UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

PROYECTO SIGEP

Título: Mecanismo de actualización para el Sistema de Gestión Penitenciaria Venezolano.

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

AUTORES: JORGE ERNESTO MARTÍNEZ CABRERA

RAISEL MILIÁN RODRÍGUEZ

TUTOR: ING. MADELÍN HARO PÉREZ

CIUDAD DE LA HABANA, JULIO DE 2007

AÑO 49 DE LA REVOLUCIÓN Julio, 2007

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas para que haga el uso que estime pertinente con este trabajo.

Para que así conste firmamos la presente a los ___ días de mes de junio del 2007

Jorge Ernesto Martínez Cabrera

Raisel Milián Rodríguez

Ing. Madelín Haro Pérez

Dedicatoria

A todas las personas que de una forma u otra están en mi corazón, mi familia, mis padres, mi hermano, mi (titi) esposa, mis amigos y porqué no, a mis enemigos también, por hacer de mi lo que hoy soy, con mis virtudes y defectos, a todos les dedico este trabajo.

Jorge Ernesto Martínez Cabrera

A mis padres, por el amor y el cariño que me han dado, por ser mis más fieles amigos de toda la vida, por educarme y ser mis mejores maestros, por apoyarme y confiar en mí siempre, por ser mi inspiración. A mis hermanos por estar siempre a mi lado, por darme su apoyo y cariño, por ayudarme en todo lo que he necesitado. A mi compañera, amiga, novia y esposa que me ha apoyado y dado aliento en todo momento. Por tener paciencia y esperarme. A mi pequeño nené, por ocupar otro espacio de mi corazón. A mi familia por preocuparse y estar atenta a mi superación. A mis compañeros que de una forma u otra me han ayudado en toda mi carrera y han contribuido a que hoy pueda escribir estas líneas. A la revolución cubana por hacer este sueño realidad y en especial al comandante en jefe Fidel Castro R, por hacer de mí quien soy.

Raisel Milián Rodríguez

Resumen

La Universidad de las Ciencias Informáticas (UCI) se encuentra desarrollando el Sistema de Gestión Penitenciara (SIGEP) para la República Bolivariana de Venezuela. Este sistema será instalado en un gran número de instituciones que están dispersas por la hermana nación. La dirección general de tales instituciones se encuentra sometida a una profunda reorganización por lo que de antemano se sabe que el modelo de negocio de SIGEP cambiará durante el transcurso de su desarrollo. Este cambio provoca que las versiones del sistema que se encuentren en uso deban ser actualizadas con el nuevo modelo de negocio. Debido a la lejanía entre todas las instalaciones donde se instalará el software estas actualizaciones aumentan considerablemente los costos por toda la logística asociada para realizar este proceso, como preparar el personal responsable, transportación, alojamiento, etc. Por tales motivos es que surge la necesidad de automatizar el proceso de actualización. El presente trabajo tiene como objetivo desarrollar un mecanismo de actualización que se ajuste a las necesidades y características de SIGEP.

Actualmente no existe ningún modelo de actualización que abarque todos los escenarios posibles sobre los que estará desplegado el sistema SIGEP, por tal razón haciendo un estudio de los modelos de actualización existentes, partiendo de uno general se ha adaptado para que se ajuste a las características de despliegue de SIGEP. Además de proponerse un mecanismo de actualización, también se ha desarrollado una propuesta de aplicación que lo ponga en práctica. Esta propuesta ha sido desarrollada en el lenguaje Java, apoyándose en algunos marcos de trabajo de la plataforma como Spring.

Palabras Claves

Actualización de Software, Despliegue, Mecanismo de Actualización, Actualizador.

TABLA DE CONTENIDO

INTRODUCCIÓN **1**

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA **3**

1.1 El Despliegue del Software y su Ciclo de Vida 3
1.1.1 Actividades que se llevan a cabo en el proveedor *4*
1.1.2 Actividades que se llevan a cabo en el cliente *5*
1.2 Actualización del Software. 6
1.2.1 Modelo general de actualización. *7*
1.2.2 Tipos de Actualizadores *8*
1.2.3 Comparación entre los tipos de actualizadores existentes *9*
1.2.4 Comparación teniendo en cuenta la entrega y la instalación *11*
1.3 La actualización del software dentro de las plataformas de desarrollo más utilizadas. 16
1.3.1 Plataforma .NET *16*
1.3.2 Opciones para actualizar un software en la plataforma .Net *17*
1.3.3 Plataforma Java *18*
1.3.4 Opciones para actualizar software en la plataforma Java *19*
1.4 ¿Cómo se actualizan los productos creados por la UCI con características de distribución similares a SIGEP? 20
1.5 Conclusiones 21

CAPÍTULO 2: CARACTERÍSTICAS DEL MECANISMO DE ACTUALIZACIÓN. **22**

2.1 Características de despliegue del Sistema de Gestión Penitenciaria. 22
2.2 Problemática de la actualización manual del Sistema de Gestión Penitenciaria 24
2.3 Características que debe tener la actualización de SIGEP. 25
2.4 Modelo de actualización 26
2.5 Mecanismo de actualización de SIGEP 28
2.6 Conclusiones 38

CAPÍTULO 3: CONSTRUCCIÓN DE LA SOLUCIÓN PROPUESTA. **39**

3.1 Construcción de la solución. 39
3.1.2 Proveedor de actualización *41*
3.1.2.1 Diseño del Proveedor de Actualización 41
3.1.2.2 Descripción del las clases mas importantes del diseño 41
3.1.3 Actualizador *45*
3.1.3.1 Diseño del Actualizador 45
3.1.3.1 Descripción de las clases más importantes del diseño 45
3.1.4 Visualizador de Información *48*
3.1.4.1 Diseño del Visualizador 48
3.1.4.2 Descripción del las clases más importantes del diseño 49
3.2 Construcción del Modelo de actualización para SIGEP 50
3.2.1 Descripción de las colas de mensajería *50*
3.2.2 Funcionamiento del mecanismo de actualización implementado *51*
3.2.3 Actualización del mecanismo *54*
3.2.4 Beneficios de la mensajería *54*

3.3 CONCLUSIONES 57

CONCLUSIONES 58

RECOMENDACIONES 59

BIBLIOGRAFÍA 60

Introducción

El proyecto encargado de la Gestión del Sistema Penitenciario en Venezuela está compuesto por diferentes aplicaciones distribuidas por la geografía venezolana, las cuales crecerán en funcionalidad progresivamente debido a que La Dirección General de Custodia y Rehabilitación del Recluso, institución para la cual se pretende desarrollar el sistema, se encuentra inmersa en un profundo proceso de transformación organizacional; lo cual conlleva que el despliegue de la solución de software planteada por el proyecto se realizará de manera cíclica y por partes, de forma que se puedan agregar un conjunto de funcionalidades a la vez. Esto y el indiscutible hecho de que toda solución debe dar soporte al mantenimiento y de que ningún software esta excepto de errores, crean la necesidad de definir un proceso para la actualización del mismo.

Las tareas que se necesitan para actualizar el software manualmente traerían como consecuencia un mayor esfuerzo por parte de la organización ya que habría que capacitar al personal responsable de la actividad y garantizar cuestiones logísticas como el transporte, alojamiento, etc. Otro factor importante sería el tiempo que demorarían estas tareas en realizarse, debido a la distancia que separa al sistema en explotación de la dirección nacional.

Para garantizar que el proceso de actualización cumpla con las exigencias del despliegue de este sistema informático, el mismo debe ser automatizado a través de un mecanismo que posea la capacidad de disminuir el esfuerzo que empleará la organización en la actualización de su aplicación informática.

Por tanto, el problema a resolver queda formulado a modo de interrogante de la siguiente forma: ¿Cómo automatizar las tareas referentes al proceso de actualización de software en el proyecto SIGEP de manera que cumpla con las exigencias de despliegue del mismo?

El objeto de estudio lo constituyen los procesos de actualización de software y el campo de acción de este trabajo queda enmarcado en la actualización de software del Sistema de Gestión Penitenciaria de Venezuela.

Como hipótesis se parte de la idea que si se automatiza el proceso de actualización de software para el Sistema de Gestión Penitenciaria, se logrará disminuir la influencia negativa que trae para el organismo, la realización manual de este proceso.

Con el propósito de dar solución al problema anteriormente planteado se ha trazado como objetivo principal de esta investigación elaborar un mecanismo de actualización e implementar un sistema

informático (que responda al mecanismo) para la automatización del proceso de actualización en el proyecto SIGEP.

De acuerdo con esta propuesta se derivan los siguientes objetivos específicos:

- Definir o adaptar un modelo para la actualización de software.
- Definir las herramientas a usar para la construcción del mecanismo.
- Concebir las aplicaciones necesarias para construir el mecanismo.

Para cumplir con los objetivos y resolver la problemática que ocupa esta investigación se proponen las siguientes tareas:

- Investigar sobre mecanismos de actualización de software existentes.
- Estudio de un modelo general de actualización de software.
- Estudio de herramientas que brinda la plataforma Java EE, que dan soporte a las actualizaciones de software y a la seguridad de los sistemas.
- Implementar una primera aproximación del mecanismo que cumpla con los objetivos planteados.

El presente trabajo está dividido en tres capítulos que recogen todo lo abordado en el trabajo investigativo.

Capítulo 1: Fundamentación Teórica, se brinda un estado del arte sobre los elementos que forman parte del problema que se presenta, así como aquellos que son importantes a tener en cuenta para dar la solución que se requiere.

Capítulo 2: Características del Mecanismo de Actualización, describe la situación problémica y la propuesta de solución. Además se especifican las características que debe tener el mecanismo, se propone un flujo de actividades y se identifican las entidades que interactúan en dichas actividades.

En el Capítulo 3: Construcción de la Solución Propuesta, aborda los detalles relacionados con la construcción del mecanismo y de cada uno de los elementos que lo conforman, además se fundamenta la elección de herramientas que agilizan la implementación del mismo.

Capítulo 1: Fundamentación Teórica

Introducción

En el presente capítulo se brinda un estado del arte sobre los elementos que forman parte del problema que se presenta, así como aquellos que son importantes a tener en cuenta para dar la solución que se requiere. Se plasma toda la información, conceptos y definiciones necesarias para llevar a cabo una investigación exhaustiva de los modelos de actualización existentes. Se establece además, una comparación entre los diferentes tipos de actualizadores, así como también se identifican las alternativas existentes en las plataformas más usadas para construir actualizadores. Por ultimo, se analizan los métodos utilizados por otros proyectos desarrollados en la UCI con similares características de distribución al Sistema de Gestión Penitenciaria (SIGEP).

1.1 El Despliegue del Software y su Ciclo de Vida

A través de los años el despliegue de software ha sido ampliamente definido como la mera instalación de un sistema de software. Esta vista simplista, irreal e incompleta del despliegue de software no cumple con las necesidades actuales de un proceso de desarrollo en el cual la conectividad ya no es un problema y la cooperación entre productores y clientes es cada vez más notable, lo cual le permite a los productores de software brindar servicios de alto nivel en el despliegue de sus soluciones. En el pasado solo la instalación tradicional del software era soportada, pero ya es común dentro del despliegue de la mayoría de las soluciones de software el soporte para procesos de actualización.

El despliegue del software es en realidad una colección de actividades interrelacionadas que en conjunto forman el ciclo de vida del despliegue de software[1]. Informalmente el termino despliegue de software se refiere a todas las actividades que hacen al sistema de software disponible para su uso. Una definición más formal seria la de Hall [2] en la que el proceso de despliegue contiene la entrega, el ensamblaje y el mantenimiento de un sistema de software en particular.

Las actividades en el ciclo de vida del despliegue de software son realizadas en ambos lugares tanto en el proveedor como el cliente e incluyen la *liberación*, *retiro*, *instalación*, *activación*, *desactivación*, *actualización*, *adaptación* y *desinstalación* de un sistema de software.

A pesar de que se puedan identificar un conjunto de actividades distintivas que típicamente conformarían un proceso de despliegue genérico, no se puede determinar específicamente qué prácticas o procedimientos particulares se aplican a cada actividad. El peso depende de la naturaleza del software que esta siendo liberado y de las características de los requerimientos en los proveedores

y clientes. Por eso, la figura # 1 debería ser interpretada como un proceso razonable que necesita ser adaptado y enriquecido acorde con los requerimientos específicos de despliegue.

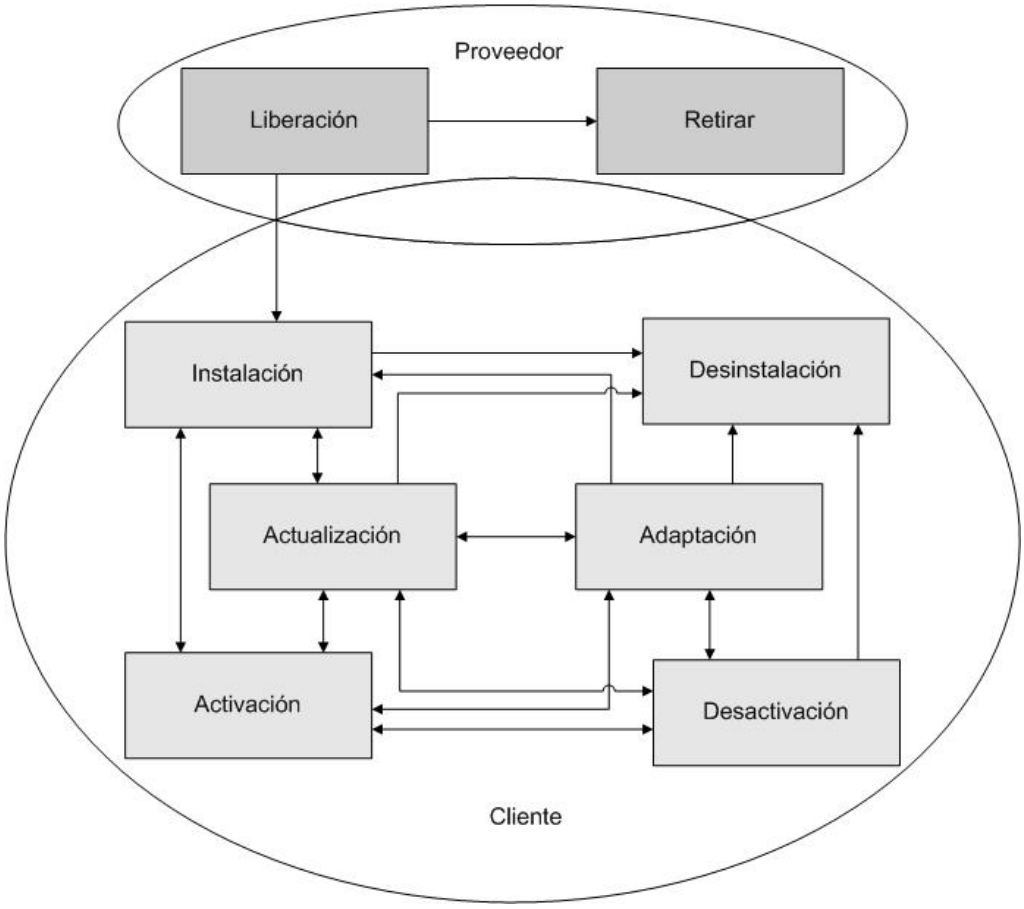

Figura 1 - Actividades del Ciclo de Vida del Despliegue

En los epígrafes a continuación se describen cada una de estas actividades que se realizan tanto en el proveedor como en el cliente.

1.1.1 Actividades que se llevan a cabo en el proveedor

En el lado del proveedor se llevan a cabo dos actividades principales, la *liberación* y el *retiro* de la solución. La *liberación* del producto o solución de software es el puente entre el desarrollo y el despliegue, abarca todas las actividades necesarias para empacar, preparar y anunciar un sistema para desplegarlo en los clientes[2]. El paquete liberado contiene artefactos físicos que incluyen el sistema de software y la descripción de los requerimientos de despliegue del sistema. Cuando se

realicen modificaciones o actualizaciones al producto, el proveedor debe repetir el proceso de *liberación* para crear un paquete liberado actualizado.

Cuando el proveedor de software ya no es capaz de seguir soportando un producto o simplemente no desea seguir dando soporte al mismo, debe realizar el proceso de *retirar* la solución. Las actividades relacionadas con la retirada del producto son distintas a las que ocurren en el cliente llamadas *desinstalación* o *eliminación*. *Retirar* un producto lo hace no disponible para futuros despliegues[2], pero los clientes pueden seguir utilizando el software sin tener conocimiento que ha sido retirado, aunque dicha actividad debe intentar notificar a los usuarios que el soporte del sistema de software ya no esta disponible.

1.1.2 Actividades que se llevan a cabo en el cliente

La *instalación* es la primera actividad de despliegue que se realiza en el cliente y es la encargada de configurar y ensamblar todos los recursos necesarios para el uso de un sistema de software específico[2], utilizando el paquete creado anteriormente en la *liberación* del producto. Para paquetes específicos, la *instalación* interpreta el conocimiento codificado y examina el cliente para determinar cómo configurar adecuadamente el sistema[2]. Una vez la *instalación* ha sido completada, el sistema de software desplegado está listo para su uso y se puede pasar a otras actividades del despliegue.

Después que el software ha sido instalado, la *activación* y *desactivación* permiten al usuario final usar el producto. La *activación* es la actividad responsable de hacer al software desplegado ejecutable y usable[2]. Para una herramienta simple la activación envuelve el establecimiento de algún comando o icono al cual se le puede aplicar un clic para ejecutar la herramienta binaria. En sistemas de tipo cliente-servidor, por ejemplo, múltiples componentes necesitan ser ejecutados en paralelo. La *desactivación*, por el contrario, es la actividad responsable de apagar cualquier componente que se esté ejecutando de un sistema activado[2].

