REVISTA ODIGOS

Período junio - septiembre 2020
Vol. 1, Núm. 2
rodigos@uisarel.edu.ec
https://doi.org/10.35290/ro.v1n2.2020.322 • e-ISSN: 2697-3405

Componentes de alto rendimiento para la plataforma Bonita BPM

Fecha de recepción: 15/03/2020 • Fecha de aceptación: 14/03/2020 • Fecha de publicación:10/06/2020

Darwin Stalin Ramírez Supe
Universidad Técnica de Ambato
dramirez8774@uta.edu.ec
https://orcid.org/0000-0003-0568-6489

RESUMEN

Existe la necesidad de incorporar componentes personalizados en plataformas empresariales cuyas ediciones al público en general son libres y limitados de características que solo cuentan las ediciones premium o pagadas. Esto se ha vuelto un tema de incomodo por parte de los desarrolladores de software de empresas públicas y privadas porque al desarrollar sus aplicaciones con estas soluciones de TI se ven restringidos al momento de extender funcionalidades de acuerdo al modelo de negocio. Por consecuente, con la crisis económica a nivel mundial, las organizaciones optan por manejar versiones libres para evitar elevados costos de licencias, pero esto obliga a los programadores a tener un desarrollo de software de una manera no convencional para extender las funcionalidades.

La metodología CBSE proporciona un modelo de desarrollo especializado que permite a los programadores desarrollar e integrar componentes personalizados al software privativo. Por esta razón, en la presente investigación se presenta el proyecto SEGIC de la Universidad Técnica de Ambato, adoptado para integrar nuevos componentes en Bonita BPM, una plataforma especializada en workflow para la gestión de procesos en donde se integró elementos de alto rendimiento para la carga masiva de archivos.

PALABRAS CLAVE: CBSE, componentes de software reutilizables, arquitectura de software.

ABSTRACT

There is a need to incorporate custom components into business platforms whose editions to the general public are free and limited to features that only premium or paid editions have. This has become an issue of inconvenience for software developers of public and private companies, because when developing their applications with these IT solutions they are restricted when it comes to extending functionalities according to the business model. Consequently, with the global economic crisis, organizations choose to manage free versions to avoid high cost of licenses, but these forces programmers to have software development in an unconventional way to extend functionality.

The CBSE methodology provides a specialized development model that enables programmers to develop and integrate custom components into proprietary software. For this reason, this research presents the SEGIC project from the Technical University of Ambato, adopted to integrate new components into Bonita BPM, a specialized workflow platform for process management where high-performance elements for loading were integrated massive file.

KEYWORDS: CBSE, reusable software components, software architecture.

Introducción

Actualmente las empresas operan en un entorno global que cambia rápidamente y deben responder a las nuevas oportunidades y tendencias de los mercados, condiciones económicas cambiantes y la aparición de productos y servicios competitivos. Por esta razón, el software se ha convertido en parte esencial de casi todas las operaciones de negocio, por lo que es fundamental que se desarrolle rápidamente para aprovechar tiempos de respuesta y agilidad de entrega de productos precautelando siempre la eficiencia y eficacia de los procesos de negocio (Sommerville, 2004).

La Universidad Técnica de Ambato (UTA) puso en marcha un proyecto para la automatización del proceso de recolección de datos y evidencias para la evaluación de carreras establecido por el concejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior (CEEACES) (Flores, Lavín, Alvarez, & Calle, 2014). El mencionado proyecto se ha desarrollado mediante la aplicación de una herramienta especializada en Gestión de procesos de negocio (BPM, por sus siglas en inglés) denominado Bonita BPM Studio, la cual permite el modelado de todo el proceso de recolección de evidencias necesarias para la acreditación de cada una de las carreras de la Universidad y la generación de una interfaz basada en Web.

Así pues, la plataforma Bonita BPM cuenta con varias ediciones listadas a continuación community, *TeamWork*, *Efficiency*, y *Performance* (P, 2011), siendo la versión *community* elegida por los directores del proyecto. Por consecuencia, esta versión tiene características muy reducidas en comparación con las ediciones de pago, lo que ha generado que el modelo Workflow sea funcional pero no correspondiente al estándar BPMN 2.0 (Jonk, Voeten, Geilen, Basten , & Schiffelers, 2020) al momento de la solicitud de archivos a los usuarios.

Esta versión carece de componentes que no permiten a los programadores desarrollar una interfaz web que sea capaz de solicitar al usuario múltiples archivos y generar una vista para los mismos. A su vez, siendo una plataforma de edición muy restrictiva para programar, estos componentes obligan a los programadores a producir un diseño web con una sobrecarga de trabajo para el usuario por lo cual se ha puesto en marcha esta investigación.

