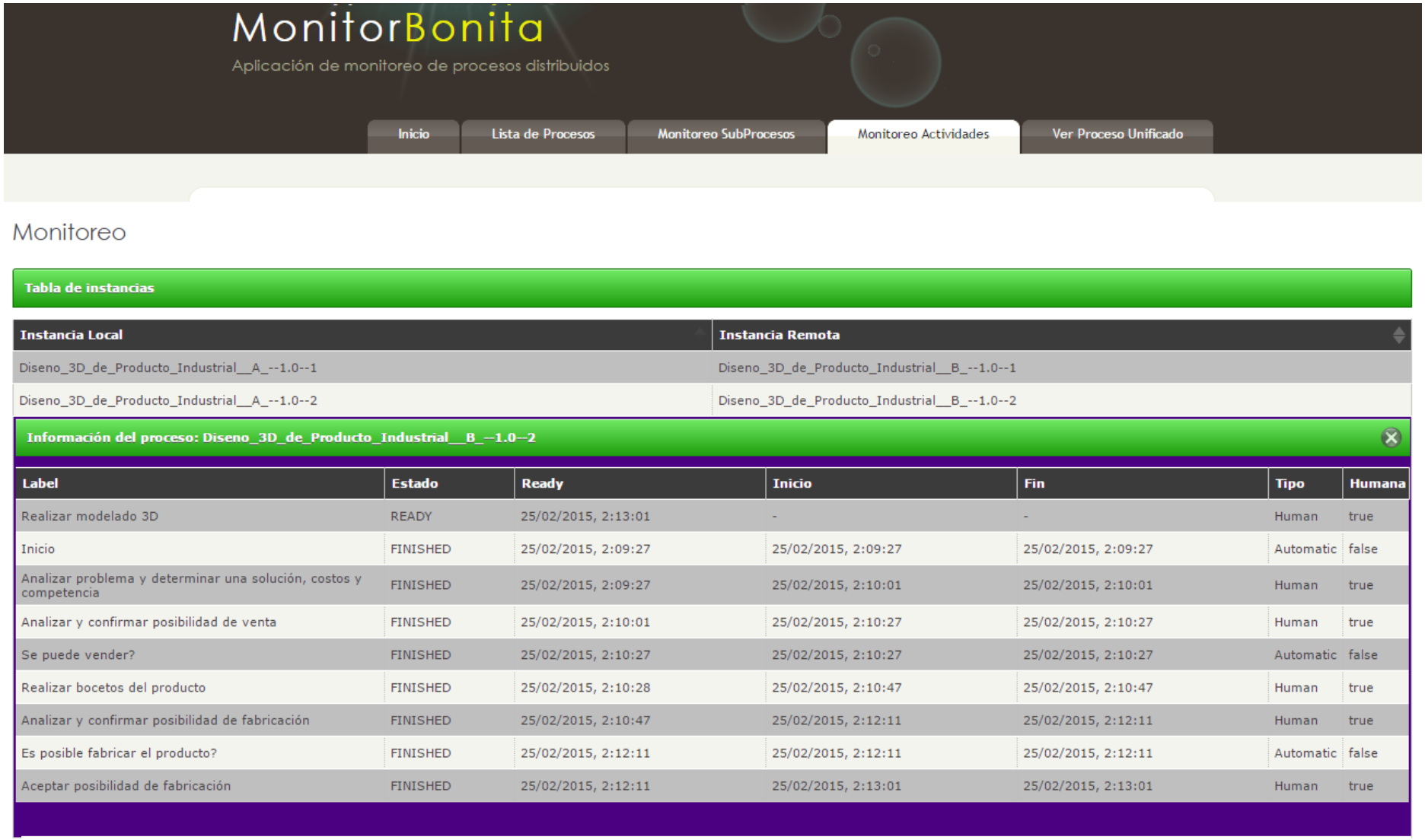


Figura 6.12. Monitoreo de las actividades del subproceso B

6.5.4. HTTP_Request 2 y respuestas XML de la API REST de Bonita

El paquete HTTP_Request2 es un paquete desarrollado por PEAR (*PHP Extension and Application Repository*) [33] que provee a las aplicaciones PHP una manera simple de realizar requerimientos HTTP. Soporta un amplio conjunto de configuraciones HTTP en donde se pueden realizar requerimientos POST con datos y archivos adjuntos, autenticación *basic* (sin encriptar) y *digest* (encriptada), cookies, proxies, entre otras. [34]

Esta librería es la encargada de establecer la comunicación con la API REST de Bonita y recolectar la información de las instancias desplegadas, tal como se ha mencionado en la Sección 5.6.