Durante el tiempo que el software ha estado instalado en el cliente, no se comporta como una entidad estática en cuanto a despliegue respecta. Por esta razón se exponen las actividades de *actualización* y *adaptación* que son las responsables de cambiar y mantener la configuración de software existente. Estas pueden ocurrir en cualquier orden y cualquier número de veces.

La *actualización* modifica sistemas de software instalados con anterioridad, es decir despliega una nueva configuración del sistema de software previamente no disponible y se vuelve necesaria cuando un proveedor de software realiza un cambio a la descripción del sistema de software desplegado[2].

Los cambios a la descripción de un sistema de software pueden denotar una nueva versión, una actualización del contenido, o simplemente una actualización de la descripción.

El propósito de la *adaptación* es mantener la consistencia de la configuración actual de un sistema de software previamente desplegado[1]. Esta actividad debe monitorear los cambios en el cliente y responder a estos cambios, con el objetivo de mantener la consistencia del sistema de software desplegado[2]. La *adaptación* se vuelve necesaria cuando se realiza un cambio en el cliente que afecta el sistema desplegado, por ejemplo, cuando un fichero requerido por el sistema es borrado o corrupto, mediante la adaptación se determina el fichero afectado y se repone.

Una vez que el sistema de software ya no es requerido en el cliente, las actividades de *desinstalación* o *eliminación* son ejecutadas[2]. La *desinstalación* debe deshacer todos los cambios provocados en el cliente, por las actividades de despliegue ejecutadas con anterioridad sobre el sistema de software[1]. Esta actividad debe prestar atención especial a recursos compartidos como ficheros de datos o librerías con el objetivo de prevenir referencias peligrosas al recurso requerido. Como resultado, la *desinstalación* debe examinar el estado actual del cliente, sus dependencias y restricciones, de manera que la *desinstalación* o *eliminación* del sistema de software no viole con estas dependencias o restricciones.

1.2 Actualización del Software.

Un sistema de software instalado está en constante evolución a través de extensiones, mantenimiento, cambios en los requerimientos o cambios en la configuración. Manejar la evolución de sistemas de software liberados e instalados es un problema complejo y frecuentemente subestimado que puede ser causa de dificultades tanto para el cliente, como para el proveedor del software.

El proceso de actualización de software puede ser visto como el movimiento de una configuración a otra por adición, eliminación, reemplazo o reconfiguración de las funcionalidades del software[3]. Una actualización física del software contiene alteraciones apropiadas de la funcionalidad y de la configuración.

En la actualidad se han creado herramientas con el propósito de cubrir las actividades relacionadas con la actualización del software. Dichas herramientas se conocen como actualizadores de software, un “actualizador” de software automatiza los pasos implicados en el proceso de actualización del software[4]. Las actividades del proceso de actualización del software se han generalizado en modelos, que intentan abarcar la mayor cantidad de características que un actualizador debe cumplir

para satisfacer los requisitos de despliegue de un software. A continuación se estudiará uno de los modelos de actualización de software más difundido en la bibliografía consultada.

1.2.1 Modelo general de actualización.

Acorde con R.Jensen [4] un modelo general de actualización de software tendría dos participantes: el cliente y el proveedor.

Dicho modelo planteado en la figura # 2 representa un diagrama de transición de estados por los cuales el cliente transita, así como la interacción entre el cliente y el proveedor del software. Los bloques representan estados en el diagrama, los estados finales son representados con los bordes subrayados. Las flechas sólidas son transiciones de estados, que se pueden activar tanto por el proveedor como por el cliente. Las flechas discontinuas demuestran la interacción entre ambos participantes. Una vez que el proveedor anuncie al cliente que la actualización del software esta disponible, comienza el proceso de actualización.

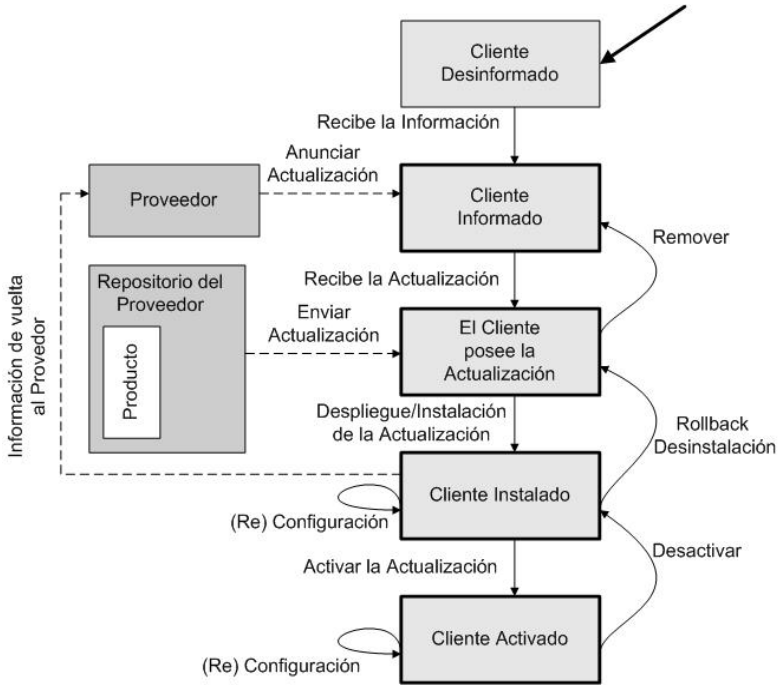

Figura 2 - Modelo General de Actualización

Cada paso del proceso tiene requisitos específicos y áreas problemáticas, siendo el envió y el despliegue pasos cruciales del mismo. Nótese además que dicho modelo es un subconjunto del ciclo de vida del despliegue de software presentado con anterioridad y que por tanto, presupone que la

versión del software ya esta liberada, aunque a su vez, en el modelo de actualización se repiten algunas actividades como la instalación o la activación.

1.2.2 Tipos de Actualizadores

Dentro de las tecnologías de actualización de software disponibles se han identificados cuatro tipos fundamentales de "actualizadores" [1]. Estos cuatro tipos son diferenciados observando los métodos de entrega y despliegue, así como las políticas y alcance del proceso. [5]

- Herramientas de Despliegue por Paquetes (HDP): Estas tecnologías de despliegue se basan en el concepto de paquetes y en un sitio de depósito o repositorio que almacena la información que representa el estado de cada paquete instalado. Un paquete es un archivo que contiene los ficheros que constituyen el sistema junto a algunos meta-datos que describen el mismo. Ejemplos de estas herramientas de paquetes son, APT[1], Loki Update, RPM-Update[2], Nix, SWUP[3] y Portage[4]. Siendo RPM, Portage y Nix las más avanzadas.
- Productos Actualizadores Genéricos (PAG): Estos actualizadores se abstraen completamente del producto a actualizar e intentan ser aplicables al proceso de actualización de cualquier producto. Tres PAG que están disponibles comercialmente son Red Carpet, InstallShield[5] y PowerUpdate[6].
- Productos Actualizadores del Proveedor (PAP): Este tipo de actualizadores facilitan el proceso de actualización de un solo producto, como Microsoft Windows XP Update, Exact Software's Product Updater y Symantec's LiveUpdate[7].
- Productos Actualizadores en Caliente(PAC): Este tipo de actualizadores [6] tienen como propósito actualizar el software sin interrumpir el servicio, es decir la ejecución del programa no se ve afectada, esto es particularmente útil en aplicaciones críticas como conmutadores telefónicos, procesadores de transacciones financieras, sistemas de reservación y control aéreo, entre otras.

[1] Advanced Packaging Tool, una herramienta informática de gestión de paquetes creada por Debian.
[2] http://www.kleemann.org/rpm-update/oldindex.html
[3] http://swup.trustix.org/
[4] http://www.gentoo.org/doc/en/portage-manual.xml
[5] http://www.macrovision.com/products/flexnet_installshield/index.shtml
[6] https://www.powerupdate.com/powerupdate/template.jsp
[7] http://www.symantec.com/press/1996/n960618b.html

1.2.3 Comparación entre los tipos de actualizadores existentes

Si se analiza como encajan cada uno de los tipos de actualizadores existentes con el modelo de actualización general presentado, se puede apreciar que existen claras diferencias entre ellos. Una de estas diferencias radica en la ausencia total de retroalimentación de información hacia el proveedor por parte de las Herramientas de Despliegue por Paquetes.

Los Productos Actualizadores Genéricos cubren gran parte del proceso, especialmente en las aéreas de Licencia e Interacción con el Usuario debido a que la inmensa mayoría son herramientas comerciales, además cuando se compara con los otros tipos de actualizadores tienen mayor cantidad de opciones para la retroalimentación de información del cliente hacia el proveedor.

Las Herramientas de Despliegue por Paquetes son especialmente diseñadas para desplegar e instalar paquetes mayormente en sistemas open-source. Cubren a cabalidad gran parte de los procesos de actualización del modelo general, pero las aéreas de interacción con el usuario y licencia son insuficientes, debido principalmente a dos razones: La primera es que temas como la retroalimentación de información hacia el proveedor son solucionados a otros niveles como sistemas de manejo de errores y comunidades de desarrolladores. Y en segundo lugar que la licencia no es un tema a considerar ya que la mayoría del software disponible en la comunidad open-source es gratuito.

Los Productos Actualizadores del Proveedor son generalmente débiles en las áreas de la transferencia y despliegue, sin embargo están fuertemente representados en las aéreas de interacción con el usuario y licencia. Una diferencia clara entre este tipo de actualizador y los Productos Actualizadores Genéricos es que la eliminación y el rollback[8] no están soportados por la mayoría de estos. Mientras que los Productos Actualizadores Genéricos asumen que los productos desplegados serán eliminados, los Productos Actualizadores del Proveedor asumen que sus productos y actualizaciones se mantendrán desplegados para siempre, lo cual no es sorprendente en el caso de los antivirus o de las actualizaciones de seguridad. Otra característica importante es que estos sistemas tienen funcionalidades restringidas porque han sido diseñados solo para un producto.

En la tabla # 1 se muestra como las técnicas de actualización evaluadas cumplen con los pasos del modelo de actualización general, así como los diferentes tipos de actualizadores existentes. Además se identifican las áreas en las cuales se centran algunas de las técnicas y dónde se necesita mayor investigación.

[8] Es la acción de volver a la configuración anterior del sistema instalado.

Producto	**Tipo**	**Interacción con el Cliente**			**Transferencia**		**Despliegue**				**Licencia**	
		Recibe Información	Anuncia Actualización	Retroalimentación de inf. al Proveedor	Recibe Actualización	Envía Actualización	Instalación de la Actualización	Rollback	Eliminar	Re-Configuración	Activa Actualización	Desactiva Actualización
PowerUpdate	PAG	○	○	○	●	○	○		●			
InstallShield	PAG	○	○	○	●	○	○		●		●	●
Red Carpet	PAG	●	●	●	●	●	●	○	●	○		
APT	HDP				●	○	●		●			
RPM-Update	HDP				●	○	●		●			
SWUP	HDP				●	○	○		●			
Portage	HDP	○	○		●	○	●	○	●	●		
Exact PU	PAP	○	○	○	●	○	○				○	○
WinXP SUS	PAP	●	●	○	●	○	○					
LiveUpdate	PAP	●	●		●	○	○				●	●
Leyenda: ●Soporte Completo ○Soporte Parcial												

Tabla 1- Comparación entre actualizadores, partiendo de los estados del modelo general.

Para una mayor comprensión de las técnicas de actualización se hace necesario proveer una gama de conceptos y definiciones para los procesos más importantes dentro del modelo de actualización,

siendo estos la *entrega* y la *instalación*. En la siguiente sección se explican cada uno de estos procesos y se evalúan las técnicas de actualización de software en base a estas definiciones.

1.2.4 Comparación teniendo en cuenta la entrega y la instalación

Entrega: El formato de la entrega identifica muchas de las características de los actualizadores. Algunos de los actualizadores como las Herramientas de Despliegue por Paquetes se centran solamente en la entrega de paquetes mientras que los Productos Actualizadores Genéricos intentan soportar la gran cantidad de formatos de entrega existentes. El formato de la entrega afecta considerablemente el tamaño de las actualizaciones que son enviadas a los clientes. La elección de un formato de entrega afecta completamente el modelo de entrega, especialmente en entornos con recursos limitados.

Nuevas configuraciones pueden ser entregadas al cliente de diferentes vías. Las configuraciones pueden ser transmitidas en los siguientes formatos:

- Paquetes de componentes: Un paquete de componentes puede ser entregado al cliente. Usualmente estos paquetes necesitan ser desempacados antes de poder instalarlos y activarlos. Ejemplos de técnicas que usan paquetes son RPM-Update, APT, DeployMe, RedCarpet, Portage and Nix.
- Componentes: Un componente separado consiste en un conjunto de ficheros.
- Archivos: La forma más simple de transferir datos son ficheros separados. Estos ficheros pueden ser licencias, valores de configuración y binarios.
- Archivos Deltas: Las diferencias entre la configuración del cliente y las del proveedor puede ser expresada en archivos deltas. Los archivos deltas pueden ser transferidos usando diferentes algoritmos como Rsysnc[7]. Un archivo delta es un listado de diferencias entre dos versiones de un mismo archivo, con cualquiera de las dos versiones se puede generar la otra. Enviar solo la diferencia entre dos archivos es más eficiente que enviar el archivo completo.

Sin algún tipo de pre-procesamiento en el cliente, cada uno de estos formatos podría traer restricciones en el entorno final del despliegue. Sin embargo, cuando son correctamente ensamblados antes del despliegue estos formatos son intercambiables, por ejemplo archivos deltas para un componente completo, pueden ser usados para generar un nuevo componente. La elección del formato de entrega afecta diferentes factores, como son el tamaño de la actualización y el método de despliegue, esto junto con las políticas de despliegue y las otras cuestiones del despliegue

completamente identifican un actualizador. Los Services Packs[9] son similares a los paquetes de componentes.

Instalación: El proceso de instalar las actualizaciones introduce más complejidad a los proveedores de software. La arquitectura de software de los sistemas determina la extensibilidad del mismo, por ejemplo, si las actualizaciones ocurren en caliente o si existen herramientas script disponibles para ejecutar ciertas tareas (como *Make*[10]). Finalmente se necesita chequear las dependencias, como las del sistema operativo, la presencia de ciertos componentes y la compatibilidad entre la actualización y la configuración existente en el cliente.

Para desplegar o instalar el software entregado, se necesita escoger un apropiado método de instalación. Algunos de estos métodos son los siguientes:

- Sobrescribir[4]: El método de instalación mas comúnmente utilizado por los proveedores de software es la sobre escritura de los ficheros de la aplicación, archivos de licencia y valores de configuración. La solución se basa en el supuesto que el conjunto de ficheros o componentes desplegados no cambian durante el tiempo por factores externos. No hay forma de dar marcha atrás a la sobre escritura, a no ser que el cliente este usando un sistema de archivos que soporte versiones. Un ejemplo de un método de actualización de sobre escritura es Windows Updater que primero desregistra una dll, luego la sobrescribe y después la registra de nuevo. Otro ejemplo es Exact Software Product Updater que compara todas las versiones de los ficheros locales disponibles con los ficheros remotos. Cuando existen diferencias, el actualizador solo sobrescribe los ficheros diferentes en el cliente.
- Plug-in: Los Plug-ins son frecuentemente usados para crear configuraciones extensibles[4]. El método de usar la arquitectura de plugin simplemente soporta la extensión de configuración por la adicción o eliminación de plugins. Otros plugins pueden manejar diferentes versiones del mismo plugin también.
- Des-Reinstalación[4]: Para muchas aplicaciones una actualización constituye la desinstalación de todo lo anteriormente instalado.

En las aplicaciones dentro de las comunidades de código abierto comúnmente la entrega y el despliegue se realizan a través de distribuciones de código. Estas distribuciones de código primero

[9] Una actualización a una versión de software que arregla problemas existentes, o provee una mejora del producto que aparecerá en la próxima versión.
[10] Herramienta de generación o automatización de código, muy usada en los sistemas operativos tipo Unix/Linux.

necesitan ser compiladas, lo que se puede ver como un proceso aparte del despliegue. Sistemas bien conocidos que asisten a las distribuciones de código están Maak[8] y Rtools[9]. Es valido mencionar que los tres métodos de instalación mencionados anteriormente pueden ser aplicables a las distribuciones de código.

Existen además otras cuestiones referentes al despliegue o instalación de la actualización como son la habilidad de ejecutar script para ciertas tareas, hacer análisis de dependencia, el chequeo de la integridad, soportar varias versiones del un mismo componente y permitir o no la tecnología push[11]. Cada una de estas habilidades inserta requerimientos específicos en el despliegue y la arquitectura implementada.

La ejecución de scripts es usado para configurar la actualización una vez que esté instalada. Estos script pueden ser usados para ejecutar, activar, configurar, compilar y construir una actualización. Pueden ser Shell Script, los cuales son comúnmente usados por las Herramientas de Despliegue por Paquetes, pero también lenguajes específicamente diseñados para registrar y des-registrar Plug-ins.

El análisis de dependencia tiene como principal objetivo proveer un listado completo y consistente de componentes a instalar, los necesarios debido a las dependencias existentes. Para lograr este objetivo muchos problemas necesitan ser afrontados, como son el soporte de múltiples versiones de un componente, la resolución automática y explicita de dependencias.

Finalmente la tecnología Push pone requerimientos extras en la implementación de la arquitectura de mensajes del actualizador. El cliente necesita estar preparado para recibir las actualizaciones automáticamente y el proveedor necesita estar al tanto de todos los espacios de trabajo en los clientes.

La entrega e instalación son los procesos más complejos dentro del modelo de actualización general debido a que existen muchas alternativas para eficientemente lograr los objetivos específicos de cada uno de estos procesos.

La comparación mostrada en la tabla #2 incluye una descripción de cuales son los formatos de entrega usados y cuales métodos de instalación o despliegue son soportados por cada actualizador, así como también se muestran otras características que identifican al actualizador.