Como resultado de este caso de estudio se desarrollaron componentes de alto rendimiento basados en la metodología CBSE para la gestión de archivos que maneja la plataforma Bonita BPM, pudiendo mejorar notablemente el *workflow* y adaptándose mejor al estándar de modelado de procesos BPMN 2.0.

Metodología

Bonita BPM es una alternativa de código abierto a BPM's comerciales existentes, consta de dos partes diferenciadas: Bonita BPM Studio y Bonita BPM Platform. La primera es un ambiente gráfico que hace posible el modelado de procesos, para ello dispone de un entorno de desarrollo con una barra de herramientas con elementos BPMN para dibujar el flujo del diagrama de

procesos definiendo las etapas, transiciones, puntos de decisión y otros elementos del proceso (Ramakrishnan & Kaur, 2020).

Por su parte, el desarrollo de software basado en componentes es el proceso para definir, implementar e integrar elementos en sistemas independientes débilmente acoplados. Esto se ha convertido en una importante aproximación de desarrollo del software debido a que los sistemas (software) son cada vez más grandes y complejos, y los clientes demandan soluciones tecnológicas más confiables que sean desarrollados de manera rápida y eficaz. Una de las soluciones para tratar esta complejidad y entregar sistemas más precisos es la reutilización de componentes.

Los fundamentos de la ingeniería del software basada en componentes son:

1. **Componentes independientes**. Son completamente especificados por sus interfaces. Debería haber una clara separación entre la interfaz de los componentes y su implementación para que la ejecución de un elemento pueda reemplazar por otro sin cambiar el sistema.

2. **Estándares**. Facilitan la integración de los componentes. Estos se incluyen en un modelo de componentes y definen en el nivel más bajo como las interfaces deberían especificarse y como se comunican los componentes.

3. **Middleware**. Proporciona soportes independientes y distribuidos que trabajan juntos. Se necesita un soporte middleware que maneje las comunicaciones de los componentes.

4. **Proceso de desarrollo**. Se adapta a la ingeniería del software basada en componentes. Si se intenta añadir una aproximación basada en componentes a un proceso de desarrollo que está adaptado a la producción de software original, se puede observar que las suposiciones inherentes al proceso limitan el potencial del CBSE (Sommerville, 2004).

En general, el desarrollo de software basado en componentes puede verse como una extensión natural de la programación orientada a objetos dentro del ámbito de los sistemas abiertos y distribuidos. El paradigma se basa en el uso de los componentes de software como entidades básicas del modelo, entendiendo por componente una unidad de composición de aplicaciones software que posee un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes, de forma independiente en tiempo y espacio (Szyperski, 1998).

En el proceso CBSE, la reutilización con éxito de componentes requiere un proceso de desarrollo adaptado a esta metodología contando con una serie pasos exigentes a cumplir, en la siguiente *Figura 1* se muestran las principales sub actividades dentro del marco de trabajo relacionado.

Figura 1. *Proceso CBSE*
Fuente: elaboración propia

1. Desarrollo de Componentes para la Reutilización

La visión a largo plazo de CBSE es que habrá proveedores de componentes basados en el desarrollo y venta de componentes reutilizables, los problemas de confianza implican que hasta ahora no se haya desarrollado un mercado abierto de estos, y la mayoría de los existentes que son reutilizados han sido desarrollados dentro de las organizaciones en forma personalizada.

Los factores que debe cumplir un componente para que sea más reutilizable son:

1. Eliminar los métodos específicos de la aplicación.
2. Cambiar los nombres para hacerlos más generales.
3. Añadir métodos para proporcionar una cobertura funcional más completa.
4. Hacer que el manejo de excepciones sea consistente para todos los métodos.
5. Añadir una interfaz de configuración para permitir que el componente se adapte a diferentes situaciones de uso.
6. Integrar los componentes requeridos para incrementar la independencia.

Existe un desequilibrio inevitable entre reusabilidad y la usabilidad de un componente. Hacer que este sea reutilizable implica proporcionar una serie de interfaces genéricas con operaciones que

abarcan todas las formas en las cuales podría ser utilizado. Hacer que el componente sea usable significa proporcionar una interfaz mínima sencilla que sea fácil de comprender. La reusabilidad añade complejidad y, por eso, reduce la comprensibilidad del componente, por lo tanto, es más fácil decidir cuándo y cómo reutilizar el mismo.

De igual forma el/los componentes de software son las partes modulares, distribuibles e intercambiables de un sistema, que encapsula implementación y presenta un conjunto de interfaces. Un componente está especificado típicamente por uno o más clasificadores, por ejemplo (clases de implementación) que residen en él, y puede ser implementado por uno o más elementos software, por ejemplo, ver *Figura 2* (ficheros binarios, ejecutables o de script) (P, Barrera, & Serrano, 2002).