Una vez que se realiza el requerimiento HTTP, Bonita nos devuelve una respuesta con un archivo XML con los datos solicitados. Con el uso de SimpleXML [32], visto en la Sección 5.6, se tratan estos datos y se muestran en la aplicación web tal como vimos en la Figura 6.11 o Figura 6.12.

6.5.5. WebServices (NuSOAP)

Si bien con el uso de HTTP_Request2 es posible recolectar toda la información de las instancias y de los subprocesos involucrados en el proceso de negocio distribuido, no nos tenemos que olvidar que si utilizamos el servicio de *Cloud Computing*, la trasferencia de información desde y hacia la nube es uno de los factores (entre otros) que más encarecen el servicio, y a veces, la información que nos entrega Bonita es demasiada o contienen datos que no nos interesa, por lo que estaríamos pagando de más en cada uno de esos requerimientos.

Una solución a este inconveniente, que a corto plazo y con pocos subprocesos es mínimo, pero que a medida que van creciendo los procesos de negocio la factura del servicio se hace cada vez más grande, es utilizar servicios web.

Para la construcción de los servicios web se utiliza NuSOAP. NuSOAP es un conjunto de clases para desarrollar servicios web bajo el lenguaje PHP. Está compuesto por una serie de clases que nos hacen mucho más fácil el desarrollo de servicios web. Provee soporte para el desarrollo de clientes (aquellos que consumen los servicios web) y de servidores (aquellos que los proveen). [52]

En nuestro caso, la diferencia entre usar HTTP_Request (REST) y Servicios Web (SOAP) comienza por la API REST que nos ofrece Bonita. Bonita ya dispone de funciones que tienen respuestas a estos requerimientos, y no es posible cambiar estas respuestas a menos que modifiquemos el código fuente de Bonita con nuestras funciones, sin embargo esto resulta una ardua tarea y posiblemente lleve mucho tiempo de desarrollo y *testing*.

Para evitar modificar el código de Bonita, podríamos modificar directamente la respuesta que nos entrega el BPMS, lo que es posible a través de los servicios web, y retornar a la aplicación cliente, es decir, la que efectuó el requerimiento, los datos que sólo va a utilizar.

A partir de NuSOAP, como medio de comunicación, y PHP, lógica de la aplicación, la aplicación de monitoreo cliente realiza el requerimiento a la aplicación de monitoreo que contiene el subproceso solicitado. Esta última será la encargada de efectuar el requerimiento HTTP a la API REST de Bonita que está situada en el mismo servidor, trabajará con los datos de la respuesta que envíe Bonita obteniendo solamente los datos necesarios y devolverá la respuesta al servicio web con

tan sólo estos datos. Para poder comprenderlo de mejor forma, la Figura 6.13 muestra el comportamiento anteriormente mencionado.