[11] Tecnología Push en Internet se refiere al estilo del protocolo de comunicación donde las peticiones para una transacción determinada se originan desde el proveedor, editor o servidor central. Es contrario a la tecnología Pull donde las peticiones se originan desde el cliente o receptor.

	Tipo	**Formatos de Entrega**				**Políticas de Despliegue**			**Otras cuestiones referentes al Despliegue**				
		Paquetes	Componentes	Archivos	Archivos Delta	Sobrescribir	Plug-in	Des-ReInstalar	Uso de Scripts	Análisis de Dependencia	Chequeo de Integridad	Multiples Versiones	Push Technology
PowerUpdate	PAG	●	●	●	●	●		●	●		●		
InstallShield	PAG	●	●	●	●	●		●	●				
Red Carpet	PAG	●		●		●		●	●	●	●		○
APT	HDP	●						●	●	●	●		
RPM-Update	HDP	●						●	●	●	●		
SWUP	HDP	●						●		●	●		
Portage	HDP	●						●	●	●	●	●	
Exact PU	PAP			●		●							
WinXP SUS	PAP			●		●	●			○	○		
LiveUpdate	PAP			●		●	●				●		
Leyenda: ● Soporte Completo ○ Soporte Parcial													

Tabla 2 - Comparación entre los tipos de actualizadores

De la evaluación de los actualizadores en base a los conceptos y descripciones de entrega y despliegue presentados se deduce lo siguiente:

Para comenzar los Productos Actualizadores Genéricos soportan todos los formatos de entrega existentes, especialmente las dos herramientas más avanzadas en esta categoría, PowerUpdate e InstallShield, las cuales además son las únicas herramientas que tienen la capacidad de mandar archivos deltas en vez de archivos completos. Sin embargo los Productos Actualizadores Genéricos no están bien representados en las áreas que respectan al despliegue o instalación, ya que estas características son específicas del entorno de despliegue, del cual este tipo de actualizadores intenta abstraerse. Es significativo también que hagan uso parcial de la comercialmente interesante tecnología Push, especialmente cuando se compara con otras categorías de actualizadores. Este tipo de actualizador generalmente no hace uso de la tecnología de Plug-in lo cual puede ser explicado por el hecho de que los plugins dependen fuertemente de que la arquitectura del software utilizada sea orientada a plugins. Los Productos Actualizadores Genéricos hacen un fuerte uso del scripting ya que se requiere para realizar configuraciones post-instalación.

Las Herramientas de Despliegue por Paquetes soportan solo la instalación de paquetes y generalmente solo soportan desinstalación y reinstalación de un paquete actualizado. El scripting y el análisis de dependencias siempre están presentes en este tipo de actualizadores para habilitar configuraciones post instalación y chequeo de integridad con otros componentes. Las Herramientas de Despliegue por Paquetes no hacen uso de la tecnología Push, lo cual se explica por el hecho de que los usuarios de la comunidad de código abierto usualmente no desean que sistemas externos sean los encargados de realizar las operaciones de actualización. Sin embargo este tipo de herramientas están fuertemente representadas en las áreas referentes al chequeo de integridad y análisis de dependencias. El análisis de dependencias en este tipo de actualizador es requerido ya que los paquetes tienen muchas dependencias y relaciones entre ellos, la resolución automática de estas dependencias por lo tanto es un característica importante e imprescindible. Por su parte el chequeo de la integridad de los paquetes evita problemas de estabilidad y asegura autenticidad.

Finalmente, todos los Productos Actualizadores del Proveedor dependen de los archivos como principal formato de entrega a los clientes, estos archivos generalmente sobrescriben la instalación anterior, excepto cuando son plugins especiales como es el caso de las definiciones de virus para LiveUpdate o dlls[12] sin registrar para Microsoft Software Update System. Los Productos Actualizadores

[12] DLL es el acrónimo de Dynamic Linking Library (Bibliotecas de Enlace Dinámico), término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda del programa por parte del sistema operativo. Esta denominación

del Proveedor no incorporan a sus soluciones análisis de dependencias, scripting o chequeo de integridad y no hacen uso de la tecnología Push.

1.3 La actualización del software dentro de las plataformas de desarrollo más utilizadas.

En el desarrollo de software moderno se hace necesario, debido a la complejidad que impone un mercado cada vez más competitivo, un conjunto de facilidades, por así decirlo, que agilicen y aseguren un software de calidad. Las plataformas de desarrollo de software[13] vienen a cubrir un papel importante en el desarrollo de soluciones de software y pueden ser vistas como un "todo en uno" ya que, no solo brindan un lenguaje, sino que intentan independizarse de las arquitecturas de hardware / software y proveen de un espacio de trabajo (framework) que (junto con herramientas y utilidades) facilitan la puesta en marcha de la solución. Dos clásicos exponentes de este tipo de tecnología de software son Java y .Net, los cuales en la actualidad marcan una pauta para la creación de aplicaciones empresariales[14].

El objetivo de esta sección es exponer las diferentes alternativas que proveen las plataformas mencionadas para actualizar un software. Para lograr dicho objetivo se presentarán, junto con las alternativas, una breve descripción de ambas plataformas

1.3.1 Plataforma .NET

Es una plataforma de programación o desarrollo de software orientado a objetos, construida por Microsoft que simplifica el desarrollo en entornos distribuidos como Internet[10].

Tiene como elemento fundamental el marco de trabajo o framework de .Net que intenta cumplir con los siguientes objetivos[10]:

- Proveer un entorno de programación orientada a objetos consistente, donde el código objeto puede ser almacenado y ejecutado localmente, remotamente o ejecutado localmente pero distribuido en Internet.
- Proveer un entorno de ejecución que minimice los problemas de despliegue y de cambios de versiones, garantizando una ejecución segura y eliminando problemas de rendimiento de los ambientes interpretados.

---

se refiere a los sistemas operativos Windows siendo la extensión con la que se identifican los ficheros, aunque el concepto existe en prácticamente todos los sistemas operativos modernos.

[13] No existe una definición que plasme con exactitud el significado de este concepto.

[14] Típicamente es una aplicación de software hospedada en un servidor de aplicaciones que simultáneamente provee servicios a un gran número de usuarios sobre una red de computadoras.

- Unificar las experiencias adquiridas por los desarrolladores entre la gran cantidad de tipos de aplicaciones existentes, como las aplicaciones de escritorio, las aplicaciones Web, etc.
- Construir estándares de la comunicación que aseguren que el código de las aplicaciones hechas con este marco de trabajo puedan integrarse con otras.

Los principales componentes del marco de trabajo de .Net son[10]:

- El conjunto de Lenguajes de Programación:
- La Biblioteca de Clases Base o BCL[15]:
- El Entorno Común de Ejecución para Lenguajes o CRL[16] por sus siglas en inglés.

1.3.2 Opciones para actualizar un software en la plataforma .Net

Existen tres alternativas fundamentales para instalar una actualización de software en la Plataforma .NET, las cuales son[11]:

1. Despliegue usando la implementación “no-touch” de .NET.
2. Instalador de paquetes de windows.
3. Las actualizaciones automáticas.

Implementación “no-touch” de .NET:

Uno de los principales beneficios de usar la tecnología “no-touch” en una aplicación es lo fácil que se logra actualizar la misma. Si se hace un cambio en cualquier archivo del servidor Web, la próxima vez que el cliente ejecute la aplicación, esta se actualizará automáticamente, descargando los ficheros actualizados. Esto ocurre ya que el framework de .NET automáticamente chequea el estado de los archivos en uso, para verificar si necesitan ser descargados nuevamente o simplemente se ejecutarán desde la caché de ficheros de descarga o desde los ficheros temporales de Internet.

Instalador de paquetes de Windows:

El instalador de paquetes de Windows ofrece la solución más integral para actualizar aplicaciones basadas en .NET. Si se planea actualizar una aplicación usando la tecnología de instalador de Windows se tienen tres alternativas para implementarla[11]:

[15] Se clasifica en tres grupos fundamentales ASP.NET, Windows Forms, ADO.NET.
[16] Entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes

1. Crear un nuevo proyecto de despliegue y agregar las salidas de tu proyecto y otros archivos, como se hizo en la primera liberación de la aplicación.
2. Actualizar el proyecto de despliegue existente y asegurarse que los cambios en la aplicación son reflejados en el setup[17] y luego recompilar esa versión.
3. Se puede construir un paquete de parche (.msp) y aplicárselo a la aplicación instalada.

Actualizaciones automáticas:

Otra alternativa para lograr actualizar las aplicaciones hechas sobre .NET es la de mover la responsabilidad de la actualización de los administradores o usuarios a la aplicación en si. En este caso la aplicación cliente tendría que ser diseñada para automáticamente descargar e instalar las actualizaciones provenientes del servidor. Para lograr esto la aplicación tendría que[11]:

- Automáticamente chequear si existen actualizaciones.
- Descargar las actualizaciones existentes.
- Aplicar dichas actualizaciones a la configuración actual.

Para ayudar en la configuración de la aplicación para que soporte actualizaciones automáticas Microsoft ha creado un diseño reutilizable llamado “Updater Application Block” [12]. Dicho diseño plantea principalmente el uso de un servicio de Windows encargado de manejar el proceso de actualización.

1.3.3 Plataforma Java

Es una plataforma de desarrollo de software originaria de Sun Microsystems que está compuesta por un conjunto de tecnologías, cada una de las cuales ofrece una parte importante del desarrollo o del entorno de ejecución en tiempo real. Está compuesto por una gama de lenguajes de programación, la maquina virtual y el conjunto estándar de bibliotecas o librerías.

Existen tres ediciones de dicha plataforma:

1. Plataforma Java, Edición Estándar o Java SE.
2. Plataforma Java, Edición Empresarial o Java EE.
3. Plataforma Java, Edición Micro o Java ME.

[17] Programa de preparación, de montaje y de ajuste que se utiliza para configurar un sistema o aplicación para un entorno computacional determinado.

1.3.4 Opciones para actualizar software en la plataforma Java

Las estrategias de despliegue y actualización de las aplicaciones hechas en Java difieren en cada una de las ediciones de esta plataforma.

Así en la edición empresarial el despliegue de un software es relativamente más fácil cuando se compara con las aplicaciones hechas en Java SE o Java ME. Las razones vienen de la naturaleza de alto nivel de las especificaciones de esta edición, siendo el descriptor de despliegue un elemento fundamental de la misma. Además los programas Java EE corren en un ambiente estándar y bien definido, lo que ayuda en los procedimientos de despliegue y mantenimiento. Otro factor importante es que los programas Java EE son menos móviles y residen en un limitado número de servidores, administrados por expertos que a menudo son desarrolladores también[13].

Las aplicaciones Java SE ME difieren de las construidas en otras ediciones por sus librerías, además de que la memoria disponible varía de un dispositivo a otro. Esto es un problema cuando una misma aplicación es desplegada en diferentes dispositivos. La actualización de dichas aplicaciones comúnmente se realiza a través del medio inalámbrico o de conexiones punto a punto con una computadora en donde residiría el software encargado de realizar la actualización.

En la edición estándar de Java existe una tecnología llamada JNLP[18] con la cual se pueden realizar actualizaciones a un software de manera efectiva y segura, esta tecnología habilita la ejecución de programas Java SE remotos[13]. Naturalmente se necesita tener instalado en el cliente el entorno de ejecución de Java y los recursos necesarios para ejecutar el programa, por eso JNLP no es una tecnología de despliegue en sí, más bien es un protocolo de ejecución remota con algunas características agregadas. Estas características agregadas además de la ejecución de programas Java SE desde una URL[19], serían:

- Ofrecer un entorno en el cliente para ejecutar la aplicación.
- Asegurar un ambiente seguro.
- Realizar los procesos de chequeo de integridad y actualización de la aplicación.

[18] Java Network Launching Protocol de sus siglas en inglés, es un protocolo de ejecución en la red.
[19] Universal Resource Locator que significa Localizador Universal de Recursos. Es el término técnico que se utiliza para referirse a una dirección de Internet.

Existen varias implementaciones de este protocolo y a pesar de que la implementación de referencia de Sun ofrece mas funcionalidades, alternativas libres como OpenJNLP[20] o NETwork eXecute[21] están ganando en popularidad dentro de la comunidad de código abierto.

1.4 ¿Cómo se actualizan los productos creados por la UCI con características de distribución similares a SIGEP?

La Universidad de las Ciencias Informáticas (UCI) desarrolla en estos momentos un conjunto de proyectos que solucionan problemas sociales estratégicos de la Republica Bolivariana de Venezuela. Proyectos que abarcan la automatización de los Registros de Notarias, la Identificación de los Ciudadanos y la Gestión de los Procesos Policiales, comparten un único cliente y por ende tienen un conjunto de características de distribución y despliegue que se asemejan al Sistema de Gestión Penitenciaria. Por lo tanto, analizar el soporte que brindan estas soluciones para los procesos de actualización es una tarea prioritaria antes de brindar una propuesta de solución al problema que ocupa la investigación en curso.

El proyecto Identidad cuyo objetivo principal es identificar la población venezolana está distribuido en diferentes puntos geográficos donde la identificación es un proceso crítico. Dicho proyecto está implementado en la Plataforma .Net y se beneficia de la estrategia de actualización automática que la plataforma propone a través del Updater Applicatión Block[12], basado en dicho diseño se ha implementado un mecanismo de actualización genérico que extiende las funcionalidades y brinda métodos de configuración adecuados para dicho proceso, de manera que el uso del mecanismo no es especifico de una solución de software en particular sino de la plataforma de desarrollo utilizada.

El proyecto registro de notarias también tiene gran parte de su funcionalidad implementada usando la Plataforma .Net, por lo que se ajustó el mecanismo construido por el Proyecto Identidad a fin de automatizar el proceso.

El proyecto encargado de la Gestión de los Procesos Policiales en Venezuela será construido usando la Plataforma Java, pero se encuentra en una temprana etapa del ciclo de desarrollo de software y todavía no han identificado las características de despliegue necesarias que regirán los procesos de actualización de la solución informática.

[20] http://openjnlp.nanode.org/index.html
[21] http://jnlp.sourceforge.net/netx/index.html

1.5 Conclusiones

A pesar de que el estudio de las tecnologías existentes en el mercado, las soluciones construidas por proyectos de la Universidad y las alternativas que proporcionan las diferentes plataformas de desarrollo, brinden un marco teórico apropiado para la investigación en curso, no existe, en la actualidad, un mecanismo de actualización o componente reutilizable adecuado para realizar las actualizaciones de software necesarias para el Sistema de Gestión Penitenciarias.

Capítulo 2: Características del mecanismo de actualización.

Introducción

En el presente capítulo se exponen las características de despliegue del Sistema de Gestión Penitenciaria, se profundiza en la problemática que implicaría actualizar este sistema manualmente y se identifican las características que debería cumplir un mecanismo de actualización automatizado. Además se presenta una adaptación del modelo de actualización general estudiado en el Capítulo # 1 que satisface las necesidades de dicho mecanismo y luego se argumentan a través de una primera aproximación al flujo de actividades necesarias para su implementación así como las entidades identificadas en dicho flujo.

2.1 Características de despliegue del Sistema de Gestión Penitenciaria.

El Sistema de Gestión Penitenciaria (SIGEP) tiene como propósito general dar respuesta a las necesidades de gestión, información y apoyo a la toma de decisiones de la Dirección General de Custodia y Rehabilitación del Recluso (DGCRR), en sus tres niveles, que se describen a continuación:

- **Operativo**: integrado por los Establecimientos penitenciarios (Internados Judiciales y Centros Penitenciarios), Centros de Tratamiento Comunitario (CTC) y Unidades Técnicas de Apoyo al Sistema Penitenciario (UTASP).
- **Táctico**: integrado por las Coordinaciones Regionales.
- **Estratégico**: integrado por la Dirección General de Custodia y Rehabilitación del Recluso.

Para obtener dicho objetivo se hace imperativo mantener comunicación en línea de los treinta (30) centros penitenciarios, diecinueve (19) Centros de Tratamiento Comunitario, treinta y tres (33) Unidades Técnicas de Apoyo al Sistema Penitenciario (UTASP) y cinco (5) coordinaciones regionales, distribuidos por todo el país con la Dirección General de Custodia y Rehabilitación del Recluso.

La solución informática concibe un sistema Web construido sobre la plataforma Java EE que estaría desplegado en el Servidor Web Apache Tomcat de cada uno de los Centros Penitenciarios e Internados Judiciales. La base de datos sería local a cada centro. Los Centros de Tratamiento Comunitario, las UTASP y las coordinaciones regionales se conectarían a una aplicación central, donde radicarían sus datos y la lógica de procesos que se ejecutan.

Se establecerá una infraestructura de comunicaciones que incluya conexiones WAN[22] entre el centro de datos y los 30 puntos de centros penitenciarios, 19 CTCs, 33 UTASP y 5 coordinaciones regionales, distribuidos en todo el territorio nacional. Se prevé además un canal de comunicación de alta velocidad entre la Dirección General y el centro de datos.

La figura # 3 presenta el esquema de la arquitectura de distribución en cada centro penitenciario y su comunicación con el centro de datos ubicado en el MIJ:

- Los establecimientos penitenciarios, CTC, UTASP y coordinaciones regionales (en general las sedes) se conectarán al Ministerio a través de una red privada virtual (VPN[23]) que transmitirá información al Ministerio.
- Según su ubicación geográfica, las sedes podrán conectarse a Internet usando Internet Banda Ancha o una conexión Celular. La velocidad deseada es de 1,5mbps tanto de subida como de bajada.
- El acceso a los servidores del Ministerio estará protegido por sistemas cortafuegos (firewalls) y servidores de autenticación, que sólo permitan la entrada a los usuarios y máquinas autorizadas.
- Esta arquitectura proporcionará conectividad segura vía túneles VPN's encriptados entre sedes remotas y la sede central, a través del proveedor de telecomunicaciones seleccionado.