Un componente debe cumplir los criterios siguientes:

- Puede ser usado por otros elementos software.
- Puede ser usado por clientes sin intervención del desarrollador del componente.
- Incluye una especificación de todas sus dependencias.
- Incluye una especificación de la funcionalidad que él ofrece.
- Es usable sólo en la base de sus especificaciones.
- Se puede componer con otros componentes.
- Se puede integrar en un sistema de forma rápida y sin conflictos.

Figura 2. *Requisitos de un Componente*
Fuente: elaboración propia

Asimismo, un modelo de componente (*Figura 3*) debe estar definido y sujetado a ciertos estándares determinados por la metodología sobre documentación y despliegue, dichos estándares son utilizados por los desarrolladores de componentes para asegurar que estos puedan inter-operar con el sistema macro como uno solo, además, se han propuesto muchos modelos, pero los más usados son el modelo de componentes CORBA de OMG, Enterprise Java Beans de Sun, y COM+ de Microsoft (Sommerville, 2004).

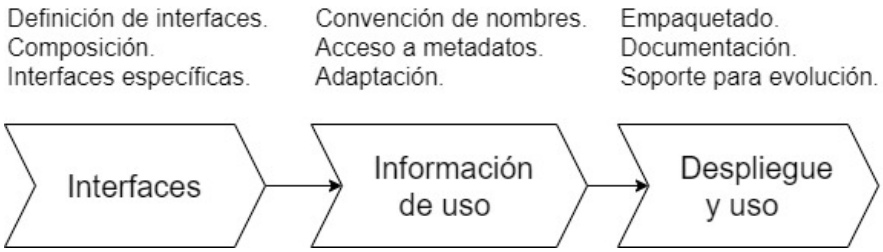


Figura 3. *Modelo de componentes*

Fuente: elaboración propia

Respectivamente, las interfaces de un componente determinan tanto las operaciones que implementa, como las que precisa utilizar de otro componente durante su ejecución. En los modelos habituales cada interfaz va a venir determinada por el conjunto de atributos y métodos públicos que el componente implementa, y por el conjunto de eventos que emite (Bertoa, Troya, & Vallencillo, 2002).

En igual forma, la programación orientada a componentes (POC) nace con el objetivo de construir un mercado global de componentes software, cuyos usuarios son los propios desarrolladores de aplicaciones que necesitan reutilizar componentes ya hechos y probados para construir sus aplicaciones de forma más rápida y robusta. Las entidades básicas de la POC son los componentes, es necesario que estén empaquetados de forma que permitan su distribución y composición con otros, especialmente con aquellos desarrollados por terceras partes.

Se considera a la POC como un paradigma de programación que se centra en el diseño e implementación de componentes, y en particular en los conceptos de encapsulación, polimorfismo, composición tardía y seguridad (de Oliveira Dantas, de Carvalho Junior, & Barbosa, 2020).

En primer lugar, los componentes Commercial-OFF-THE-SHELF (COTS), este término hace referencia al software comercial, son clases de componentes de software generalmente adquiridos en formato binario, sin posibilidad de tener acceso al código fuente y sin información adicional que ayude a los integradores en la selección correcta de los mismos. Esto impide pensar en tareas para la automatización de procesos (Meyers & Oberndorf, 2001).

En segundo lugar, las líneas de productos se refieren a técnicas de ingeniería para crear un portafolio de sistemas de software similares, a partir de un conjunto compartido de activos de software y usando un medio común de producción (Krueger, 2006).

Debe señalarse que la especialización de líneas de productos se puede desarrollar con varios tipos de especialización:

1. Especialización de la plataforma, se desarrollan versiones de la aplicación para diferentes plataformas.

2. Especialización del entorno, se crean versiones de la aplicación para gestionar entornos

operativos y dispositivos periféricos concretos.

3. Especialización de la funcionalidad, se crean versiones de la aplicación para clientes específicos que tienen diferentes requerimientos.

4. Especialización del proceso, el sistema se adapta para tratar con procesos de negocio específicos.

2. Modelo empleado en el desarrollo de software basado en componentes

En la *Figura 4* se muestra un conjunto de pasos para desarrollar un componente desde su inicio hasta la integración con la aplicación recolectando datos desde el análisis de dominio, diseño de dominio y la ingeniería de componentes delimitando así la funcionalidad del componente.