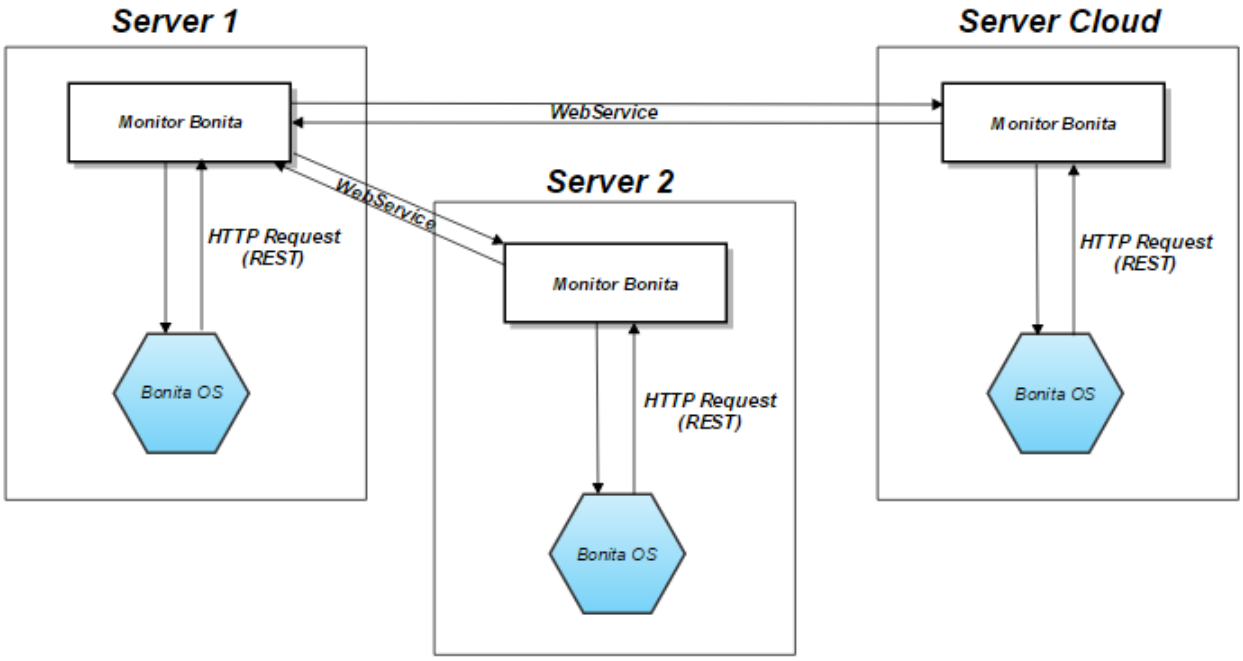


Figura 6.13. Ejecución de los Web Services

Un ejemplo de esto ocurre con la visualización unificada de los subprocesos para mostrar el proceso original, ya que solo son necesarias las actividades que forman al proceso de negocio, y debido a la respuesta de la definición de procesos que entrega Bonita, sólo las actividades contienen la información necesaria.

6.5.6. GraphViz

Graphviz es una aplicación de visualización de gráficos de código abierto que incluye un gran número de programas de trazado de grafico, existiendo versiones tanto para Windows como para Linux.

Los programas de diseño de Graphviz parten de descripciones de gráficos en texto plano, lo que les permite ser editados y no necesitar un programa adicional para ello. Los diagramas son realizados en varios formatos: imágenes (jpg o png), SVG (Scalable Vector Graphics, gráficos vectoriales en dos dimensiones) para páginas web, Postscript para ser incluido en PDFs u otros documentos.

Graphviz cuenta con muchas características para personalizar los diagramas tales como opciones para etiquetas, colores, fuentes, diseños en forma de tabla, estilos de línea, enlaces y formas. En la práctica, los gráficos suelen ser generados partiendo de fuentes externas de datos, pero también puede hacerse manualmente, bien editando un fichero de texto plano en lenguaje DOT o bien mediante un editor gráfico.

Dentro del paquete Graphviz se encuentran distintos programas para generar gráficos en función de unos parámetros determinados. Algunos de los programas más utilizados son:

- **dot**: realiza dibujos "jerárquicos" o por capas de gráficos directos. El algoritmo de representación trata de colocar todos los enlaces en la misma dirección (de arriba abajo o de izquierda a derecha) y después trata de evitar los cruces entre enlaces, y por último, reducir sus longitudes. Es la herramienta que se utiliza en gráficos con dirección.
- **neato**: crea "trazados elásticos". Se utiliza en grafos que tienen demasiados nodos (más de 100) ya que trata de minimizar los enlaces.
- **fdp**: crea "trazados elásticos". Implementa el heurístico *Fruchterman-Reingold* que incluye un solucionador por cuadrículas para manejar los gráficos más grandes y los grupos dentro de los gráficos indirectos. Se usa, al igual que neato, con grafos sin dirección.
- **twopi**: diagrama radial. Se sitúan los nodos en círculos concéntricos dependiendo de su distancia a un nodo raíz dado. [53]

Dentro de la aplicación de monitoreo, se utiliza GraphViz para generar la unión de todos los subprocesos de negocio, y mostrar el proceso unificado como si se tratase del proceso original mostrado en la Figura 6.1.

La manera de unificar a los subprocesos es como se ha mencionado en la sección anterior, es decir, a través de las propiedades que se pueden obtener de las respuestas XML que brinda la API REST. Con esta respuesta en XML, las actividades o compuertas no tienen una propiedad que las diferencie explícitamente, por lo que no disponemos de la misma cantidad de indicaciones que si lo viéramos en Bonita Studio.