[22] Red de comunicaciones que conecta ordenadores dispersos en una amplia área geográfica.
[23] Red Virtual Privada, red de comunicación privada utilizada por la red publica, utiliza protocolo de eficiencia y seguridad de información para guardar la privacidad de la información.

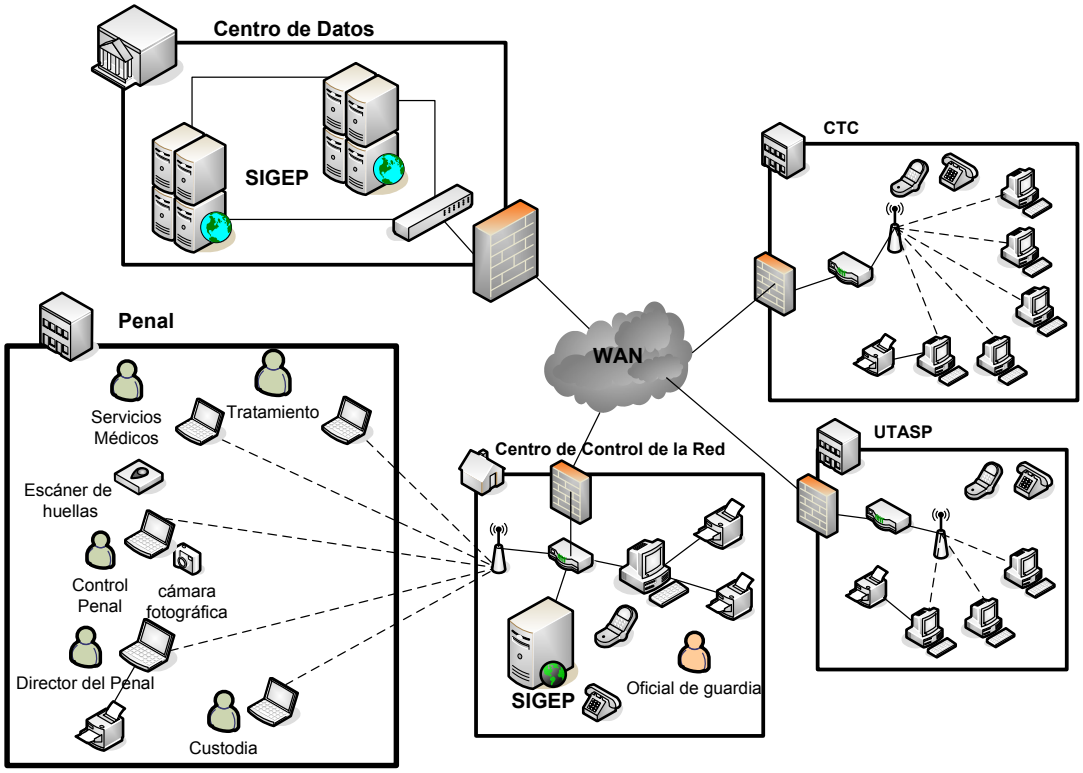

Figura 3 Arquitectura de distribución de SIGEP

2.2 Problemática de la actualización manual del Sistema de Gestión Penitenciaria

El hecho que el Sistema de Gestión Penitenciaria tenga las características de distribución anteriormente expuestas, agrega complejidad adicional al mantenimiento de la solución informática una vez implantada. Los parches de seguridad y las actualizaciones necesitan ser instalados debidamente en cada uno de los centros penitenciarios, así como en el centro de datos. Para que esta actividad pueda ser realizada con éxito, se deben considerar un conjunto de factores como la distancia, los recursos humanos, la logística y la seguridad institucional.

Los factores que intervienen en el proceso de actualización del Sistema de Gestión Penitenciaria se describen a continuación:

1. La geografía distante de los centros penitenciarios trae como consecuencia que las actividades destinadas a actualizar el sistema en su totalidad, tomen un tiempo significativo, esto puede dar como resultado que existan diferentes versiones de la aplicación en uso al mismo tiempo.

2. Las actividades de actualización requieren de la participación de un personal técnico con conocimientos sobre administración de aplicaciones Web sobre Java EE y con dominio del sistema operativo Linux. Este personal no existe actualmente en la institución, por lo que se requiere su presencia oportuna a fin de recibir la preparación necesaria para desempeñar la actividad.

3. Se necesitan cuantiosos recursos materiales por concepto de transporte y alojamiento del personal técnico responsable de la actividad, considerando además el medio de almacenamiento (CDs, Memory Flash, etc.) necesario para el traslado de la nueva versión del sistema.

4. Por el carácter sensible de la información que gestiona el sistema y el hecho de que el mismo sea construido para una organización gubernamental incorpora requisitos extras para la seguridad y la integridad de la información. Este tipo de sistema intenta involucrar la menor cantidad de personal posible para garantizar en alguna medida que el uso de a la aplicación quede dentro de los marcos previstos por la institución. En este sentido, las actividades relacionadas con la actualización del sistema de gestión penitenciaria introducen riesgos potenciales referentes a la seguridad institucional debido a la gran cantidad de personal involucrado en dichas actividades.

2.3 Características que debe tener la actualización de SIGEP.

Los problemas que implica el desempeño manual de las actividades de actualización del sistema de gestión penitenciaria y la importancia que tiene el soporte y el mantenimiento de dicho sistema una vez implantado, crean la necesidad de automatizar dichas actividades a través de un mecanismo de actualización.

Dicho mecanismo debe cumplir con las siguientes características:

- Para que los procesos de actualización se puedan manejar de manera centralizada y no exista la necesidad de aplicar la actualización en cada uno de los locales donde el sistema este en uso, el mecanismo debe ser remoto.

- El mecanismo debe garantizar la integridad del sistema, el sistema debe mantenerse funcional antes y después de realizada la actualización. Si por algún motivo la actualización falla, el sistema debe mantenerse en ejecución con la versión anterior e informar la existencia de un problema en el proceso, para su posterior solución.

- El mecanismo debe prever la inexistencia de conectividad, es decir, de no existir conectividad en los locales donde el sistema este en uso, los procesos de actualización se realizarían automáticamente en cuanto retorne la conectividad.
- El mecanismo debe retroalimentarse de información, es decir, en todo momento debe informar el estado de la actualización, de esta manera se puede medir el progreso del proceso completo e identificar los lugares puntuales donde la actualización no se pudo llevar a cabo.
- La comunicación debe realizarse por canales seguros de comunicación y se deben aplicar medidas para garantizar la veracidad de los datos.

Estas características no son únicas de la actualización del sistema de gestión penitenciaria sino que pueden ser retomadas en cualquier sistema que se enfrente a una problemática similar con los procesos de actualización de software.

2.4 Modelo de actualización

Basado en el modelo de actualización general expuesto en el Capítulo 1 y representado en la figura # 2, se ha modelado un mecanismo de actualización de software que cumple con las características expuestas en la sección anterior. Dicho modelo, es representado en la figura # 4 también a través de un diagrama de transición de estados por los que transita el proveedor de la actualización y el cliente encargado de actualizar el sistema, además el modelo incluye la interacción entre el proveedor y el cliente. El punto de partida, a diferencia del modelo general, es cuando el proveedor todavía no posee la nueva versión del producto.

La siguiente lista muestra todos los pasos del proceso en el modelo de actualización de software presentado:

Publica el Producto: La nueva versión del producto es publicada en el sitio del proveedor, usualmente se realiza mediante un servicio accesible para los clientes a través de algún protocolo de comunicación como http[24] o ftp[25].

Notifica nueva versión del producto: Cuando la nueva versión del producto se encuentre en el sitio del proveedor se notifica al cliente la existencia de la versión. Este comunicado pudiera contener

[24] Protocolo de transferencia de Hipertexto. Se usa para transmitir documentos de hipertexto en la Web.
[25] Método de transferencia de archivos por Internet utilizado para descargar archivos públicos de una computadora remota a una local.

información adicional, como por ejemplo, los cambios de una versión a otra, el tiempo que va ha estar disponible, etc.

Recibe la Notificación: El cliente se mantiene a la espera de que exista una nueva versión del producto, una vez que se envía la notificación, esta es automáticamente recibida por el cliente, nótese que el proceso de recepción de la notificación comúnmente involucra algún tipo de chequeo que garantice que la notificación proviene del lugar indicado y que la notificación enviada contiene la misma información que la recibida.

Procesa Notificación: El cliente procesa la notificación, este proceso implica hacer un análisis de la aplicación instalada contra la información del nuevo producto, de este análisis se genera una descripción de la información que el cliente necesita para realizar la actualización.

Pide la información necesaria: El cliente luego de tener la descripción de la información, necesaria para llevar a cabo los demás procesos que intervienen en la actualización del producto, pide al sitio del proveedor dicha información, o simplemente la adquiere si la información ya se encuentra disponible para descargar.

Envía la actualización: El servidor le envía la actualización, es decir, solo la información necesaria, al cliente. Cómo se estudio en el Capítulo 1 el formato de la actualización es variable, aunque sin duda los formatos más utilizados son los archivos y los paquetes.

Despliegue o Instalación de la actualización: Esta es, como se explicó con anterioridad, la actividad más compleja dentro del modelo de actualización tanto en el general, como en el adaptado por los autores. Está directamente relacionada con una de las técnicas de instalación antes presentadas, la sobre escritura y con la variabilidad del entorno de ejecución.

Activar la Actualización: Luego de la instalación, la actualización debe ser activada, para que la nueva versión del producto pueda ser usada por los usuarios finales. Esta conlleva a cambios en las configuraciones y en la ejecución del software actualizado.

RollBack: Si el cliente no es instalado con éxito, el mecanismo vuelve a la versión anterior del producto, esto implica tener algún sistema de respaldo de información.

Remover: La información que fue descargada o recibida del Proveedor necesita ser eliminada completamente del cliente para que próximas actualizaciones puedan volver a descargarse sin afectar el ambiente de la actualización.

Información devuelta al Proveedor: En cualquiera de los estados que se encuentre el cliente una vez informado, el mecanismo permite enviar información de vuelta al proveedor, de esta forma el proveedor conoce en que estado se encuentra cada uno de los clientes que se están actualizando.

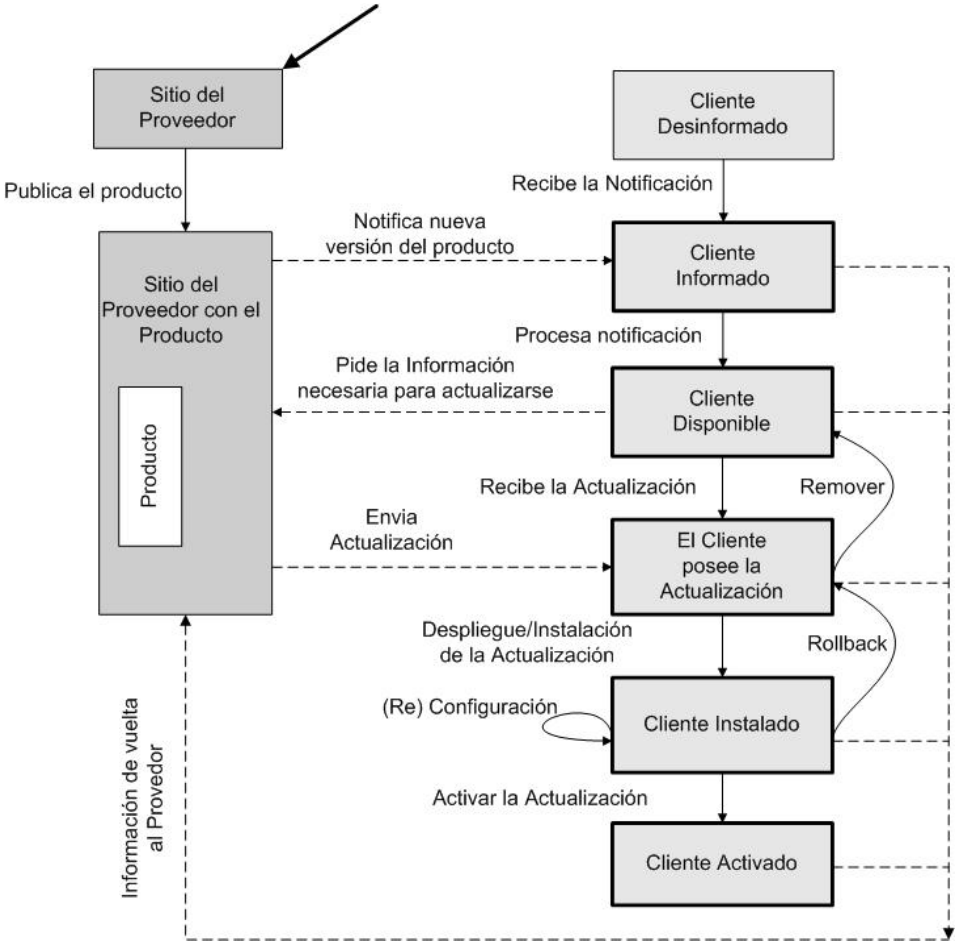

Figura 4 Modelo de actualización de Software

2.5 Mecanismo de actualización de SIGEP

La automatización del mecanismo de actualización del sistema de gestión penitenciaria cumple en las premisas del modelo de actualización adaptado en el epígrafe anterior y agrega nuevas funcionalidades al proceso de actualización.

En la confección del mecanismo se han dividido las funcionalidades en tres aplicaciones informáticas con responsabilidades definidas de la siguiente forma:

- La aplicación "*Visualizador de Información*" es la encargada de la interacción con el responsable de la actualización, permitiéndole visualizar el estado del proceso general a través de una interfaz amigable.
- La aplicación *"Proveedor de Actualización*" es la encargada de notificar y servir las actualizaciones para cada centro penitenciario en particular.
- La aplicación "*Actualizador*" es la encargada de escuchar los eventos que provienen del *Proveedor de Actualización* y realizar actividades en el cliente que garanticen que la actualización enviada sea instalada y activada correctamente.

De conjunto, estas tres aplicaciones colaboran para realizar actividades, estas actividades han sido generalizadas en procesos que describen de manera general como funciona el mecanismo. Los procesos identificados son los siguientes:

1. Publicar la nueva versión de SIGEP:
2. Notificar la existencia de una nueva versión de SIGEP.
3. Generar el pedido de los recursos necesarios para actualizar al software.
4. Proporcionar o servir el paquete de actualización.
5. Conformar el sistema actualizado.
6. Aplicar la actualización.
7. Informar el Estado.

Los procesos enumerados necesitan una representación estructural de la información que procesan, generan o utilizan para intercomunicarse entre ellos. Por esta razón se identificaron entidades que describen, no solo la información, sino también el formato.

Las entidades identificadas se describen a continuación:

- Aplicación empaquetada: Es la aplicación web que se desea actualizar empaquetada en un fichero WAR[26].

[26] Acrónimo de Web Application Archive, es el tipo de archivo que se utiliza para empaquetar las aplicaciones Web en Java EE.

- Notificación: Fichero XML[27] que contiene un resumen de los recursos de la aplicación a actualizar.
- Listado de los recursos necesarios: Fichero XML que contiene el listado de la información de todos los recursos que el *Actualizador* necesita para actualizar la aplicación.
- Paquete de actualización: Son todos los recursos cuya descripción está contenida en el "Listado de recursos necesarios" empaquetados y compactados en un fichero ZIP[28].
- Tabla General de Actualización: Esta contenida en el *Visualizador de Información* y lista el estado de todas las actualizaciones que se encuentran en ejecución en ese momento. Es actualizada asincrónicamente por las aplicaciones *Proveedor de Actualización* y *Actualizador.*

Estas entidades junto con los procesos mencionados y el flujo de información se representan en la figura # 5 a través de un diagrama de procesos. Nótese que en este diagrama las entidades están representadas con el icono de documento y que el rombo con la cruz en el medio indica un "AND" lógico, es decir que se realizan todos los procesos a los que apunta. El proceso inicia con la publicación del sistema y termina con la aplicación de la actualización.

[27] eXtensible Markup Language. Un lenguaje de marcado extensible que puede usarse para almacenar datos en un formato estructurado, basado en texto y definido por el usuario.
[28] Extensión de los ficheros comprimidos con el algoritmo ZIP.