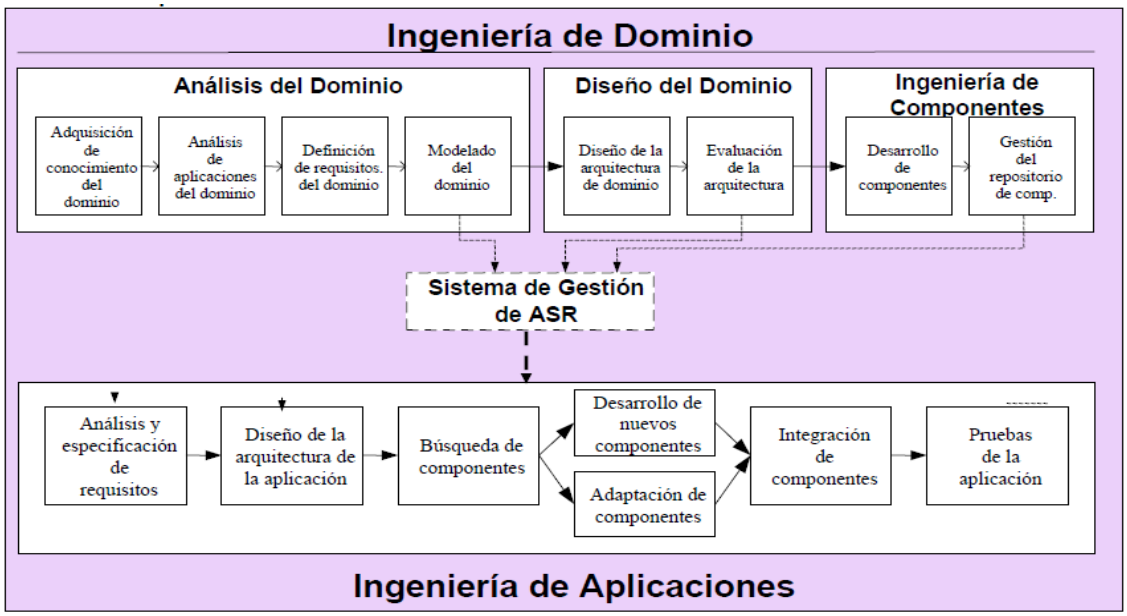


Figura 4. *Modelo TWIN Extendido*

Fuente: elaboración propia

3. El modelo ESPLEP

Para el siguiente modelo usamos la representación de casos de uso para definir los requerimientos en forma general visto desde la perspectiva del desarrollador de componentes, como se puede observar en la *Figura 5*.

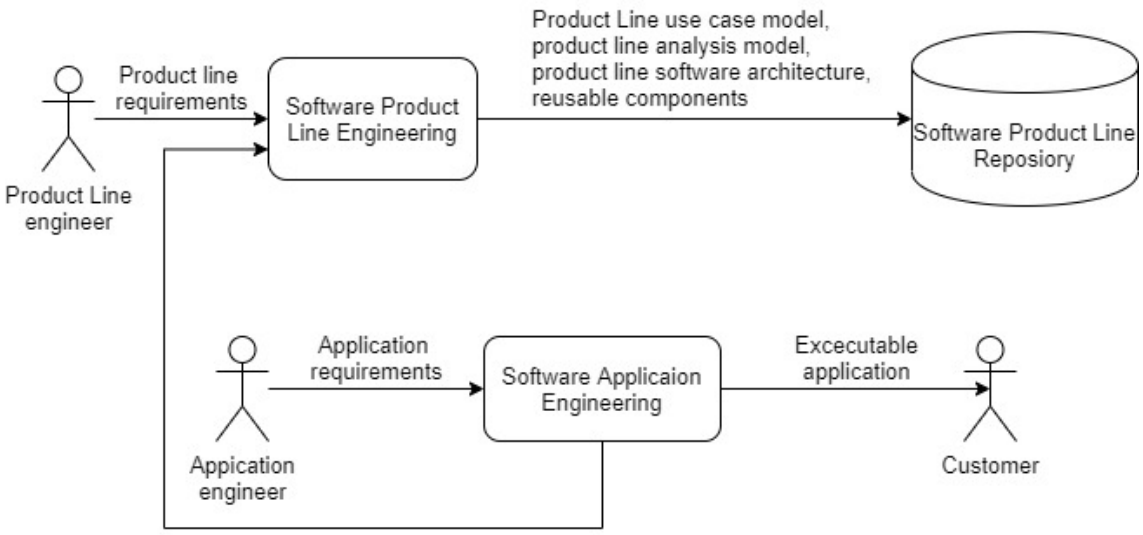


Figura 5. *Modelo ESPLEP*
Fuente: elaboración propia

Finalmente, la reutilización de componentes COTS se aplica a un sistema de software que puede utilizarse sin cambios por su desarrollador. En la actualidad es normal para los grandes sistemas tener definidas interfaces de programación de aplicaciones (APIs) que permitan programar el acceso a las funciones de dichos sistemas.