Por ejemplo, para diferenciar una compuerta "*Inclusive"* de las tareas automáticas, es necesario tener en cuenta si contienen algún código o script dentro de la tarea, como así también ocurre con las tareas *Inicio* y *Fin*, donde la diferencia entre estos nodos se obtiene de acuerdo a la cantidad de aristas que llegan o salen del nodo.

Por esto es que fue necesario interpretar el XML, llegando a imponer ciertas pautas para graficar los procesos. Por el momento se llegaron a las siguientes conclusiones a tener en cuenta para realizar el diagrama:

- **Token Inicio**: no tiene ninguna arista que entre.
- **Token Fin**: no tiene ninguna arista que salga.
- **Compuertas**: traen la propiedad *<type>Automatic</type>* y no tienen ningún comportamiento dentro de la misma. *JoinType, SplitType* para diferenciarlas.
- **Tareas Humanas**: traen la propiedad *<type>Human</type>*
- **Tareas Automáticas**: traen la propiedad *<type>Automatic</type>* y tienen que tener algún comportamiento, si no lo tienen se tomaran como compuertas.
- **Tareas de Script**: igual que las automáticas. Se tratarán como automáticas.
- **Tareas que llaman a subprocesos**: traen la propiedad *<type>Subflow</type>*

- **Tareas que reciben mensajes**: traen la propiedad *<type>ReceiveEvent</type>* y seguramente una propiedad *incomingEvent* está seteada con parámetros.
- **Tareas que envían mensajes**: traen la propiedad *<type>SendEvents</type>* y seguramente una propiedad *outgoingEvents* está seteada con parámetros.
- **Tareas abstractas**: imposible de diferenciar con las compuertas Inclusive.

Siguiendo estas reglas, se puede armar el archivo *dot* que contiene la configuración para armar el grafo dirigido obteniendo las características de cada una de las actividades. Por ejemplo, el grafo unificado de los subprocesos contiene casi 500 líneas de datos. Un parte del archivo puede verse a continuación:

```
digraph G {
       graph [label="Proceso: Diseno_3D_de_Producto_Industrial__A_--1.0 --- 25/02/2015 3:28:46",
rankdir="LR", bgcolor=white, ranksep=0.2];
       node [label="\N",style=filled, width="1.0" height="0.8", shape=box];
       edge [fontsize=9 fontname=helvetica];

      subgraph cluster_subgrafo_1 {
      rank=same;
      width=11;
      nojustify=true;
      graph [color=blue, fontcolor=blue, fontsize=30, label="<<< gerencia >>>", rankdir="LR"];
      …
      Documentar_imposibilidad_de_realizacion_del_producto->Fin1;
      Cancelar_el_proyecto_y_documentar_los_resultados->Fin;
      }

      subgraph cluster_subgrafo_2 {
        graph [color=blue, fontcolor=blue, fontsize=30, label="<<< mercado >>>", rankdir="LR"];
      …
      }

      subgraph cluster_subgrafo_3 {
        graph [color=blue, fontcolor=blue, fontsize=30, label="<<< ofitecnica >>>", rankdir="LR"];
      …
      }

      subgraph cluster_subgrafo_4 {
        graph [color=blue, fontcolor=blue, fontsize=30, label="<<< disenio >>>", rankdir="LR"];
      …
      }

      subgraph cluster_subgrafo_5 {
        graph [color=blue, fontcolor=blue, fontsize=30, label="<<< ingenieria >>>", rankdir="LR"];
      …
      }
      …
      Informar_errores_en_el_boceto->Realizar_bocetos_del_producto;
      Aceptar_posibilidad_de_fabricacion->Realizar_modelado_3D;
      Informar_imposibilidad_de_fabricacion->Analizar_si_existe_otra_solucionn;
}
```

Una vez que se genera el archivo "*.dot"*, es necesario ejecutar desde PHP, la función *shell* con el programa *dot* y sus respectivos parámetros:

```
$codigo = "dot -Tpng graphviz/".$processUUID.".dot -o graphviz/".$processUUID.".png";
$res = shell_exec($codigo);
```

De esta forma es posible llegar a realizar una aproximación al diseño del proceso de negocios original como se encuentra en la Figura 6.14.