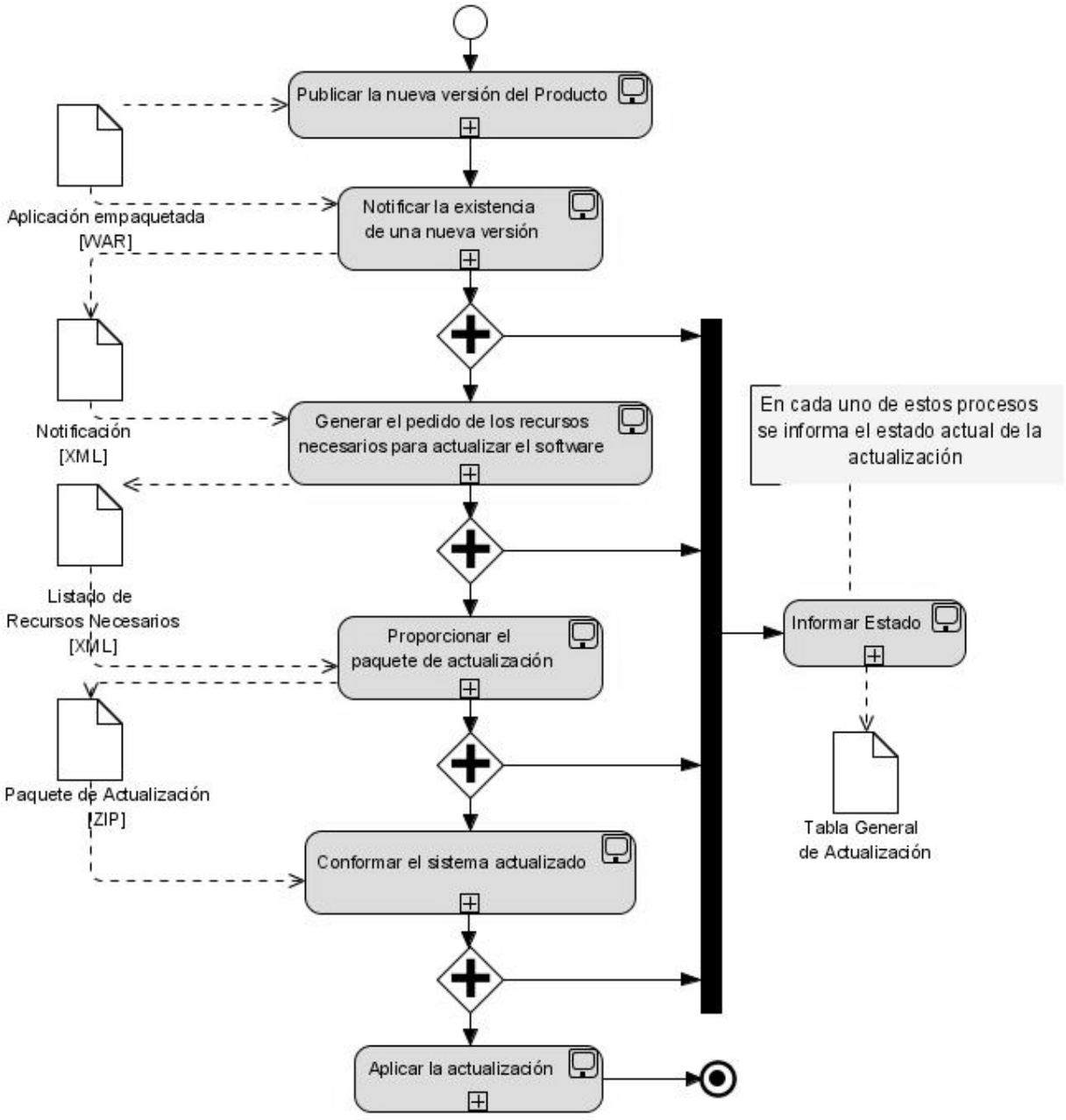

Figura 5 Mecanismo de Actualización de SIGEP

Cada uno de los procesos representados en la figura # 5 serán descritos a continuación en el mismo orden en que fueron enumerados con anterioridad. En la descripción de estos procesos sí se tiene en cuenta la interacción entre las aplicaciones, identificado cuales aplicaciones son responsables de cada una de las actividades o tareas especificas realizadas en el proceso.

1. Publicar la nueva versión del producto.

En la publicación de la nueva versión interviene el técnico responsable de la actualización del Sistema de Gestión Penitenciaria a través del *Visualizador de Información*. Se localiza donde se encuentra la nueva versión del sistema, en este caso la aplicación web empaquetada en un archivo WAR, luego se le envía el archivo al Proveedor de Actualización, este lo recibe y lo descomprime en un directorio temporal, para de esta forma acceder más rápidamente a cada uno de los recursos de la aplicación a actualizar.

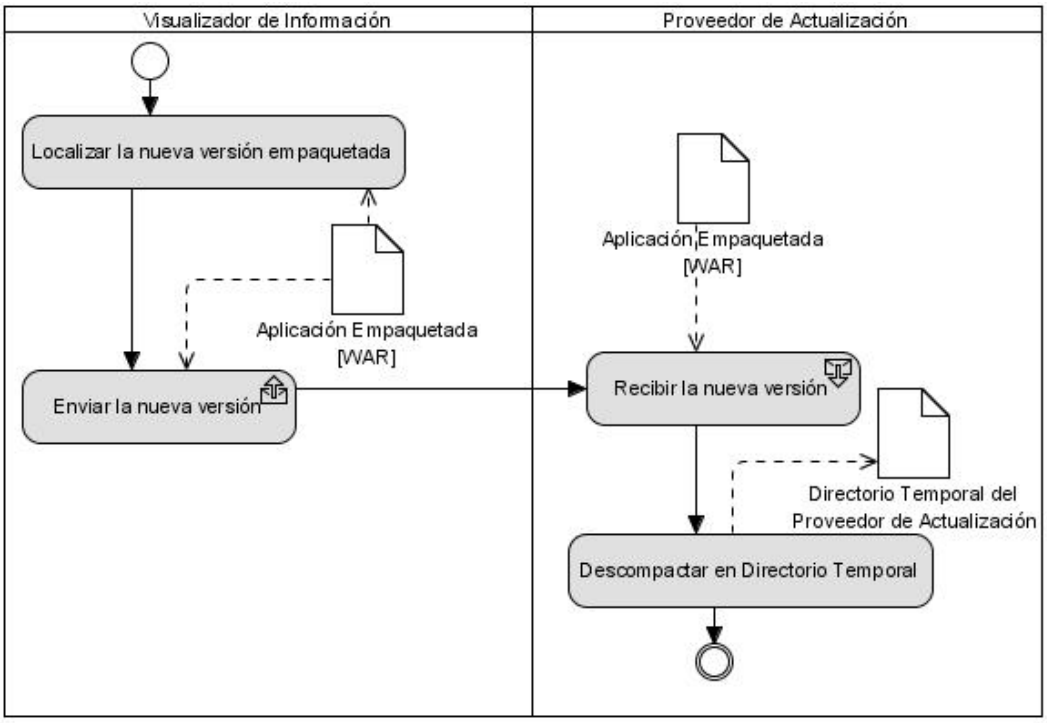

Figura 6 - Publicar la nueva versión del producto

2. Notificar la existencia de una nueva versión.

Para notificar la existencia de la nueva versión el *Proveedor de Actualización* genera un fichero XML con la información de cada uno de los recursos de la aplicación a actualizar, esta información contiene la localización del recurso así como una cadena que representa el estado del recurso. El fichero es enviado al *Actualizador,* este lo recibe y verifica que el fichero proviene del *Proveedor de Actualización*

y que el fichero enviado es el mismo que el recibido a través de un algoritmo resumen como MD5[29] o SHA-1[30].

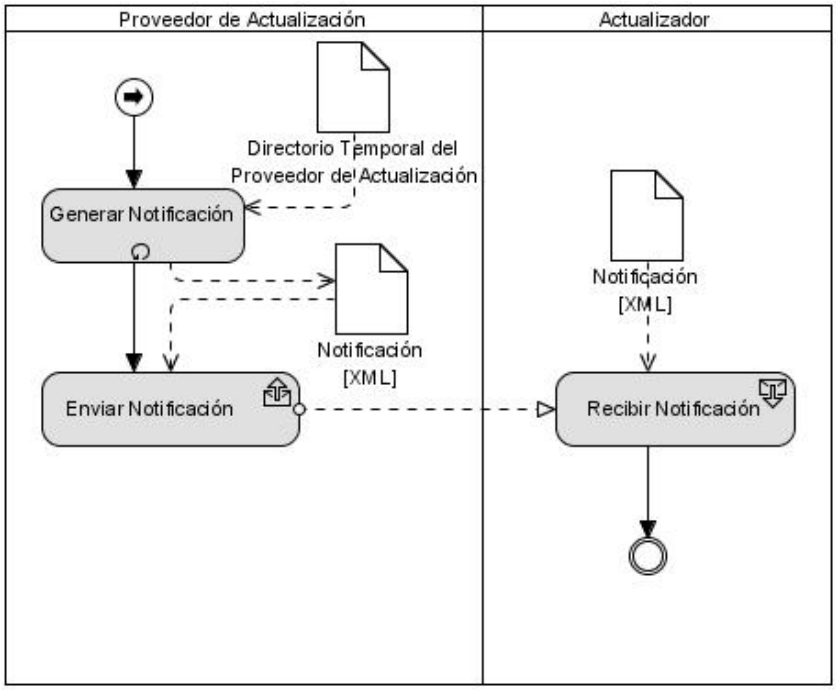

Figura 7 - Notificar la existencia de una nueva versión

3. Generar el pedido de los recursos necesarios para actualizar el software.

Para generar el pedido de recursos el *Actualizador* analiza la información proveniente del fichero de Notificación y a través de la localización y del estado de los recursos logra identificar los elementos que se agregan en la nueva versión, los que se modifican y los que se eliminan. Con los elementos que se agregan y los que se modifican el *Actualizador* conforma un fichero XML jerárquico que envía al *Proveedor de Actualización.*

[29] acrónimo de Message-Digest Algorithm 5 o Algoritmo de Resumen del Mensaje 5 es un algoritmo de reducción criptográfico de 128 bits amplimente usado.
[30] De la familia SHA (Secure Hash Algorithm, Algoritmo de Hash Seguro) es un sistema de funciones hash criptográficas relacionadas.

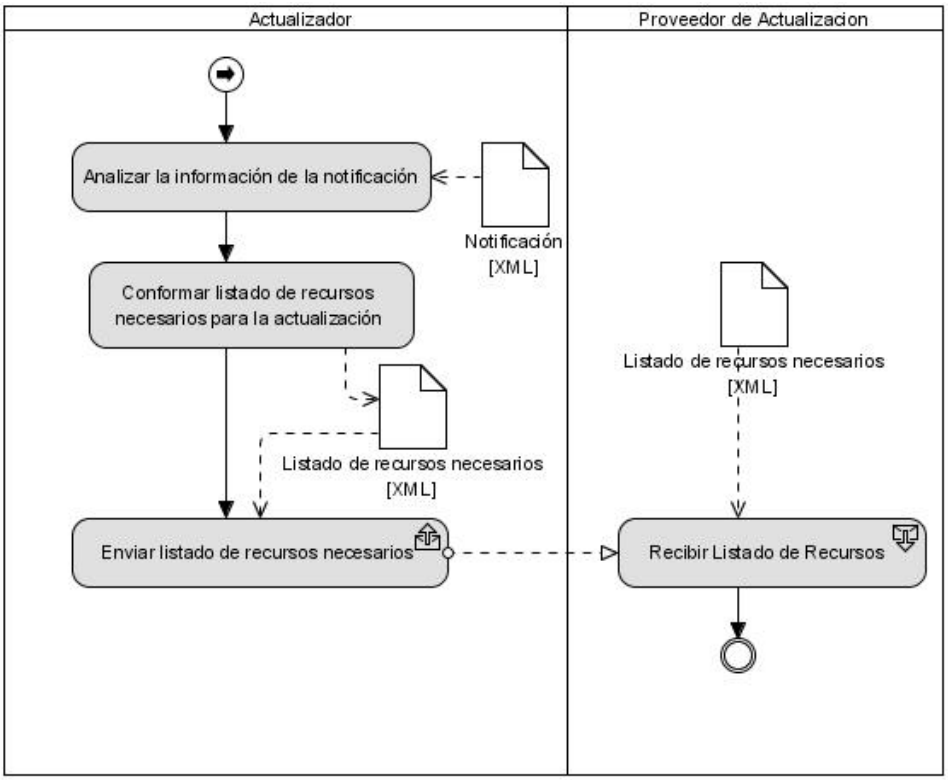

Figura 8 - Generar el pedido de los recursos necesarios para actualizar el software

4. Proporcionar o servir el paquete de actualización.

Para servir el paquete de actualización el *Proveedor de Actualización* obtiene de la lista de recursos la localización de cada elemento, luego carga todos los elementos por su localización del directorio temporal y conforma el paquete de actualización, agregando cada uno de los elementos a un fichero compactado. Se envía el paquete de actualización y es recibido por el *Actualizador*.

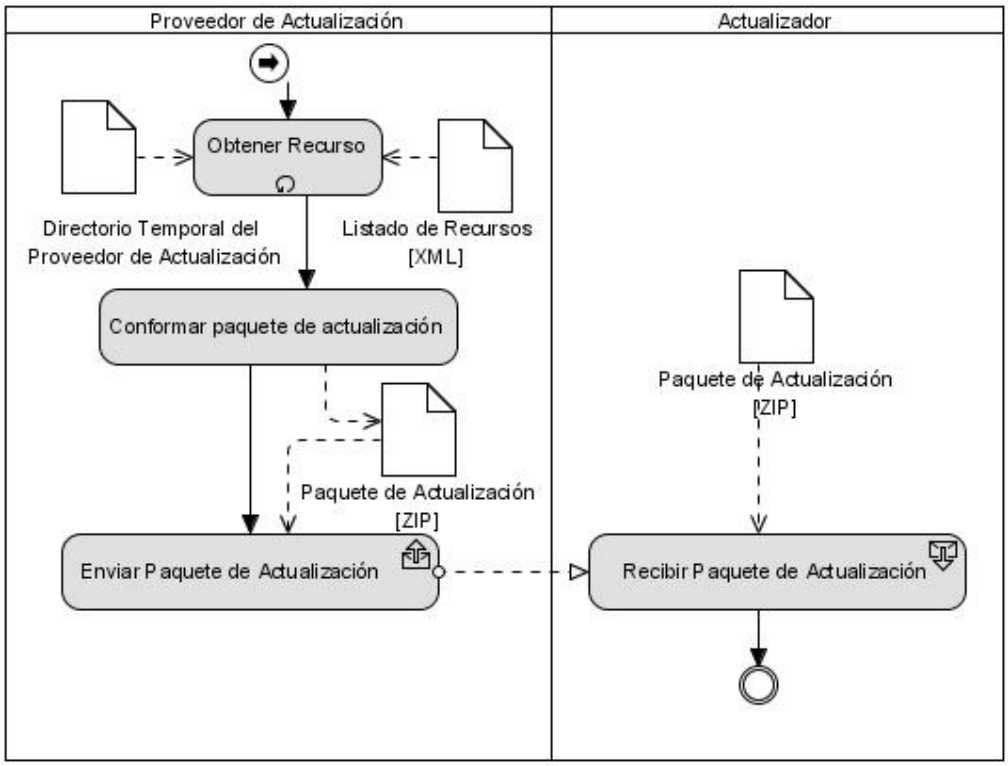

Figura 9 - Proporcionar o servir el paquete de actualización

5. Conformar el sistema actualizado.

Para conformar el sistema actualizado el *Actualizador* descomprime el paquete de actualización en un directorio temporal, luego hace una copia del sistema instalado en otra carpeta, elimina los recursos inexistentes en la nueva versión de la versión instalada y copia o sobrescribe los recursos agregados o modificados en la nueva versión. Por último se agregan elementos de configuración de esta versión y se configuran apropiadamente. Nótese que el hecho de que todas las acciones se realicen sobre una copia del sistema instalado garantiza la marcha atrás o rollback del proceso en caso de que exista algún problema.

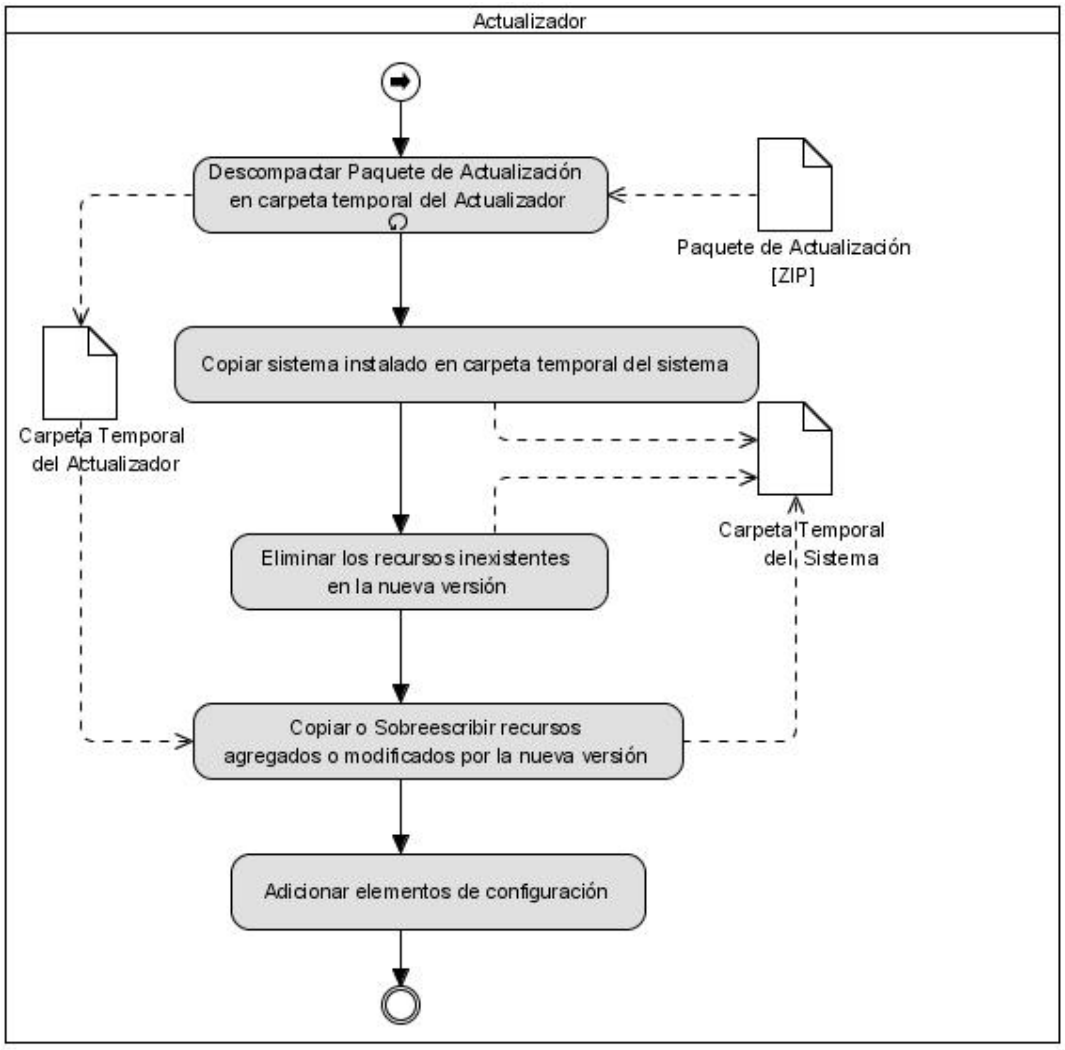

Figura 10 - Conformar el sistema actualizado

6. Aplicar la actualización.

Para aplicar la actualización, es decir, dejar el sistema actualizado listo para su explotación, se necesita interactuar con el servidor web ya que la versión anterior de la aplicación se encuentra ejecutándose, por lo que el *Actualizador* debe parar la aplicación en curso, copiar por sobre escritura la nueva versión y luego iniciarla, o interactuar con el servidor para que la inicie.