Para desarrollar sistemas utilizando productos COTS, se tienen que tomar varias elecciones de diseño:

1. Qué productos COTS ofrecen la funcionalidad más adecuada.

2. Cómo se intercambiarán los datos.

3. Qué características de un producto se utilizaran realmente.

4. Integración de componentes en la plataforma Bonita BPM

Para la integración de los componentes se procedió con el estándar que dicta la metodología CBSE, especificando una serie de:

Arquitectura y descripción de los componentes

Los componentes de alto rendimiento que se han propuesto desarrollar e integrarse a la plataforma Bonita BPM son: *MultiFilesUpload* y *ViewFilesUpload*, estos componentes son muy importantes para el proyecto SEGIC que se implementa en la UTA para permitir una mejor gestión de los archivos en el proceso de recolección de evidencias. Para este aspecto se utilizó modelos ontológicos y de colaboraciones en donde se puede apreciar la funcionalidad de cada componente

dentro de un dominio específico.

Modelo Ontológico

Como se puede observar en la *Figura 6* (a) representa el modelo Ontológico del componente *MultiFilesUpload* y la Fig. 6 (b) representa el modelo ontológico del componente *ViewFilesUpload* respectivamente. Para definir los modelos ontológicos se usó una notación extendida de los casos de uso que nos permite apreciar una acción abstracta a partir de la cual se inicia el proceso de refinamiento del componente y la interacción con el usuario.

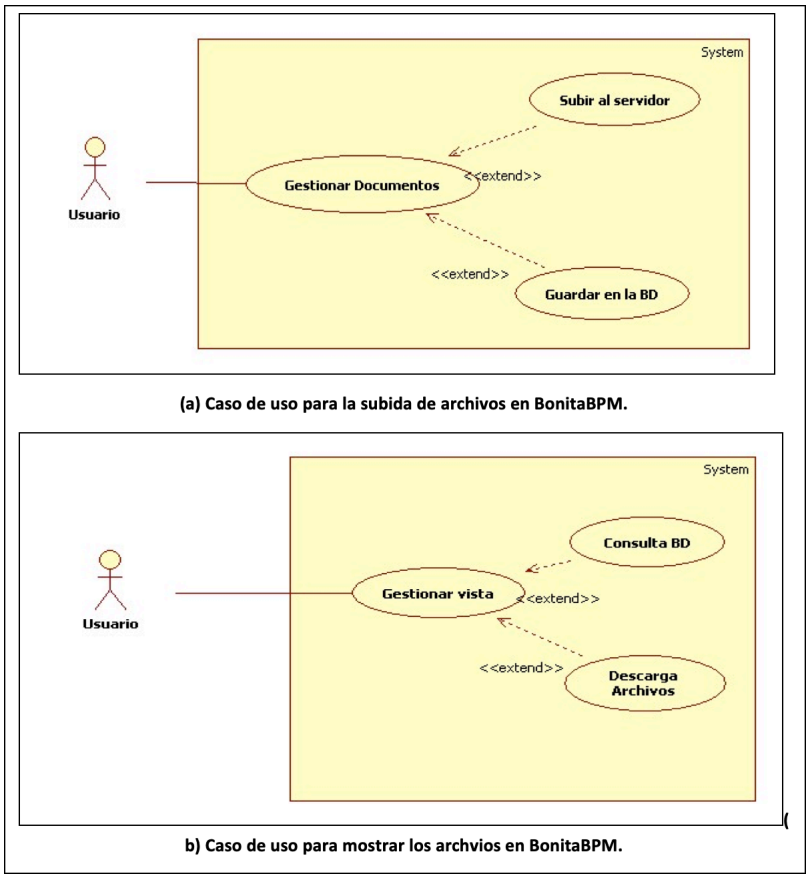


Figura 6. *Modelo Ontológico de los componentes a integrarse en Bonita BPM*

Fuente: elaboración propia

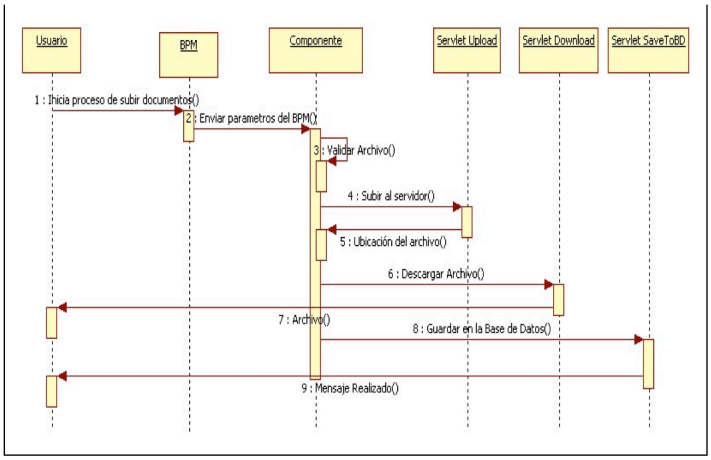
Modelo de Colaboraciones

La dinámica de los componentes se describe a través del modelo de colaboraciones, en la *Figura 7* (a) y (b) se pueden apreciar los diagramas de secuencia de los componentes *MultiFilesUpload* y *ViewFilesUpload* respectivamente, para definir el modelo de colaboraciones se ha utilizado un diagrama de secuencias.