Como puede verse en la figura, las actividades se encuentran en posiciones que no son convenientes para realizar un diseño de proceso de negocio. Esto sucede por la forma en que trabaja *dot*.

dot dibuja los grafos en 4 fases principales, y una característica fundamental es que el grafo es acíclico. Las fases que realiza *dot* son las siguientes:

- El primer paso es eliminar cualquier ciclo que se encuentre en el grafo, para evitar el *deadlock* a medida que se procesa el diseño.
- El siguiente paso es asignar rankings o niveles a cada uno de los nodos, que es donde se situaran cada uno de los nodos a través de la coordenada Y del gráfico.
- Luego ordena los nodos del mismo nivel para evitar que se crucen.
- Y por último, se posicionan los nodos a través de la coordenada X del gráfico para mantener las aristas lo más cortas posibles.

Este tipo de comportamiento es común en la mayoría de los programas que dibujan grafos jerárquicos, sin embargo, un proceso de negocio, sin bien puede ser visto como un grafo, no lo es, por lo tanto es necesario disponer de otra herramienta, que tenga la lógica de proceso de negocio, para realizar este gráfico. Lamentablemente no he encontrado una herramienta de este tipo que sea *open souce* o libre, pero no dudo en que se pueda utilizar *dot* para realizar una herramienta para el diseño de procesos de negocios.

