Universidad Nacional de San Luis
Facultad de Ciencias Físico-Matemáticas y Naturales
Departamento de Informática

Una arquitectura de software para el desarrollo remoto de software

Sebastián Matías Gun

Asesor: ***Dr. Daniel Riesco***
Co-asesor: Ms. Ariel Leiva

Trabajo final para optar al grado de
Licenciado en Ciencias de la Computación

San Luis - Argentina
2015

A mi familia por la paciencia.

A mi novia por el empuje.

Una arquitectura de software para el desarrollo remoto de software

Índice general

Índice de figuras

Índice de tablas

1 - Introducción

Durante muchos años se han dedicado esfuerzos para encontrar procesos y metodologías, que sean sistemáticos, predecibles y repetibles, con el fin de mejorar la productividad en el desarrollo y la calidad del software producido. La ingeniería de software ofrece métodos y técnicas para desarrollar y mantener software de calidad; abordando todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistema de información y aplicables a infinidad de áreas [Somm07, Press05]. Todo el proceso de producción de software puede beneficiarse con la aplicación de las técnicas de la ingeniería de software; desde etapas tempranas como el análisis de requerimientos o la definición de la arquitectura, pasando por la implementación y documentación, hasta las fases finales como las pruebas y el mantenimiento. Las tecnologías CASE (Computer Aided Software Engineering, ingeniería de software asistida por computadora) comprenden diversas herramientas destinadas a mejorar la efectividad de la producción de software y reducir el costo del proceso; brindando asistencia en todos los aspectos del ciclo de vida del desarrollo de software [Kuhn89, LouKa95]. Aquellas herramientas que soportan varias etapas del proceso, se denominan I-CASE o CASE integrado.

Desafortunadamente es poco usual que una sola herramienta CASE integrada baste para dar asistencia en todo el proceso de cualquier grupo de trabajo. Muchas veces debido a la heterogeneidad de los ambientes de desarrollo y a la ubicación distante de las partes de los sistemas de software; surge la necesidad de la construcción de herramientas CASE que se encarguen de mediar entre las partes del sistema, brindando una estación de trabajo centralizada y transparente al desarrollador.

En la actualidad, muchas empresas de desarrollo de software o de tecnología cuentan con recursos e infraestructura distantes geográficamente. Esto puede deberse a la falta de disponibilidad local por el costo que esto insume, porque no se cuenta con la tecnología apropiada o simplemente por seguridad al necesitar disponer de ciertos recursos críticos centralizados. Generalmente, los desarrolladores deben acceder diariamente a estos recursos lejanos como parte de su labor; por ejemplo una terminal remota en un centro de datos (datacenter), hacer consultas a una base de datos monolítica remota, almacenar sesiones en un sistema de archivos en un servidor lejano, etc. Pese al gran avance en las comunicaciones, trabajar con grandes conjuntos de datos remotos puede volverse tedioso, debido a la latencia inherente de las redes actuales. El tiempo de espera total para poder disponer localmente de los datos que se encuentran en los recursos distantes, puede ser lo suficientemente significativo como para demorar el proceso de desarrollo e incluso frustrar al desarrollador.

Es así que, con frecuencia, el desarrollo de grandes sistemas de software involucra el uso de diversas herramientas y tecnologías, con las que el desarrollador debe familiarizarse:

distintos sistemas de cómputo, lenguajes, configuraciones y ejecutar comandos en diversos entornos. Esto requiere que el conocimiento del desarrollador sea basto en todas estas áreas y que usualmente crezca junto con cada nueva herramienta o sistema que deba ejecutarse o formar parte del proceso de producción de software. Es decir que demanda de aprendizaje continuo y un nivel de conocimientos cada vez más elevado.

1.1 - Motivación

Es desafiante formar parte de grandes equipos de desarrollo; trabajar en conjunto desarrollando aplicativos (programas informáticos o sitios web para el caso de estudio de este trabajo) que sean usados por cientos o miles de usuarios finales, sentir que lo que uno desarrolla llega a influir en la vida y hasta en la rutina de muchas personas.
Quienes estudiamos ciencias de la computación y aplicamos cada día nuestros conocimientos a nuestra labor diaria, descubrimos día a día nuevas tecnologías y nuevas maneras de hacer las cosas, para enfrentar cada uno de los problemas que debamos resolver. Es enorme la cantidad de herramientas disponibles, tecnologías, aplicaciones, lenguajes de programación, etc, con los que un desarrollador de software (o cualquier otro profesional de la informática que integre equipos de desarrollo de software) se enfrenta diariamente, y estos números sólo van en aumento con el correr del tiempo.
Lo que comienza siendo una pequeña colección de aplicaciones, diccionario de pasos para ejecutar determinadas tareas, algunos pocos recursos (físicos o digitales) con los que uno desarrolla sus actividades; con el correr del tiempo se torna en una enorme cantidad de herramientas, mayores dependencias (algunas posiblemente bloqueantes para el éxito en una tarea) y un conocimiento requerido cada vez más y más grande.

Durante el transcurso de este trabajo, nos enfrentamos al problema de desarrollar y mantener módulos para una aplicación web masiva y compleja, de alta disponibilidad y que debe soportar un importante nivel de tráfico de visitantes diariamente. Tras bastidores, esta aplicación cuenta con módulos escritos en distintos lenguajes de programación; una herramienta CASE principal para la edición de código fuente y muchos aplicativos distintos para administrar estos y otros archivos; una base de datos relacional y monolítica donde se ejecutan la mayoría de las consultas provenientes del sitio web (usuarios finales) y herramientas administrativas internas (herramientas de backoffice, utilizadas por empleados o usuarios internos). El capital humano también es grande, contando con cientos de desarrolladores ubicados en múltiples centros geográficos.
Lo antes expuesto, es un problema general de muchas compañías de desarrollo de software. En este trabajo se trata el caso de estudio específico de MercadoLibre (empresa líder en el comercio electrónico de Latinoamérica), pero lo que se resuelve es el problema en general (se investiga, implementa y ofrece una solución aplicable a cualquier compañía).