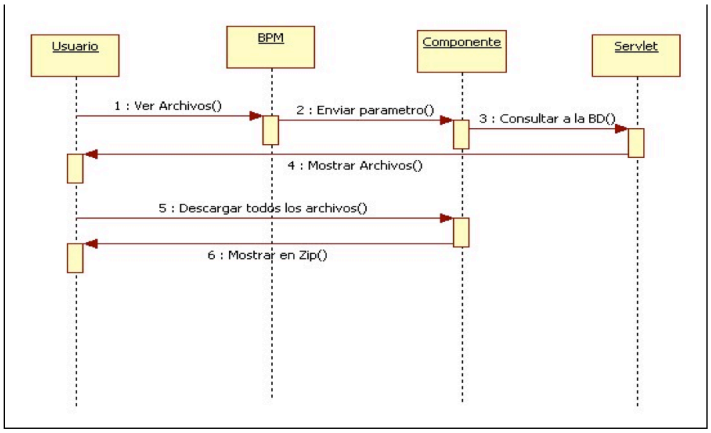
En primer lugar, la *Figura 7* (a) representa el escenario en el que el usuario inicia la petición de

subir las evidencias hacia el servidor desde la plataforma BPM. Éste encapsula los datos para posteriormente enviarlos hacia el componente, este los gestiona mediante las instancias (servlets) y permite una vista al usuario de la evidencia/archivo subido, finalmente se guardarán todos los archivos en la BD de la plataforma Bonita BPM.

Mientras que en la (b) se observa el escenario del componente ViewFilesUpload, en cual el usuario desde la plataforma BPM solicita ver todos los documentos subidos, este envía los parámetros necesarios al componente, realiza la consulta a la BD y muestra todos los documentos almacenados en la plataforma permitiendo descargarlos.



(a) Diagrama de colaboraciones(UML) para la gestión de subida de documentos.



(b) Diagrama de colaboraciones (UML) para la visualización de documentos.

Figura 7. *Modelo de colaboraciones (UML) de los componentes a integrarse en la plataforma Bonita BPM.*
Fuente: elaboración propia

Desarrollo de los componentes

En base a la metodología CBSD se desarrollan componentes independientes a la plataforma, el cual consta con mejoras respecto al funcionamiento de los componentes originales de Bonita BPM Studio.

Para el desarrollo de los componentes se utilizó las siguientes herramientas:

- Lenguaje de programación: Java, JavaScript, Groovy, Html
- Tecnologías: Servlet, Ajax, Jsp
- IDE: Eclipse
- Versión de JDK: 7
- Librerías alternas: jquery-1.11.3.min.js
- Servidor de Aplicaciones: JBoss5.10GA

La interfaz Document Object Model (DOM) consta de elementos propios del lenguaje HTML que se empleó para generar los componentes propuestos.

En la *Figura 8 (*a) se puede observar la interfaz del componente *MultiFilesUpload*, consta de un botón añadir para agregar más componentes de tipo *input files* con su respectiva opción de remover el documento, y cambiar el documento por otro. Y en la Figura 8 (b) se puede notar el componente *ViewFilesUpload*, el cual muestra todos los archivos subidos por un usuario en específico con la opción agregada de descargar todo, el cual genera un archivo de tipo zip que contiene todos los documentos.