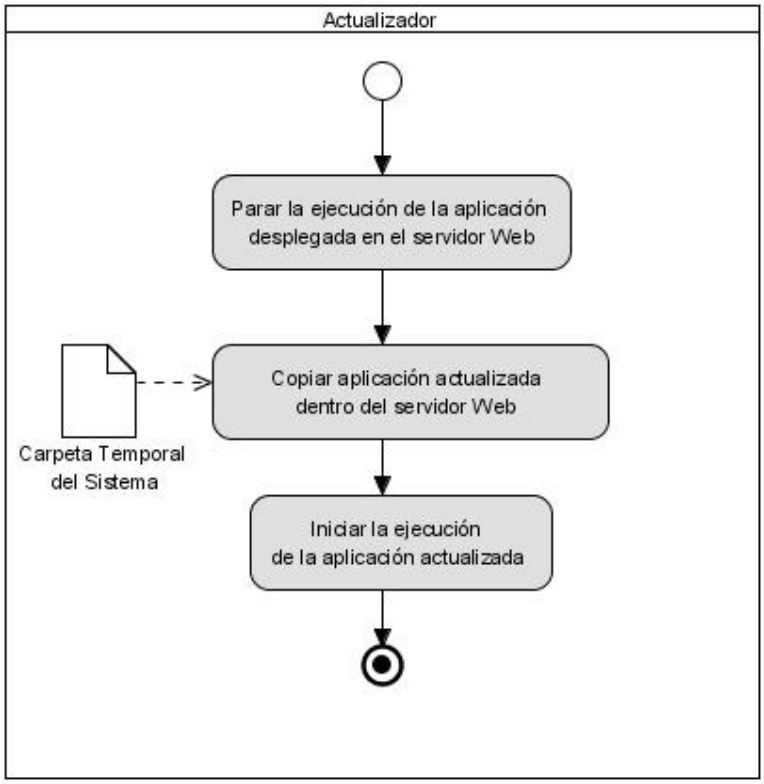

Figura 11 - Aplicar la actualización

7. Informar estado de la actualización.

Este proceso es crucial en el mecanismo, ya que asegura que el responsable de la actualización conozca a través del *Visualizador de Información*, el estado del proceso de actualización especifico de cada uno de los centros penitenciarios. Tanto el *Actualizador* como el *Proveedor de Actualización* deben enviar el estado actual de la actualización en curso, la responsabilidad del Visualizador de Información es la de mantener actualizada la tabla general de actualización y mostrarla al responsable.

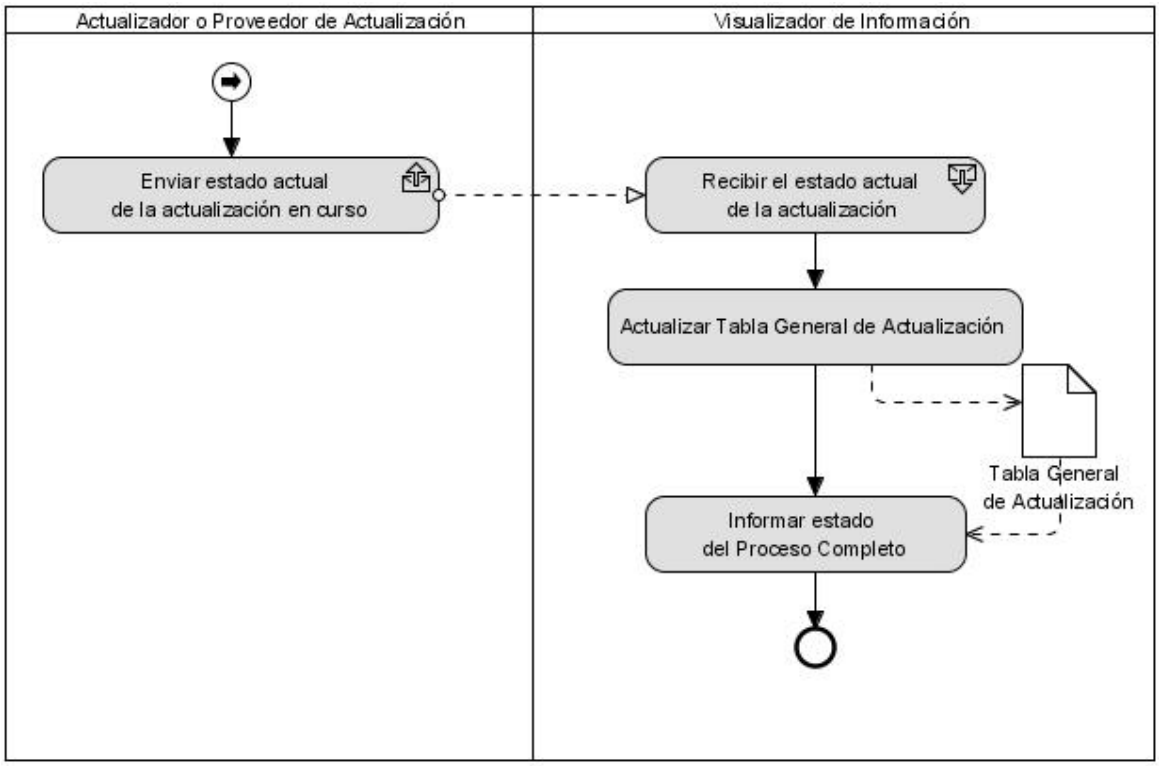

Figura 12 - Informar estado de la actualización

2.6 Conclusiones

En este capítulo se ha adaptado el modelo de actualización de software estudiado en el Capítulo # 1 para dar solución a la problemática existente en el sistema de gestión penitenciaria, además se introdujo el flujo de actividades necesarias que el mecanismo debe cumplir, así como las entidades identificadas en el mismo.

Capítulo 3: Construcción de la solución propuesta.

Introducción

En el presente capítulo se expone cómo se ha implementado el mecanismo propuesto en el capítulo dos. Se empieza hablando sobre las aplicaciones que se han construido para implementar el mecanismo. Seguidamente se describen las herramientas usadas por todas estas aplicaciones brindándose el por qué de su utilización. El diseño de las clases de cada una de las aplicaciones construidas es otro tema que se aborda. También se explica cómo interactúan todas estas aplicaciones para realizar el proceso de actualización y cómo colaboran con un servidor de mensajería para transmitir los datos de una aplicación a otra. En el capítulo también se detallan los beneficios de la mensajería y el por qué de su utilización. Finalmente se explica cómo hacer para que el mecanismo propuesto pueda ser actualizado también.

3.1 Construcción de la solución.

Como se ha visto en el capítulo 2, se propone la construcción de 3 aplicaciones para implementar el mecanismo de actualización propuesto.

- Una aplicación P*roveedora* de la actualización que se encargue de anunciar y de enviar la actualización al cliente como bien se muestra en la figura # 4.
- Una aplicación *Cliente* que se encargue de recibir, instalar, configurar y activar la actualización, además de poder desactivarla, desinstalarla y de borrar dicha actualización como se muestra en la figura # 4.
- Una aplicación que sirva como interfaz gráfica al usuario para mandar la actualización a los clientes y que brinde información de retroalimentación al usuario sobre el estado de la actualización en cada cliente tal como, si la actualización está en progreso, si la actualización completó exitosamente, si la actualización no completó, etc.

Las tres aplicaciones están hechas sobre la plataforma Java. El hecho de estar construidas en el lenguaje Java hace que estas aplicaciones sean multiplataforma, lo que quiere decir que el mismo software puede ser ejecutado en diferentes sistemas operativos sin que sea necesario cambiar ninguna línea de código. Por ejemplo, estas aplicaciones pueden ser ejecutadas en sistemas Windows como Windows Xp, Windows 2000, de la misma forma que en sistemas GNU/Linux, como Debian, Ubuntu, Gentoo, RedHat, etc.

Para la construcción de dichas aplicaciones se ha hecho uso del marco de trabajo Spring. Un marco de trabajo es una estructura de soporte definida en la cual otro proyecto de software puede ser

organizado y desarrollado. Los marcos de trabajo son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. En el caso de Spring tenemos un marco de trabajo que se integra con varios servidores de mensajería y en especial con ActiveMQ (que se verá a continuación)[14] Al integrarse Spring con ActiveMQ la interacción de la aplicación con este último se hace mucho más simple ya que las tareas de bajo nivel se dejan al marco de trabajo ocupándose la aplicación de las tareas funcionales únicamente. Este marco de trabajo oculta por completo al programador la tediosa tarea de tratar con las excepciones lo cual hace el código más simple y pequeño y además hace que la configuración de las aplicaciones construidas con este sea más sencilla.

Como el proveedor de actualización y el cliente estarán geográficamente distantes uno del otro es necesario un medio seguro y confiable por el cual transferir la actualización para lo cual se ha utilizado el servidor de mensajería Apache ActiveMQ. Se ha decidido utilizar este servidor de mensajería sobre los otros tantos que existen como Fiorano, Sonic, Joram, etc. por las siguientes razones:

- Simplicidad y sencillez de configuración. Esto hace posible que la puesta en marcha del servidor de mensajería sea un proceso rápido y seguro, ya que a medida que aumenta la complejidad de configuración aumenta el riesgo de que algún parámetro se halla olvidado o algo por el estilo.
- Soporte de una gran variedad de clientes. Esto quiere decir que podemos interactuar con ActiveMQ desde Java, C, C++, C#, Ruby, Perl, Python, PHP,[15] lo cual posibilita que tanto el proveedor de actualización como el cliente puedan ser escritos en cualquiera de estos lenguajes. Esto podría ser útil si en una futura versión del sistema sería más factible escribir los clientes en otro lenguaje distinto de Java.
- Soporte de Spring. ActiveMQ puede ser fácilmente embebido y configurado en una aplicación hecha con el marco de trabajo Spring lo cual aumenta la facilidad de configuración que este tiene.
- Soporte de varios protocolos de comunicación como TCP, SSL, UDP, etc.[15]. De acuerdo con las características de la aplicación SIGEP, es necesario que la información y en este caso la actualización entre el proveedor y los clientes viaje por un medio seguro de comunicación. ActiveMQ brinda una manera fácil de enviar mensajes sobre SSL lo cual hace que los paquetes de actualización viajen por un medio fiable y seguro.

3.1.2 Proveedor de actualización

El Proveedor de Actualización es una aplicación que cumple con las funcionalidades expuestas en el Modelo de actualización de Software de la figura # 4. Esta aplicación ha sido construida de tal forma que no tiene una interfaz gráfica con la cual pueda interactuar el usuario y ya que el proceso de actualización puede durar varios días en dependencia del estado de las conexiones de red de los clientes se hace necesario ejecutarla como un demonio sobre el sistema operativo. Por tanto, como existe otra aplicación que es quien tiene una interfaz visual con la cual el usuario puede interactuar (el Visualizador de Información) esta aplicación (el proveedor) necesita una forma de comunicación con el Visualizador de Información y para ello se ha utilizado el servidor web Jetty.

Un servidor web es un programa que implementa el *protocolo HTTP* (*hypertext transfer protocol*)[16]. Este protocolo está diseñado para transferir lo que llamamos hipertextos, páginas web o páginas HTML (hypertext markup language): textos complejos con enlaces, figuras, formularios, botones y objetos incrustados como animaciones o reproductores de música.

Una característica distintiva del servidor web Jetty entre los muchos que existen es que puede ser embebido en una aplicación[17]. Es decir, que la misma aplicación puede iniciar, detener y reiniciar el servidor web sin necesidad de que esto se tenga que hacer manualmente. Por lo tanto Jetty permite que el Visualizador de Información se comunique con el Proveedor a través del protocolo http y https. La ventaja que trae este tipo de comunicación es que el Visualizador de Información puede construirse tanto con tecnologías web como con cualquier otra tecnología que brinde interfaces visuales al usuario, como ventanas, menús, etc.

3.1.2.1 Diseño del Proveedor de Actualización

En la figura # 13 se muestra el diseño de clases del Proveedor de Actualización. En esta figura aparecen todas las clases que se han desarrollado. Además se ha representado el marco de trabajo Spring y la biblioteca de clases JMS. El marco de trabajo Spring aparece como una nube con el borde sombreado y la biblioteca de clases JMS como una nube sin el borde. Se ha elegido representar estos dos como una entidad atómica para evitar la representación de todas las clases que son utilizadas de estos por la aplicación y que harían el diagrama muy complejo y poco entendible.

3.1.2.2 Descripción del las clases mas importantes del diseño

InitAdminService: Esta clase contiene el método principal de la aplicación (el método main) por lo tanto es el punto de inicio de ejecución para el “Proveedor de actualización”. Este método es el encargado de inicializar el contexto de Spring y de instanciar al servidor web Jetty el cual instanciará el servlet

UpdateNotificatorServletService en espera de la notificación de actualización desde el Visualizador de Información.

UpdateNotificatorServletService: Esta clase es un servlet de java que escucha las peticiones http hechas por el Visualizador de Información, y reenvía la notificación de actualización mediante la clase UpdateNotificator.

UpdateNotificator: Esta clase es la encargada de mandar la notificación recibida desde UpdateNotificatorServletService al Actualizador. Mediante la clase NotificationGenerator genera un XML con la descripción de la nueva actualización y se manda al servidor de mensajería ActiveMQ mediante las interfaces que brinda el marco de trabajo Spring para que este la envíe al Actualizador.

NotificationGenerator: Esta clase genera un XML describiendo la nueva actualización. La descripción es formada por el nombre de todos los ficheros de la aplicación junto con una clave hash, la cual el Actualizador utilizará para identificar si algunos de los ficheros antiguos han sido cambiados o dañados.

RequestUpdateListener: Cuando los clientes reciben la notificación de actualización junto con la descripción de esta, generan un pedido de actualización, que puede contener el pedido de antiguos ficheros debido a modificaciones de estos, entre otras razones. Esta clase recibe el pedido de actualización de los clientes para con este conformar la actualización que se mandará a estos.

UpdateGenerator: Esta clase con el pedido que viene desde el Actualizador, que no es mas que un XML bastante parecido al que se envía con la notificación pero que contiene el nombre de los ficheros que se actualizará, genera un fichero que se manda al Actualizador mediante la clase UpdateSender.

UpdateSender: Esta clase manda el fichero de actualización generado por UpdateGenerator al Actualizador a través del servidor de mensajería ActiveMQ mediante las interfaces que brinda el marco de trabajo Spring. Esta clase en realidad es una interfaz del lenguaje Java que sirve para implementar el patrón estrategia. Esto es útil para escoger el medio por el que se enviará la actualización al Actualizador, puede ser por ejemplo, mediante mensajería, mediante conexiones de Sockets, etc.

DestinationUtil: Sirve como clase de utilidad general para hacer conversiones y validaciones en el Proveedor de Actualización.

ZipUtil: Esta clase sirve como utilidad para descomprimir, el archivo de actualización que se envía desde el Visualizador de información, el cual debe ser un fichero WAR, especifico de la plataforma Java 2 Enterprise Edition para empaquetar aplicaciones web.

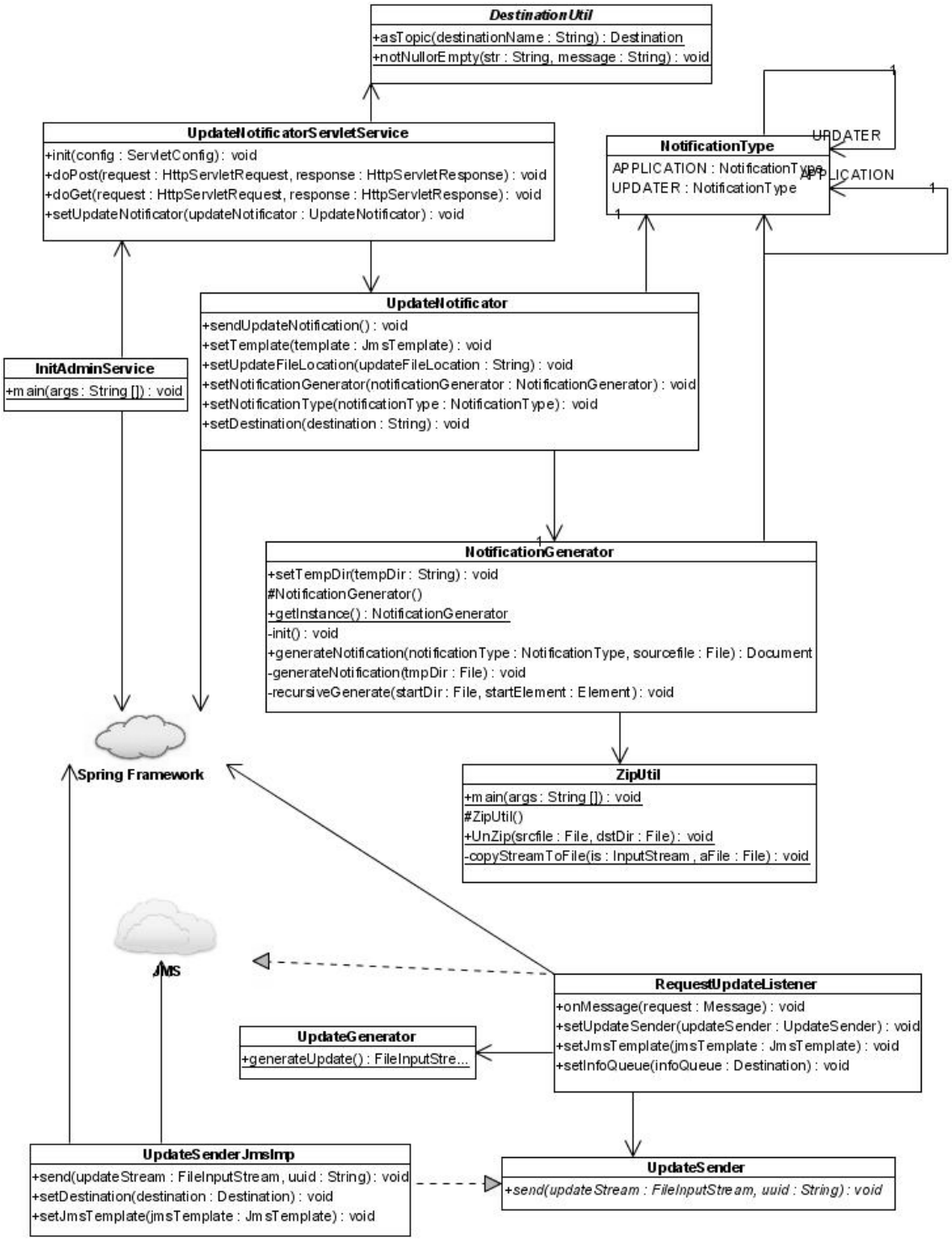

Figura 13 - Diagrama de clases del Proveedor de Actualización

3.1.3 Actualizador

El actualizador ocupa el mismo lugar que el cliente de la figura # 4. Al igual que el Proveedor de Actualización, el Actualizador hace uso del marco de trabajo Spring por las razones ya mencionadas e interactúa con el servidor de mensajería ActiveMQ para recibir la actualización e intercambiar mensajes con el Proveedor de Actualización y enviar el estado de la actualización al Visualizador de Información.