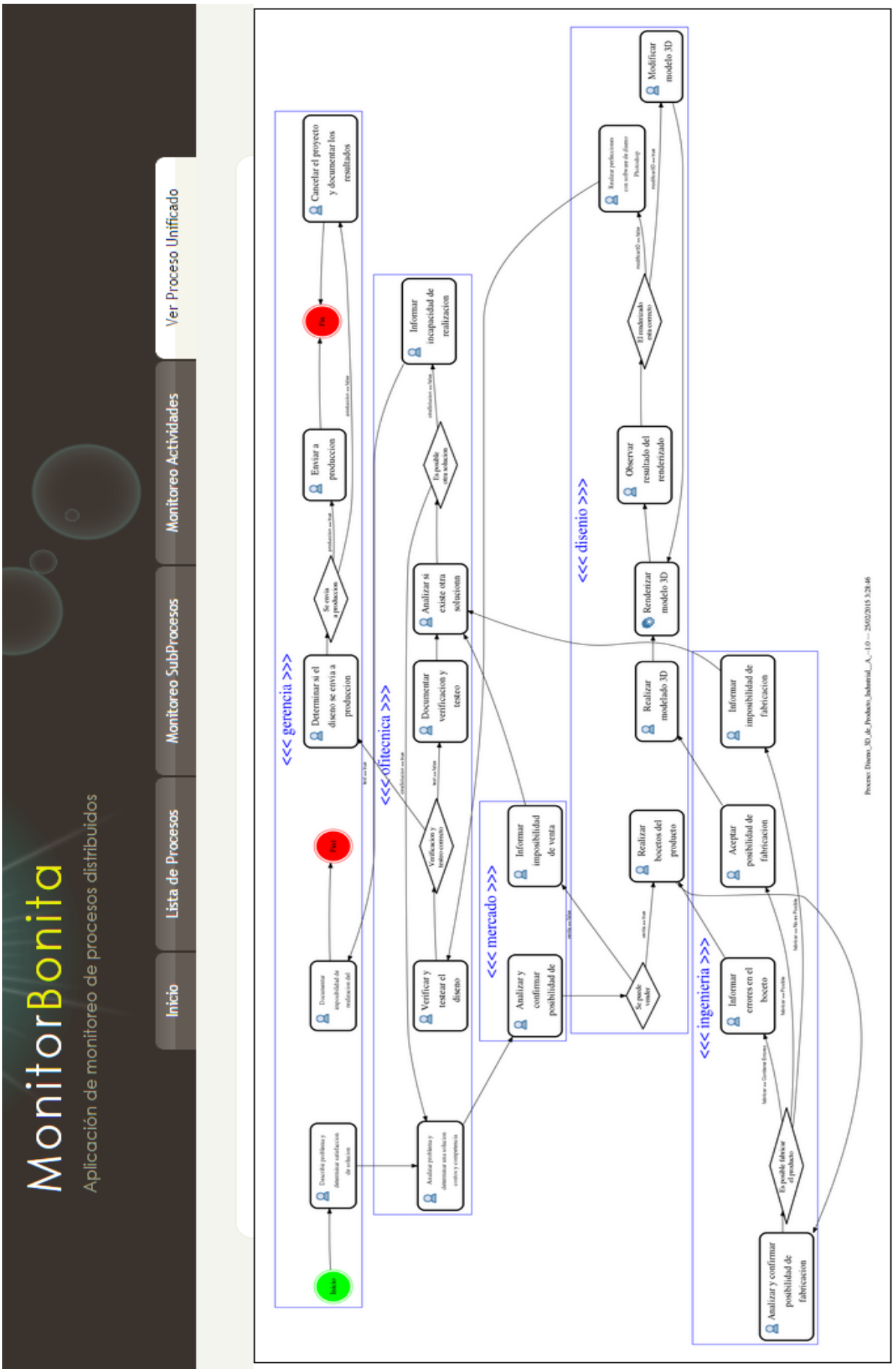


Figura 6.14. Visualización con GraphViz del proceso de negocios original a partir de la unión de los subproceso

Capítulo 7 - **Conclusiones y trabajos futuros**

7.1. Conclusiones

Hoy en día aquellas organizaciones que utilizan sistemas basados en BPMS como tecnología para generar los productos y servicios que ofrecen, tienen sistemas embebidos con arquitecturas centralizadas donde deben estar sostenidas por una infraestructura muy costosa que a medida que pasa el tiempo se va dañando, teniendo que reparar los componentes, o va quedando obsoleta, en cuyo caso se debe reemplazar por una nueva seguramente más costosa que la anterior.

A partir de la descomposición de procesos, o la ejecución de procesos de negocios distribuidos, y más aún con el uso de *Cloud Computing* podemos encontrarnos con varias ventajas respecto a esta situación:

- Se aumenta la disponibilidad de las aplicaciones que interactúan a través del proceso
- Mayor capacidad en la integración de elementos dispersos en una arquitectura a través de servidores BPM
- Mayor posibilidad en la explotación de recursos de gran riqueza en hardware y poder de cómputo a través de los sistemas *cloud* actuales.
- Ubicuidad de los servidores, permitiendo la ejecución de los procesos en áreas más cercanas al lugar de los responsables de las tareas.
- Infraestructuras *cloud* adaptables a la necesidad del negocio, y pago sólo por el uso que se le da a las mismas, evitando grandes costos en infraestructuras propias.

Sin embargo, y a pesar de estos grandes beneficios anteriormente mencionados, también existen varios obstáculos al adoptar la combinación de tecnologías entre cloud y BPM y que fueron encontrados y sorteados durante el desarrollo del presente trabajo:

1. **Configuración de servidores**: es necesario configurar correctamente los servidores que intervienen en el proceso de negocios distribuido para ponerlo operativo. Esto requiere que se disponga de especialistas en administración de servidores y, a diferencia con un único sistema embebido, también se requieren administradores de redes, ya que la comunicación entre los subprocesos demandan un mayor mantenimiento para tener el proceso de negocio funcional, y evitar demoras y pérdida del negocio a causa de configuración.

2. **Distribución transparente al flujo de ejecución**: la ejecución propia de los subprocesos de negocio, también se debe asegurar que se mantenga el flujo del proceso operativo. En nuestro caso, con el uso de los conectores de Bonita, es fundamental que

éstos sean robustos y veloces, evitando demoras o errores en la instanciación de los subprocesos en otros servidores.

3. **Monitoreo distribuido**: al no poder realizar el monitoreo debido a que los procesos no se encuentran en un solo BPMS, fue necesario crear una aplicación de monitoreo que reconstruya el proceso original y permita verlo en forma integrada, no pudiendo en muchos casos aprovechar el monitoreo individual de cada nodo distribuido por separado. Inclusive, al ser mayormente centralizada, esta aplicación puede generar un cuello de botella en caso de que el proceso sea muy extenso y complejo.

4. **Información débil de los procesos en ejecución**: un inconveniente que surgió al utilizar la API de Bonita, es la falta de información que se obtiene de sus actividades o tareas cuando se realiza consultas sobre los procesos desplegados. Se puede decir que una vez desplegados los procesos, los mismos pierden algunas de las propiedades de la notación BPMN, confundiéndose unas tareas con otras.

La utilización de GrapViz como aplicación para realizar el diseño del proceso de negocio unificado, no es una aplicación para realizar diseños de procesos de negocio, sino de grafos. Debido a que la posición en la que se encuentran los nodos de un grafo no es de vital importancia, como si lo son las tareas dentro de un proceso de negocio para su mejor comprensión, este inconveniente ocasiona que cada vez que se envíe el archivo *.dot* a procesar se generen diferentes posiciones para todos sus nodos, confundiendo en muchas ocasiones la lectura del proceso de negocio unificado.