Figura 8. *(a) Componente MultiFilesUpload para la carga masiva de documentos*

Fuente: elaboración propia

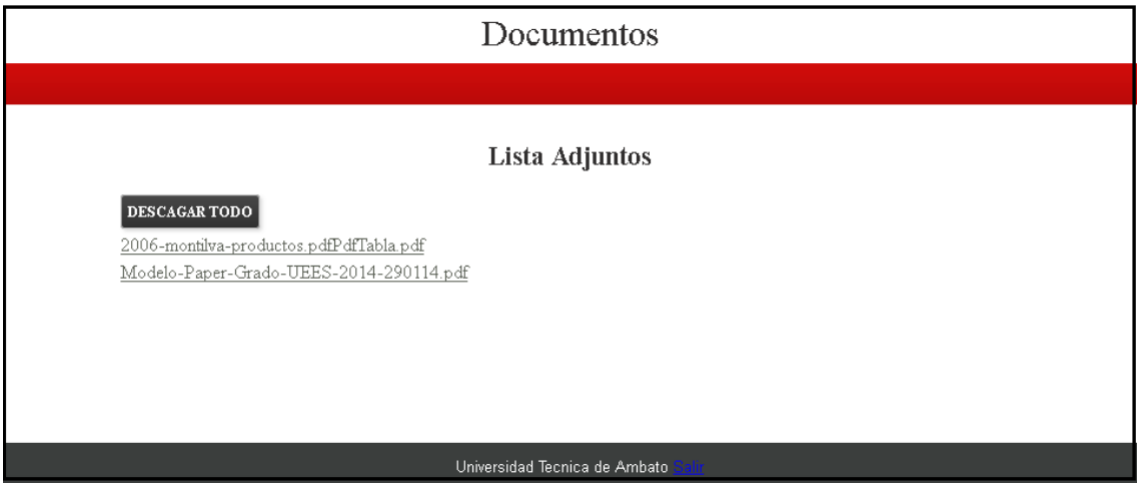


Figura 8. (b) Componente *ViewFilesUpload* para la visualización y descarga masiva de documentos
Fuente: elaboración propia

Validación

Las validaciones se hacen en el Front-End del componente como se puede apreciar en la *Figura 9*, permitiendo el traspaso de los datos limpios hacia la plataforma. Además, asegura una mayor velocidad en el desempeño del servidor teniendo que gestionar solo datos validados en el sistema.

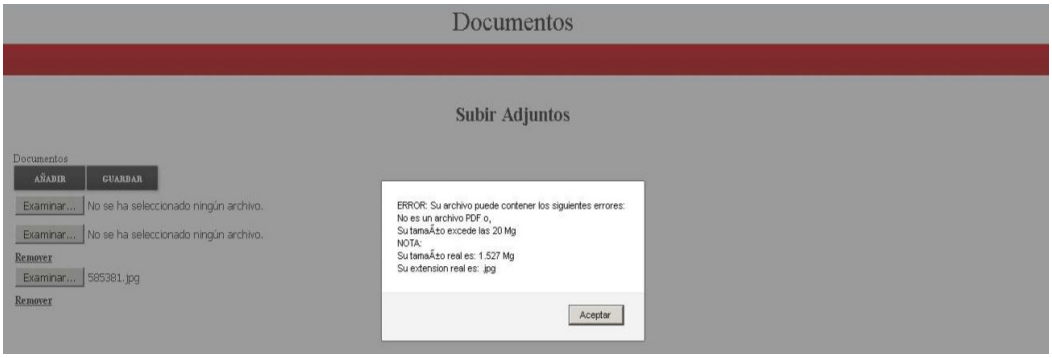


Figura 9. Validaciones *MultiFilesUpload* para la plataforma Bonita BPM
Fuente: elaboración propia

Integración

La integración con Bonita Studio se ha realizado mediante un componente propio de la plataforma denominado Widget HTML, en dicho componente se programó mediante lenguaje GROOVY una interfaz HTML permitiendo inyectar parámetros por URL necesarios para gestionar el almacenamiento en la BD de Bonita BPM como lo muestra la *Figura 10*.

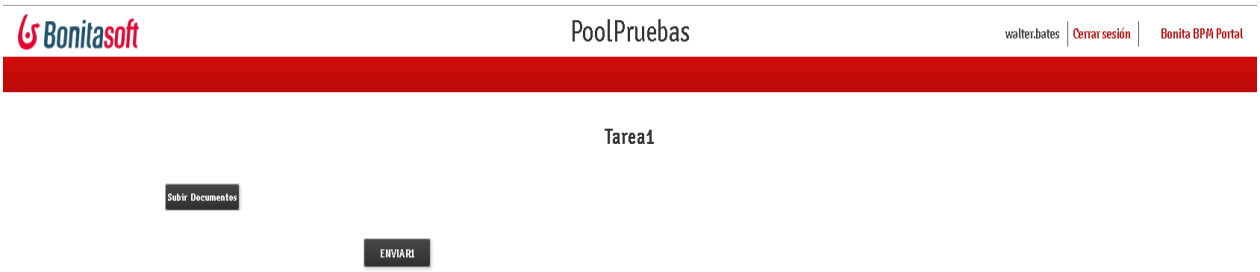


Figura 10. *Integración de los componentes a la plataforma BPM*
Fuente: elaboración propia

Resultados

Para probar el rendimiento del servidor durante la carga masiva de archivos se ha tomado una muestra de 35 documentos de diferentes tamaños. Los tiempos de respuesta se analizaron tanto con los componentes originales de bonita BPM, así como los componentes desarrollados para esta investigación.

Ante estos datos se realizó el siguiente análisis estadístico para comparar tiempos de respuesta.