3.1.3.1 Diseño del Actualizador

En la figura # 14 se muestra el diseño de clases del Actualizador. Al igual que en el diagrama del diseño de clases del Proveedor de Actualizaron para hacerlo más censillo se ha representado el marco de trabajo Spring y la librería de clases JMS como una entidad atómica.

3.1.3.1 Descripción de las clases más importantes del diseño

AppHelper: Esta clase es quien contiene el método principal de la aplicación (el método main) por lo tanto es el punto de inicio de ejecución para el Actualizador. Es quien inicializa el contenedor de Spring, el cual se encarga de inicializar las conexiones con el servidor de mensajería y dejar lista la aplicación para recibir alguna notificación de actualización.

NotificationListener: Esta clase es la encargada de recibir las notificaciones de actualización desde el Proveedor de Actualización. Con la notificación de actualización viene también información que describe la nueva actualización. Esta clase teniendo en cuenta la información que recibe y la que tiene de la propia aplicación en uso, realiza un pedido de actualización y la manda al proveedor de actualización para que este mande los ficheros que se desean actualizar.

UpdateReceiverJmsImp: Esta clase es quien recibe a través del servidor de mensajería haciendo uso de Spring el archivo de actualización generado en el Proveedor de actualización que el propio Actualizador manda a pedir

Updater: Esta clase en colaboración con TomcatManager se encargará de realizar el proceso de actualización final sobre el sistema en uso que estará ejecutándose en un servidor web Apache Tomcat. La actualización en sí, se resume a copiar archivos nuevos, a borrar o modificar otros existentes.

UUIDFactory: Se ha visto, anteriormente que no habrá una sola instancia del Actualizador ejecutándose, sino que serán varias en dependencia de la cantidad de sistemas que se desplieguen y como la actualización se enviará a un actualizador en específico hace falta una manera de

identificarlos a cada uno. Esta clase genera un identificador único de tipo UUID para cada instancia del Actualizador que se esté ejecutando lo cual sirve para identificar a cada Actualizador en el servidor de mensajería.

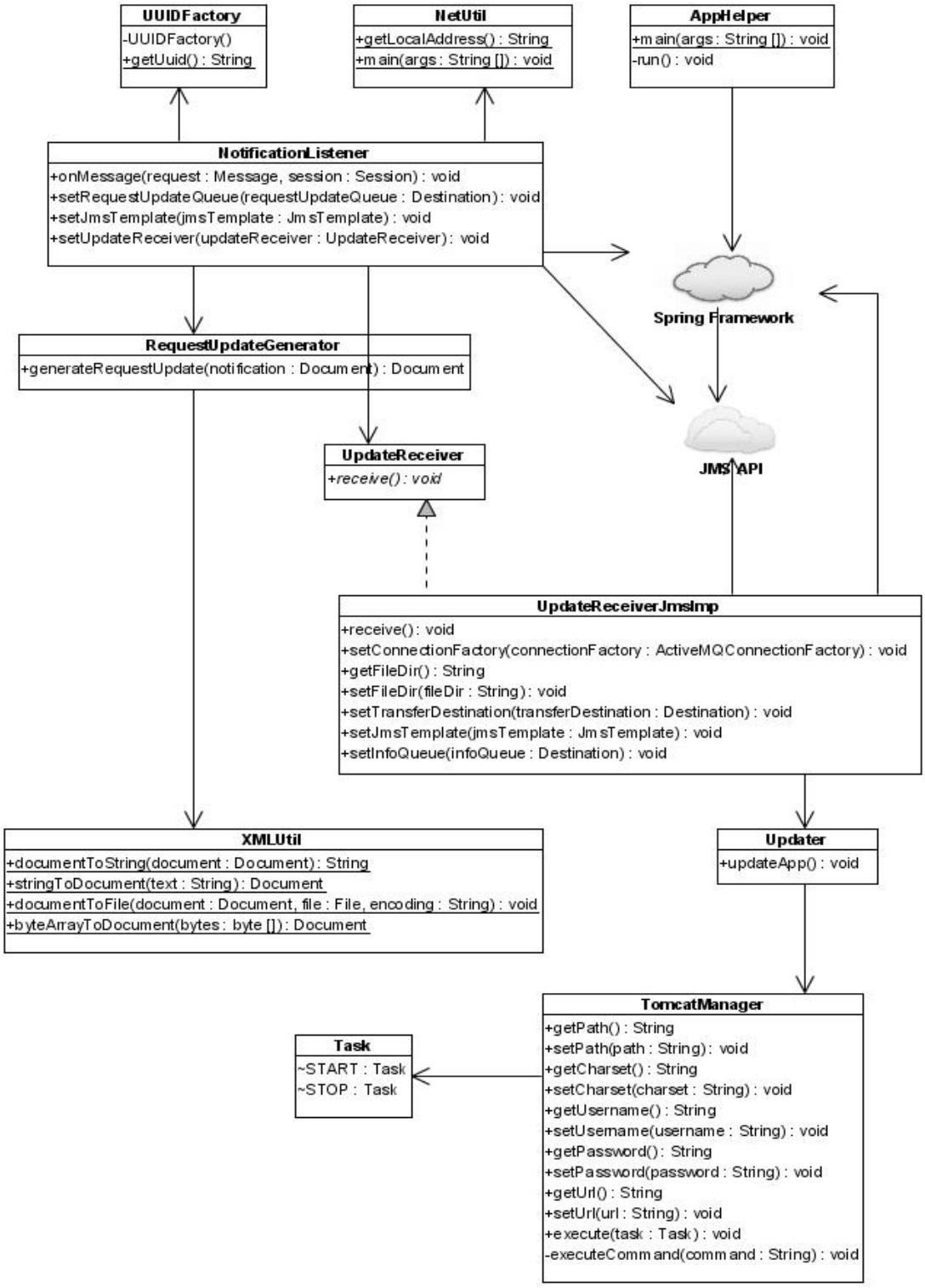

Figura 14 - Diagrama de clases del actualizador

3.1.4 Visualizador de Información

Como se ha visto hasta aquí, se tiene una aplicación encargada de notificar y mandar la actualización (el Proveedor de Actualización) y una aplicación encargada de realizar la actualización al sistema que se encuentra en explotación (el Actualizador). Estas aplicaciones son necesarias para implementar el mecanismo propuesto pero no son suficientes porque el usuario no tiene una manera fácil de realizar la actualización. Debido a esto es necesario realizar una tercera aplicación que brinde una interfaz amigable para el usuario que realizará la actualización y que brinde información del estado de cada actualización en progreso. Esta información indicará el progreso de la actualización, por ejemplo, si se recibió la notificación, si se hizo el pedido de actualización al Proveedor, si se mandó la actualización al cliente, si la actualización se completó exitosamente o si ocurrió un error en algunos de estos pasos.

3.1.4.1 Diseño del Visualizador

A continuación se muestra un diagrama que muestra las relaciones entre las clases más importantes del Visualizador de Información. De igual forma que en el caso del Proveedor de Actualización y el Actualizador se ha representado el marco de trabajo y el API JMS como entidades atómicas.

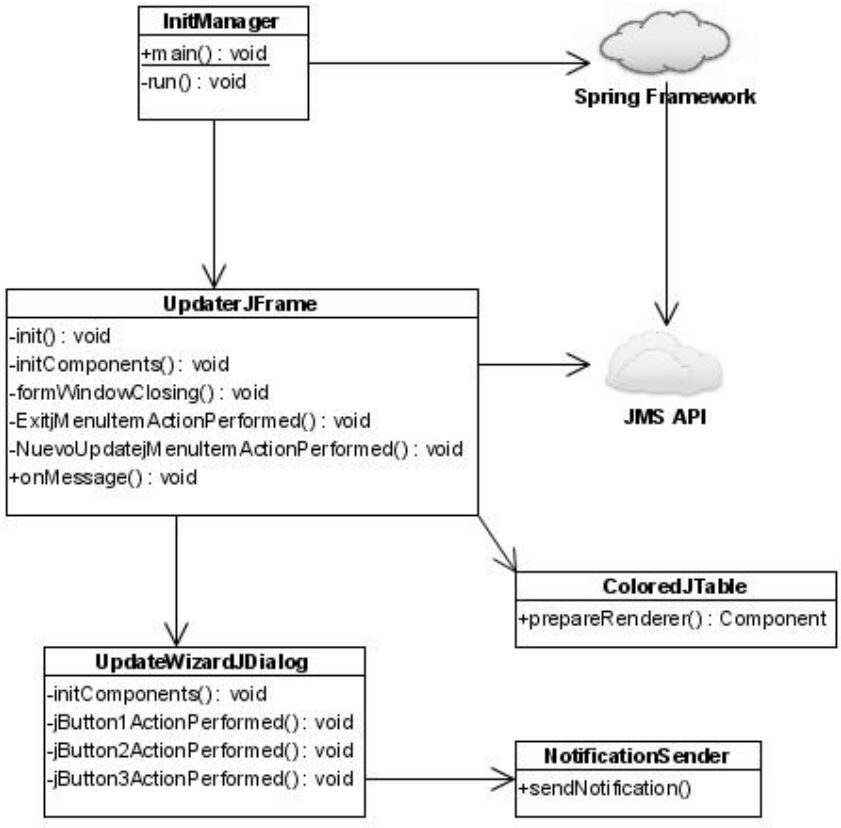

Figura 15 - Diagrama de Clases Visualizador de Información

3.1.4.2 Descripción del las clases más importantes del diseño

InitManager: Esta clase es el punto de inicio de ejecución del Visualizador de Información y quien se encarga de inicializar y ejecutar los componentes gráficos de este. Además inicia el contenedor de Spring, el cual se encarga de inicializar las conexiones con el servidor de mensajería y dejar lista la aplicación para recibir información de retroalimentación referente al estado de las actualizaciones.

UpdaterJFrame: Esta clase es el contenedor de todos los componentes gráficos de la aplicación. Hace uso del API Swing para implementar las funcionalidades gráficas.

NotificationSender: Esta clase encapsula la lógica para mandar la notificación al Servidor de Actualización para que este la mande a los Actualizadores.

ColoredJtable: Esta es una clase auxiliar para dar color a las celdas de la tabla que muestra información referente al estado de las actualizaciones.

3.2 Construcción del Modelo de actualización para SIGEP

En este epígrafe se verá como cooperan las tres aplicaciones para poner en funcionamiento el mecanismo de actualización propuesto.

En la implementación del mecanismo propuesto se ha hecho el intercambio y la transferencia de los datos entre las aplicaciones mediante mensajería asíncrona[18], utilizando para ello el servidor de mensajería ActiveMQ expuesto anteriormente en este mismo capítulo.

Para poder usar un servidor de mensajería y por tanto enviar mensajes a través de este, es necesaria la configuración de determinados objetos en el servidor de mensajería, y uno de estos son las colas de mensajes. Estas colas tienen el objetivo de almacenar los mensajes enviados por un emisor hasta que el servidor de mensajería haya entregado los mensajes al receptor correspondiente[18].

En la figura # 15 se muestra la configuración de las colas que se han configurado en el servidor de mensajería Apache ActiveMQ los cuales son utilizados por las tres aplicaciones para intercambiar datos.

3.2.1 Descripción de las colas de mensajería

Tópicos:

*All.UpdateNotification.Topic: A e*ste tópico es a donde se manda la notificación de actualización y al cual los Actualizadores están escuchando por la notificación.

Colas:

RequestUpdate.Queue: A esta cola es a donde los Actualizadores mandan el pedido de actualización y al cual el Servidor de Actualización está escuchando por dicho pedido.

Transfer.Queue: A esta cola es a donde el Servidor de Actualización manda la actualización y a la cual los Actualizadores están escuchando para recibirla.

Info.Queue: A esta cola es a donde tanto el Servidor de Actualización como el Actualizador mandan información referente al estado de la actualización y a la cual el Visualizador de Información está escuchando para mostrarlo al usuario mediante una interfaz amigable.

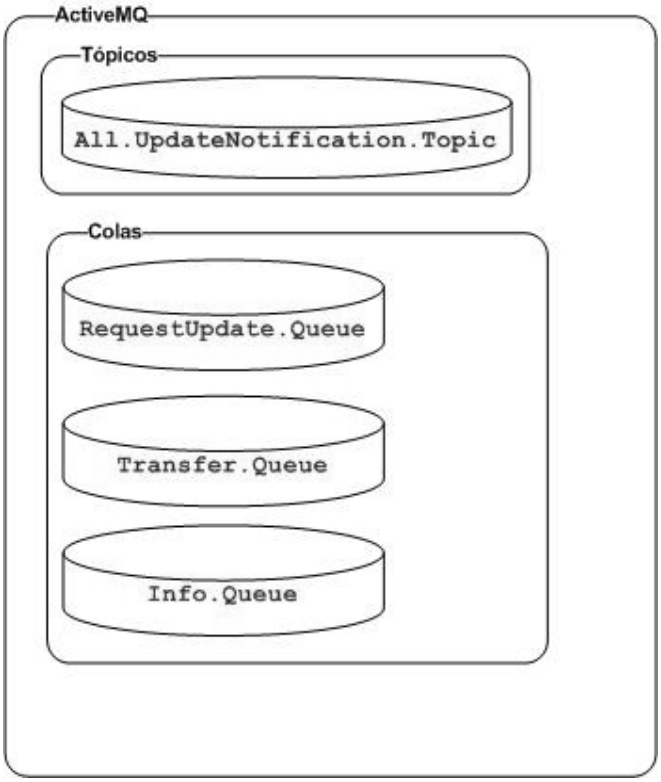

Figura 16 - Configuración de Colas (Queue) y Tópicos (Topics) en ActiveMQ

3.2.2 Funcionamiento del mecanismo de actualización implementado

La figura # 17 muestra la relación entre los componentes de cada aplicación y cómo colaboran para llevar a cabo el proceso de actualización en la forma del mecanismo propuesto en el capítulo anterior. Estos componentes engloban un conjunto de tareas para llevar a cabo su responsabilidad. Cuando un usuario (en este caso será un usuario capacitado para realizar la actualización) desea actualizar el sistema lo hace por medio del visualizador de información.

El módulo *UI_Visualizaciónr* es el encargado de mostrar las ventanas y demás gráficos con los que el usuario interactuará. Cuando el usuario manda la actualización (es un archivo con extensión .war que tiene una estructura especifica de las aplicaciones web de la plataforma Java y básicamente además de tener una estructura determinada lo que hace es empaquetar varios archivos en uno solo y compactarlos), la aplicación la envía a través del módulo *Envío*, y este manda el fichero de actualización hacia el Proveedor de Actualización, donde el módulo *Notificación* del Proveedor recibe el archivo que contiene los datos.