En este trabajo se presentó una propuesta de solución para ejecución de procesos distribuidos que incluye las ventajas presentadas y que ha logrado aplicar dicha propuesta utilizando herramientas concretas e identificando los inconvenientes que esto conlleva si desea aplicarse con otras tecnologías.

7.2. Trabajos Futuros

Como vimos a lo largo del trabajo, es indispensable que se realice correctamente la descomposición del proceso original en varios subprocesos, y que se configuren los conectores con los datos de conexión de cada servidor. Una mejora a esto sería disponer de una herramienta automática, que se encuentre de forma nativa dentro del BPMS, donde una vez configuradas sus actividades, ya sea a través de BPMN o con el uso de listas de distribución (como se ha visto en el Capítulo 3), se encargue de descomponer al proceso y desplegar cada subproceso en el servidor correspondiente de la arquitectura. Esta ventaja, tanto para los diseñadores de procesos como para los analistas, haría ver a los procesos de negocio siempre como una unidad en lugar de verlo como una dispersión de subprocesos, a la misma vez sería totalmente transparente para las personas involucradas.

La aplicación de monitoreo presentada guarda la información solo de los subprocesos que realizan la instanciación remota de procesos. De esta manera es posible monitorear solo aquellas instancias que se ejecutan en el BPMS propio o en el servidor en el que se despliega la nueva instancia. Esto ocasiona que se deba ir por cada aplicación de monitoreo para analizar las instancias de los demás subprocesos.

Para evitar ir a las aplicaciones de monitoreo en cada uno de los servidores, sería recomendable disponer de una aplicación de monitoreo centralizada (o en varios servidores replicados), la cual funcionaría a través de recolectores de eventos. Es decir, en vez de encarar un proceso de recolección al momento de mostrar una instancia, que las instancias distribuidas sean las que informen en forma asincrónica a este servidor centralizado (o servidores) su información individual útil para el monitoreo. Esto hace que al momento de monitorear una instancia no haya que encarar un barrido costoso, sino que esa información ya esté disponible y solo haya que procesarla en forma eficiente.

Referencias

[1] Weske Mathias, “Business Process Management: Concepts, Languages, Architectures”. Springer, Pag 3-67. ISBN 978-3-540-73521-2. 2008.

[2] Juric Matjaz B., Loganathan Ramesh, Poornachandra Sarang, Frank Jennings “SOA Approach to Integration XML, Web services, ESB, and BPEL in real-world SOA projects”. Packt Publishing. ISBN 978-1-904811-17-6. 2007.

[3] Bazán P. “Un modelo de integrabilidad con SOA y BPM”. Tesis de Maestría en Redes de Datos. Facultad de Informática. Universidad Nacional de La Plata. Abril 2010.

[4] Martin Owen and Jog Raj. “BPMN and Business Process Management Introduction to the New Business Process Modeling Standard”. Popkin Software. 2003.

[5] Xun Xu. “From cloud computing to cloud manufacturing”. Department of Mechanical Engineering, University of Auckland. August 2011.

[6] P. Mell and T. Grance, “The NIST Definition of Cloud Computing,” National Institute of Standards and Technology. September 2011.

[7] Barrie Sosinsky. "Cloud Computing Bible". Wiley Publishing Inc. ISBN: 978-0-470-90356-8. 2011.

[8] SANDETEL. "Cloud Computing Aplicado a los sectores de la Agroindustria, Eficiencia Energética, Industrias Culturales y Turismo". Noviembre 2012

[9] C. Baun, M. Kunze, J. Nimis and S. Tai. “Cloud Computing: Web-Based Dunamic IT Services”. Springer. ISBN 978-3-642-20916-1. 2011.

[10] Karabogosian L., Bazán P., Martinez Garro J. “Ejecución y monitoreo de procesos de negocios distribuidos entre diferentes motores de Bonita OS”. WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación. 2014

[11] Evert F. Duipmans. "Business Process Management in the cloud with data and activity distribution". Faculty of electrical engineering, mathematics and computer science software engineering. University of Twente. EWI/SE - 2012-002. November 2012.

[12] Rafael Accorsi. "Business Process as a Service: Chances for Remote Auditing". Department of Telematics, University of Freiburg, Germany. 2011. 35th IEEE Annual Computer Software and Applications Conference Workshops

[13] J. Martinez Garro, P. Bazán. "Monitoreo de procesos distribuidos en el cloud: Una propuesta arquitectónica". Jornadas Chilenas de Computación 2013.