Hipótesis:

en donde:

μ= media (bonita BPM)

$\mu 0$ = media (componentes desarrollados)

z = estadígrafo de contraste

∞ = nivel de significancia

μ = 0.520

υ = 0.137

$\mu 0$ = 0.10

H0 $\mu > \mu 0$

H1 $\mu \leq \mu 0$

$$z = \frac{x - \mu 0}{\frac{\upsilon}{\sqrt{n}}}$$

$z \geq z\infty$

6.6 ≥ 1.96

Mediante un contraste de hipótesis se ha demostrado que la hipótesis inicial es válida, aceptando H0 y rechazando H1

Al final se obtuvo un análisis entre todos los tiempos y se demuestra estadísticamente que los componentes desarrollados en esta investigación se obtienen mejores tiempos y resultados en el servidor de Bonita BPM.

Conclusiones

Se ha logrado desarrollar e integrar componentes que realicen nuevas funcionalidades en la plataforma Bonita BPM en base a la metodología CBSE. Además, gracias a que los componentes fueron desarrollados para el presente proyecto, se tiene un control total del código fuente y esto permite la constante evolución de los mismos según los requerimientos lo ameriten. También, asegura la confiabilidad y fiabilidad del origen de los componentes sin tomar riesgos de código malicioso que puedan afectar la seguridad de los datos.

Por otro lado, se consiguió que la plataforma Bonita BPM versión *community* sea escalable con respecto a sus funcionalidades, permitiendo así que los nuevos componentes funcionen de forma cohesiva con la plataforma para cualquier proceso de desarrollo de un *workflow*.

En cambio, el testeo del aplicativo se realizó a nivel de los componentes sin interferir en el desarrollo del proceso general del proyecto SEGIC compartimentado la fase de pruebas del desarrollo de software.

Para finalizar, la metodología CBSE es adaptable a cualquier modelo de desarrollo de software con sus respectivas modificaciones ya que es una metodología de desarrollo ágil. Esto permitió acoplarse perfectamente con el modelo de desarrollo que utiliza el equipo de desarrollo en el proyecto SEGIC de la Universidad Técnica de Ambato y deja una posibilidad abierta para futuros componentes a integrarse.

Referencias

Bertoa, M., Troya, J., & Vallencillo, A. (2002). Aspectos de calidad en el desarrollo de software basado en componentes. En *Calidad en el desarrollo y mantenimiento de software.*

De Oliveira Dantas, A., de Carvalho Junior, F., & Barbosa, L. (2020). A component-based framework for certification of components in a cloud of HPC services. *Science of Computer Programming*.

Flores, A., Lavín, J., Alvarez, E., & Calle, X. (2014). Buscando la excelencia educativa: Gestión de procesos académicos y administrativos en Instituciones Públicas de Educación mediante BPM. *Congreso Ecuatoriano de Tecnologías de la información y comunicaciones.* Cuenca.

Jonk, R., Voeten, J., Geilen, M., Basten , T., & Schiffelers, R. (2020). SMT-based verification of temporal properties for component-based software systems. *Workshop On Discrete Event Systems*.

Krueger, C. (2006). New methods in software product line development. *Software Product Line Conference.*

Líneas de productos, componentes, frameworks y mecanos. (1998). Washington, DC (Feb 1998). 4Zimmerman, P., and Symington, S., The IEEE Standardization Process, HLA, 1998.

Manuel Bertoa, J. T. (2002). Aspectos de calidad en el desarrollo de software basado en componentes.

Meyers, B., & Oberndorf, P. (2001). Managing Software Acquisition: Open Systems and Cots Products. *The SEI Series in Software Engineering.*

Montilva, J. A. (2006). Desarrollo de Software Basado en Lineas de Productos de Software. Mérida-Venezuela.

Montilva, J., & Arapé, N. &. (2003). Desarrollo de Software Basado en Componentes. IV Congreso de Automatización y Control.

P, C. (2011). *BONITA SOFT: Gestor de procesos de negocios BPM. Universidad Nacional de Colombia.* Colombia.

P, F., Barrera, J., & Serrano, J. (2002). *Lineas de productor, componentes, frameworks y mecanos.* Salamanca.

Palomo, S., & Gil, E. (2020). Aproximacion a la ingenieria del software. Centro de Estudios Ramon Areces SA.

Ramakrishnan, R., & Kaur, A. (2020). Performance evaluation of web service response time probability distribution models for business process cycle time simulation. *Journal of Systems and Software*, 161.

Sommerville, I. (2004). *Software Engineering: Seventh Edition.* Obtenido de https://books.google.com.ec/books?id=PqsWaBkFh1wC

Szyperski. (1998). Component software: beyond object-oriented programming.

Vallecillo, I. &. (s.f.). Elaboración de aplicaciones software a partir de componentes COTS.

Zoran Stojanovic, A. D. (1997). Integration of Component-Based Development Concepts and RM-ODP Viewpoints.