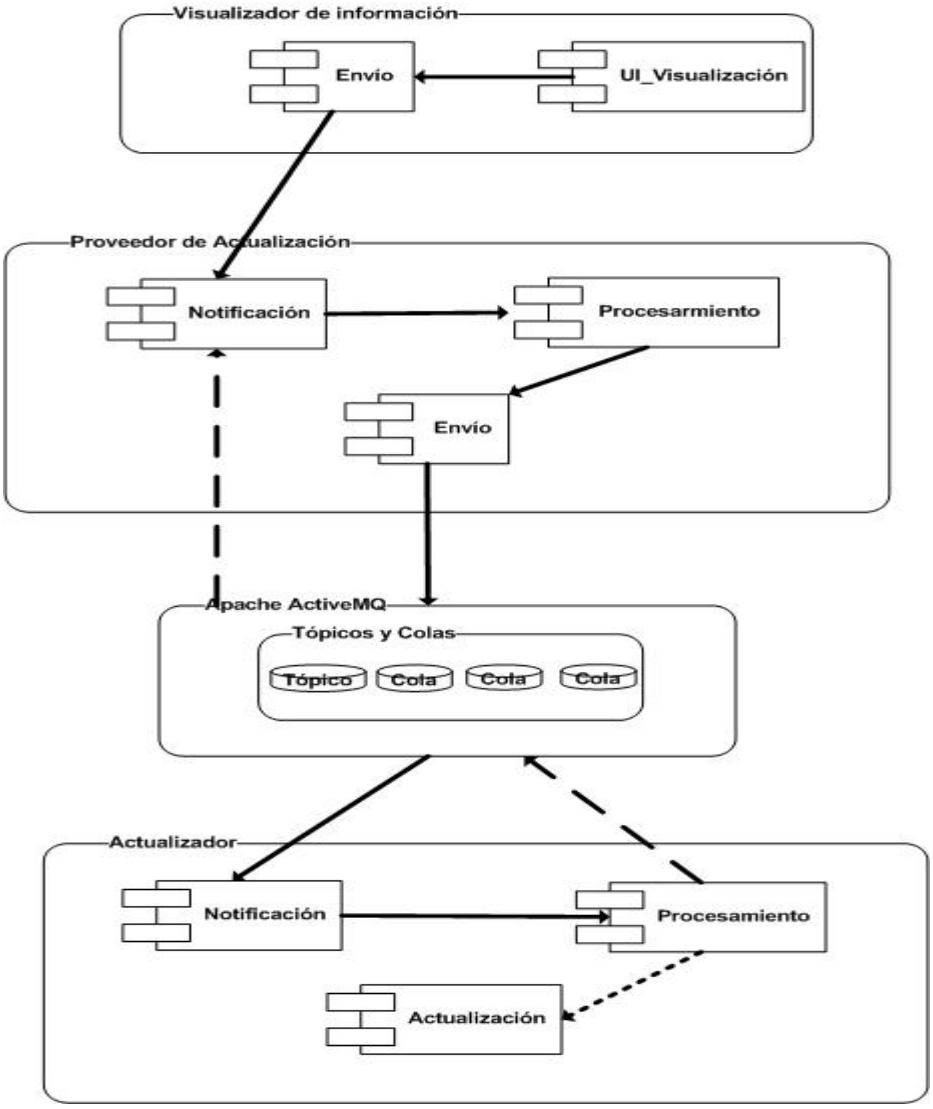

Figura 17 - Comunicación entre de las aplicaciones

Este módulo envía los datos de actualización al módulo *Procesamiento* para que se genere un archivo XML (conformado por el nombre de todos los archivos y carpetas que se encuentran dentro del archivo WAR) donde se describe cada fichero que contiene la actualización y este la mande al módulo *Envío* donde este la manda al Actualizador a través del servidor de mensajería. El servidor de mensajería guarda el mensaje localmente antes de enviarlo para prevenir fallos tales como; caída del servidor por falta de corriente, errores inesperados en la maquina virtual de java, etc. Seguidamente el servidor verifica que el cliente esté disponible y si es así le entrega el mensaje, que en este caso es la notificación con la descripción de la actualización. Si el cliente no está disponible, lo cual puede ocurrir

por causa de que la red no esté disponible u otras razones, el servidor sigue intentando entregar el mensaje hasta que el cliente esté disponible.

Cuando el Actualizador recibe la notificación lo hace por medio del módulo *Notificación,* aquí, éste manda la descripción recibida al módulo *Procesamiento* para que éste con tal información y la del sistema local genere un pedido de actualización el cual se manda al Proveedor de Actualización mediante el servidor de mensajería. Lo que realmente ocurre aquí es que éste módulo, también genera un XML con la misma estructura y el mismo objetivo que el XML recibido; describir todo lo que contiene un archivo war, para entonces con estos dos archivos XML, compararlos y la diferencia entre los dos deberá ser lo que se necesita actualizar. Esto podría parecer innecesario, pero dado el objetivo del sistema en uso, que servirá para automatizar los centros penitenciarios de Venezuela, se deriva que deben tomarse todas las medidas de seguridad posible, y una de ellas es asegurarse de que se mantenga la integridad del sistema que se está usando, algo que podría violarse si algún intruso o algún hacker, modificara o borrara algún archivo del software instalado, por lo tanto la actualización debe hacerse a pedido de cada Actualizador.

La petición la recibe el módulo *Notificación* del Proveedor el cual se la pasa a Procesamiento para que éste con el XML que contiene la descripción del pedido genere un fichero compactado que contiene la actualización que necesita cada cliente y la pase a *Envío* para que aquí se mande al Actualizador mediante el servidor de mensajería , en este caso el servidor de mensajería no guarda los mensajes puesto que el archivo de actualización puede ser grande y esto podría causar un gran consumo de espacio en disco y lentitud en el proceso puesto que el servidor tendría que guardar todos los archivos de actualización para todos los actualizadores. *Notificación* recibe los archivos con la actualización y se la pasa a *Procesamiento* para que éste verifique que los datos son correctos y la mande al Módulo *Actualización* para que finalmente se haga la actualización sobre el sistema desplegado.

Para hacer este paso final primero se para el sistema que se encuentra ejecutándose, se copian los nuevos archivos o se borran algunos existentes y se inicia nuevamente el sistema. Se debe tener en cuenta que en medio de cualquiera de estas operaciones puede ocurrir algún error inesperado por lo que el Actualizador antes de copiar o borrar archivos de la actualización primero hace una copia del sistema que se encuentra usándose para si ocurre algún fallo, asegurarse que el sistema quede en su última versión y no quede un sistema inestable e inservible.

3.2.3 Actualización del mecanismo

Una característica importante que el mecanismo propuesto debe cumplir, es la posibilidad de actualizarse él mismo. Con el Proveedor de Actualización y el Actualizador expuestos anteriormente se consigue actualizar el sistema que se encuentra en uso dando la posibilidad de corregir errores que se hayan cometido durante el desarrollo del software. Como hemos visto anteriormente el sistema estará distribuido por toda la geografía venezolana, de aquí que el mecanismo deberá ser remoto y centralizado, y esta es unas de las premisas del mecanismo de actualización. Para garantizar lo anterior completamente, es necesario poder actualizar también al propio mecanismo, ya que ningún software está exento de errores y de fallos; éstos solo se pueden reducir en la misma medida en que se vaya probando.

Para hacer que el propio mecanismo se actualice se ha utilizado la herramienta Java Service Wrapper. JSW proporciona una manera en que las aplicaciones hechas en Java puedan ser reiniciadas y así activarse la nueva actualización. Para que la aplicación pueda ser actualizada esta herramienta actúa como una fachada para ésta, y al actualizarse la aplicación el Wrapper es capaz de pararla y volver a ejecutarla y de esta forma poner en uso la nueva versión. En el caso en que se vaya a actualizar el mecanismo (por ejemplo, el actualizador), esto es muy útil puesto que se logra automatizar el proceso de actualización del mecanismo. Esta herramienta permite instalar los programas escritos en el lenguaje java como servicios nativos de Windows o demonios de Unix y sus sistemas derivados como GNU/Linux, permitiendo controlarlos con las herramientas del sistema operativo[19].

Una ventaja de esta herramienta sobre otras parecidas es su flexible configuración. JSW posee una gran cantidad de opciones de configuración que permiten configurar la maquina virtual de java en cualquier forma, además de permitir configurar cosas como la escrituras de trazas y la forma en que JSW será instalado como un servicio. Las opciones de la escrituras de trazas permiten ver errores producidos en la aplicación que corre como un servicio y que JSW captura para guardarlos y analizarlos posteriormente. JSW también tiene un alto nivel de confiabilidad ya que es capaz de detectar errores en la maquina virtual o en el software instalado y de reiniciar este último para su correcto funcionamiento[19].

3.2.4 Beneficios de la mensajería

En la implementación del mecanismo propuesto se ha utilizado la mensajería como vía para intercambiar los datos entre las aplicaciones. La mensajería además de permitir a estas aplicaciones intercambiar información resuelve otra serie de problemas derivados de las características del

mecanismo propuesto (véase Cáp. 2 epígrafe 3). A continuación se verán algunas características del mecanismo y cómo son resueltas con la mensajería:

- Comunicación remota: La mensajería posibilita que aplicaciones separadas se puedan comunicar y transferir datos. No importa lo distante que estén una de otra, el sistema de mensajería siempre se asegura que los mensajes lleguen a su destino. También las aplicaciones se pueden comunicar con el sistema de mensajería haciendo uso de varios protocolos de comunicación, como, TCP, SSL, NIO, UDP, etc. Esto es muy importante puesto que los clientes y el Proveedor de Actualización pueden estar en extremos opuestos de Venezuela, lo que hace que estén bastante distantes uno del otro.
- Comunicación asíncrona: La mensajería posibilita una estrategia enviar-y-olvidar a la comunicación. El emisor no tiene que esperar por el receptor a que reciba y procese el mensaje; incluso no tiene que esperar a que el sistema de mensajería entregue el mensaje. El emisor solo tiene que esperar a que el mensaje sea enviado; a que el mensaje sea almacenado satisfactoriamente en el canal por el sistema de mensajería. Una vez el mensaje es almacenado, el emisor es libre para hacer otra tarea mientras el mensaje es transmitido en segundo plano. El receptor puede querer enviar un acuse de recibo o resultado de vuelta al emisor, lo cual requiere otro mensaje. Debido a que los ficheros de actualización pueden llegar a ser de un tamaño significativo este tipo de comunicación es muy valiosa porque así el proveedor no tiene que esperar a que el Actualizador reciba la actualización para continuar haciendo otras tareas.
- Comunicación Fiable: La mensajería proporciona la entrega confiable que una llamada a procedimiento remoto (RPC) no puede. La razón por la que la mensajería es mas confiable que RPC es que ésta usa una estrategia Almacena-y-reenvía para transmitir mensajes. Los datos son empaquetados como mensajes, los cuales son unidades atómicas independientes. Cuando el emisor envía un mensaje, el sistema de mensajería almacena el mensaje. El sistema entonces entrega el mensaje reenviándolo a la computadora del receptor, donde es almacenado nuevamente. Almacenar el mensaje en la computadora del emisor y en la computadora del receptor es asumido de una forma confiable (para hacerlo mas confiable, el mensaje puede ser almacenado en disco en lugar de en la memoria). Lo que sí no es confiable es enviar el mensaje desde la computadora del emisor a la computadora del receptor, debido a que el receptor o la red pueden no estar funcionando correctamente. El sistema de mensajería soluciona esto mediante el reenvío del mensaje hasta que sea exitoso. Esta forma automática de reintento posibilita al sistema de mensajería solucionar problemas con la red de tal forma

que el emisor y el receptor no tengan que preocuparse sobre estos detalles. Esta característica es muy valiosa ya que la aplicación, tanto el Proveedor como el Actualizador una vez que envíen un mensaje pueden estar seguros que será entregado a su destino sin tener que preocuparse de ello.

- Integración Plataforma/Lenguaje: Cuando se conectan múltiples sistemas de computadoras vía comunicación remota, estos sistemas comúnmente usan diferentes lenguajes, tecnologías y plataformas, quizás porque fueron desarrollados sobre el tiempo por diferentes equipos. Integrar tales tipos de aplicaciones divergentes puede requerir una zona desmilitarizada de middleware para negociar entre las aplicaciones, a menudo usando el menor denominador común, tal como datos planos con formato oscuro. En estas circunstancias un sistema de mensajería puede ser un traductor universal entre las aplicaciones que trabajan cada una con su lenguaje y plataforma en sus propios términos y aun así permite que todas se comuniquen a través de un paradigma común de mensajería. Esto puede ser útil, si por alguna razón se hace algún cliente en otro lenguaje entonces no se tendría que cambiar nada en el mecanismo.
- Manejo de Hilos: Comunicación asíncrona significa que una aplicación no tiene que bloquearse mientras espera por otra aplicación a que ejecute una tarea a menos que quiera esperar. En lugar de bloquear en la espera de una respuesta el emisor puede usar un callback que alertará al emisor cuando la respuesta llegue. Un largo número de hilos bloqueados por un período largo de tiempo puede ser problemático. Demasiados hilos bloqueados puede dejar a la aplicación con muy pocos hilos disponibles para ejecutar una real tarea. Si una aplicación con algún numero dinámico de hilos bloqueados se cae, cuando la aplicación reinicie y recupere su anterior estado, reestablecer esos hilos será difícil. Con callback los únicos hilos que se bloquean son mínimos, un conocido número de oyentes esperando por respuesta. Esto deja la mayor parte de lo hilos disponibles para otras tareas y define un conocido número de hilos oyentes que pueden ser fácilmente reestablecidos después de una caída. Esto también es una característica importante del mecanismo propuesto puesto que el proveedor estará enviando la actualización concurrentemente a varios clientes y si no se hiciera de forma concurrente el proceso de actualización se haría muy lento.
- Regulación de Flujo: Un problema con la llamada a procedimientos remotos es que demasiadas a un mismo receptor al mismo tiempo puede causar una sobrecarga en el receptor. Esto puede causar degradación en el rendimiento e incluso que el receptor deje de funcionar. La comunicación asíncrona posibilita al receptor controlar la velocidad a la cual consumir las peticiones, de tal forma que no se sobre cargue debido a demasiadas peticiones simultaneas.

El efecto adverso de esto en los emisores es minimizado debido a la comunicación asíncrona, ya que el emisor no está bloqueado esperando por el receptor. Esto es deseable en los actualizadores que estarán recibiendo la actualización y podrán controlar la velocidad con que reciben los mensajes, que de otra forma podrían colapsar. [18]

3.3 Conclusiones

El desarrollo de este capítulo ha permitido una mejor compresión del mecanismo de actualización que se propone en este trabajo y de cómo es viable su implementación real. Se ha podido ver cómo se han construido tres aplicaciones para implementar el mecanismo propuesto. Se ha mostrado que se ha construido un Proveedor de Actualización para notificar y enviar la actualización a los clientes, un Actualizador que se encargue de realizar la actualización en el sistema en uso y un Visualizador de Información que muestre una interfaz amigable al usuario para que pueda interactuar con el sistema y que muestre información sobre el estado de la actualización. Se ha mostrado cómo se comunican estas aplicaciones y cómo se han solucionado los diversos problemas que acarrean las características intrínsecas del mecanismo.

Conclusiones

Como resultado del trabajo realizado se concibió, para el Sistema Penitenciario Venezolano, la creación de un mecanismo de actualización de software. Dicho mecanismo constituirá un sistema de soporte al mantenimiento de la solución de software que se implante.

Con la implantación del mecanismo de actualización propuesto se obtiene los siguientes beneficios:

- La actualización de software se realiza de manera remota, es decir desde la dirección central se actualiza todos los centros penitenciarios.
- Se gestiona el estado de las actualizaciones en cada uno de los centros penitenciarios, lo que permitirá tomar medidas a tiempo en caso de fallos inesperados.
- El tiempo utilizado para dicho proceso disminuye considerablemente comparado con el método que se utiliza actualmente de actualización manual.
- El personal técnico necesario para actualizar el sistema informático es mínimo dado las características del mecanismo.

Por lo que se puede arribar a la conclusión de que el objetivo principal de este trabajo se ha materializado con la realización de las tareas investigativas planteadas.

Recomendaciones

- Aplicar los conocimientos adquiridos durante el transcurso de esta investigación en otros trabajos enmarcados en la misma línea investigativa.
- Estudiar la viabilidad de aplicar el mecanismo propuesto en otros proyectos de la Universidad.
- Agregar funcionalidad al mecanismo propuesto, así como perfeccionar y probar en ambientes reales para garantizar que la implementación cumpla con los parámetros de rendimiento deseados.

Bibliografía

1. Richard S. Hall, A.C., Alfonso Fuggetta, Dennis Heimbigner, André van der Hoek, Alexander L. Wolf (1998) *A Characterization Framework for Software Deployment Technologies*. **Volume**,
2. Richard S. Hall, D.H., Alexander L. Wolf, *A Cooperative Approach to Support Software Deployment Using the Software Dock.* 1998.
3. Ajmani, S., *Automatic Software Upgrades for Distributed Systems*, in *Department of Electrical Engineering and Computer Science*. 2004, MASSACHUSETTS INSTITUTE OF TECHNOLOGY.
4. Remy Jansen, S.B., G. Ballintijn (2004) *A process framework and typology for software product updaters*. **Volume**, 10
5. Ajmani, S. (2004) *A Review of Software Upgrade Techniques for Distributed Systems*. **Volume**, 19
6. Michael Hicks, J.T.M., Scott Nettles, *Dynamic Software Updating.* 2002.
7. A.Tridgell, *Efficient algorithms for sorting and synchronization*. 1999.
8. E.Dolstra (2003) *Integrating software construction and software deployment*. **Volume**, 102-117
9. H. E. Harrison, S.P.S., and T. S. Yoo, *Rtools: Tools for software management in a distributed computing environment.* 1988: p. 85-93.
10. Parihar, M., *ASP.NET*, A. MULTIMEDIA, Editor. 2002.
11. Paul Slater, R.H., Jason Hogg, *Deploying .NET Applications- A Lifecycle Guide*, M. Corporation, Editor. 2003.
12. Corporation, M. *Updater Application Block–Version 2.0*. 2005 [cited; Available from: http://msdn2.microsoft.com/en-us/library/ms978574.aspx.
13. Marinilli, M., *Java Deployment with JNLP and Webstart*. 2001: Sams Publishing. 512.
14. Rod Johnson, J.H., Alef Arendsen, Thomas Risberg,Colin Sampaleanu, *Professional Java Development with the Spring Framework*. 2005.
15. ASF. *ActiveMQ*. 2007 [cited; Available from: http://activemq.apache.org/.
16. W3C. *HTTP*. 2007 [cited; Available from: http://www.w3.org/Protocols/.
17. Consulting, M.B. *Jetty web server.* 2007 [cited; Available from: http://www.mortbay.org/.
18. Hohpe, G., *Enterprise Integration Patterns*. 2003.
19. Software, T. *Java Service Wrapper*. 2007 [cited; Available from: http://wrapper.tanukisoftware.org/doc/english/introduction.html.