[14] Han YB, Sun JY, Wang GL. “A cloud-based BPM architecture with user-end distribution of non-compute-intensive activities and sensitive data”. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 25(6): 1157–1167 Nov. 2010. DOI 10.1007/s11390-010-1092-5

[15] Jiri Kolar. “Business Activity Monitoring”. Masaryk University Faculty of Informatics. Brno, spring 2009.

[16] Nathaniel Palmer. "First Impressions: Bonita Open Solution". http://www.bpm.com/first-impressions-bonita-open-solution.html. Abril 2013

[17] Bonitasoft - Open Source Workflow & BPM software. http://es.bonitasoft.com

[18] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, (Piscataway, NJ, USA), pp. 50:1–50:12, IEEE Press, 2008.

[19] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[20] Oracle Technology Network. Java. http://www.oracle.com/technetwork/java/index.html

[21] BonitaSoft. www.bonitasoft.com

[22] Bizagi. www.bizagi.com

[23] Documentación Bonita Open Solution. http://documentation.bonitasoft.com/5x/bos-59

[24] Java Developmet Kit. http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[25] Oracle Documentation. http://docs.oracle.com/cd/E19159-01/820-3189/gefou/index.html

[26] Documentación de la API REST de Bonita. http://documentation.bonitasoft.com/javadoc/rest/5.9/API/index.html

[27] Gartner Inc. http://www.gartner.com

[28] Bonita Open Solution recognized "Best" and "Cool". http://www.bonitasoft.com/newsletter/072011.html

[29] Oracle Business Process Management Suite. http://www.oracle.com/us/technologies/bpm/suite/overview/index.html

[30] Intalio BPM. http://www.intalio.com/products/pricing/

[31] Object Management Group, Inc. (OMG). "Business Process Model and Notation (BPMN) Version 2.0". January 2011

[32] SimpleXML. http://php.net/manual/es/intro.simplexml.php

[33] PEAR. http://pear.php.net/

[34] HTTP_Request2 - http://pear.php.net/package/HTTP_Request2

[35] Flores Guaño M.D., Licuy Mamallacta F.D.. "Comparativa de motores de render con imágenes fotorrealistas tridimensionales aplicada a la historia arquitectónica de la unidad educativa San Felipe Neri". Tesis de Grado Licenciatura en Diseño Gráfico. Ecuador 2011.

[36] Historia del diseño asistido por computadora. http://es.wikipedia.org/wiki/Historia_del_dise%C3%B1o_asistido_por_computadora

[37] Autodesk Revit - http://www.autodesk.es/products/revit-family/overview

[38] Autodesk 3D Studio Max - http://www.autodesk.es/products/3ds-max/overview

[39] Autodesk Inventor - http://www.autodesk.es/products/inventor/overview

[40] SketchUp - http://www.sketchup.com/es

[41] Rhinoceros 3D - https://www.rhino3d.com/

[42] Blender - http://www.blender.org/

[43] Rhinoceros 3D Wikipedia - http://es.wikipedia.org/wiki/Rhinoceros_3D

[44] Brian Budge. Autodesk 360 Rendering - Scalable and Robust Rendering in the Cloud. http://www.highperformancegraphics.org/previous/www_2012/media/Hot3D/HPG2012_Hot3D_Autodesk.pdf

[45] About Autodesk 360 Rendering - https://bim.wikispaces.com/file/view/Autodesk360Rendering_CloudRendering.pdf

[46] Oracle VM VirtualBox - https://www.virtualbox.org/

[47] Start Instance. Bonita Open Solution - http://documentation.bonitasoft.com/5x/bos-57/connectivity/connectors-bonita-open-solution/bonita/start-instance

[48] Bonita Open Solution Connectors Guide - Pag. 35 - http://www.bonitasoft.com/system/files/download/bos-5.6-connectors-guide.pdf

[49] MySQL Documentation - http://dev.mysql.com/doc/refman/5.7/en/

[50] jQuery, About jQuery - http://learn.jquery.com/about-jquery/

[51] jTable.org, A JQuery plugin to create AJAX based CRUD tables. - http://www.jtable.org/

[52] NuSOAP, SOAP Toolkit for PHP - http://nusoap.sourceforge.net/

[53] Graphviz - Graph Visualization Software - http://www.graphviz.org/