Así, cada desarrollador posee sólo una pequeña fracción de todas sus herramientas de

desarrollo en su entorno de desarrollo local (workstation): los aplicativos de edición de código fuente y demás herramientas CASE, y un servidor web montado en su misma estación de trabajo. Sin embargo, los recursos compartidos por todo el equipo de desarrollo (como se mencionó, disperso en diferentes puntos geográficos (Buenos Aires, Argentina; San Luis, Argentina; Córdoba, Argentina; São Paulo, Brasil; Montevideo, Uruguay) pueden encontrarse en otra ubicación remota (la mayoría de estos en Buenos Aires para el caso de estudio). Estos son: bases de datos (instancia principal de desarrollo ubicada en Buenos Aires, existe una copia en San Luis desactualizada en cuanto a esquemas, tablas y datos, y sin sincronización alguna entre ambas), sistemas de archivos (unidades de red compartidas, una montada en Buenos Aires y otra en San Luis, la última posee un subconjunto de los archivos de aplicación, configuraciones, etc. requeridos por el aplicativo web; nuevamente no hay sincronización alguna entre ambos recursos), repositorios de código fuente (una sola copia del repositorio de tipo Subversion/SVN desplegada en Buenos Aires), etc. La mayoría de estos recursos son centralizados o existen una o pocas instancias para toda la compañía, no por capricho o porque no exista el capital, tecnología o recursos humanos para replicarlos, sino por diversos motivos de administración, seguridad y aseguramiento de la calidad; que impiden replicarlos o bien reducirían la performance del proceso de desarrollo en cuanto a tiempos de entrega, o pruebas en múltiples ambientes.

Bajo esta arquitectura, el entorno de desarrollo local de la oficina de San Luis responde muy lento. Uno de los puntos claves es el acceso al sitio web de la aplicación, donde se realizan las pruebas y se chequean los resultados. Los administradores y los jobs (procesos programados) también son recursos de uso frecuente, afectados por este problema. Para tomar conciencia del problema y plasmarlo en números que empujen la necesidad de la iniciativa de cambio, se realizaron testeos de tiempos de ejecución de algunos de los componentes de acceso más frecuente del sistema, bajo las condiciones actuales y bajo condiciones deseadas (ideales, o al menos, mejores). En el anexo 1 se detallan los tiempos de acceso a archivos de sesión ubicados en unidades de red remotas, se evidencia el problema y se analizan alternativas concretas para mejorar los tiempos (una de las cuales se trabajará luego). Similar en el anexo 2, se realiza una prueba de tiempos de acceso a la base de datos de desarrollo remota, una comparativa accediendo a una instancia local, y se analiza nuevamente qué cambios son necesarios realizar para optimizar el entorno de desarrollo.

Cuando hay muchos factores involucrados, muchas cosas pueden salir mal, y este panorama no es la excepción a la regla. En nuestro caso, además de todas las herramientas de software con las que tiene que interactuar el desarrollador en su estación de trabajo, se suma la complejidad de requerir el acceso a recursos centralizados ubicados en centros de desarrollo distantes. El desarrollador dispone de un servidor web montado en su propia terminal de trabajo, pero accediendo a una base de datos física ubicada en otra provincia y requiriendo la descarga de archivos estáticos de la aplicación web (imágenes por ejemplo) y archivos de configuraciones de unidades de red, también montadas en la otra locación. Debido a la latencia inherente de las redes actuales y el gran volumen de datos que debe

transmitirse entre estas ubicaciones distantes para interactuar con el aplicativo durante su fase de desarrollo, esta situación genera grandes tiempos de espera y pérdida de productividad, al requerir que el desarrollador deba esperar una cantidad significativa de tiempo para ver el impacto de los cambios que realiza en el código, en su ambiente de pruebas. Esto puede llevar a malas prácticas de desarrollo, como por ejemplo escribir grandes cantidades de código sin pruebas escalonadas o intermedias durante su escritura, porque cada intervalo requeriría el reinicio del servicio en la terminal del desarrollador con el tiempo que infiere, y el desarrollador podría no estar a gusto con esta situación.
Por otro lado, en este caso de estudio, la aplicación web productiva es ejecutada bajo una arquitectura de servidores Linux, mientras que la herramienta CASE y manejadores de versiones de código fuente del desarrollador, ejecutan bajo sistemas Windows. Esta heterogeneidad entre el sistema productivo real y el entorno de trabajo local al desarrollador, es muy común entre las compañías de tecnología, y genera a su vez que deban realizarse configuraciones y pruebas duplicadas pero atentas a la plataforma de cada caso, generando más pérdida de tiempo, mayores conocimientos requeridos y posible omisión de configuraciones importantes por no poder reproducir localmente el mismo stack de tecnología utilizado en el entorno productivo.

La combinación de las diferentes herramientas requeridas para el desarrollo en un ambiente como el descripto anteriormente, como así también el flujo de comunicación entre estas, genera demoras considerables e influye directamente en la productividad del equipo de trabajo y en todo el proceso de desarrollo de software; dando lugar a posible frustración del desarrollador al requerir mucho más tiempo para llevar a cabo sus tareas, mayor posibilidad de errores y mayores tiempos de entrega de las características a desarrollar en la aplicación. Es necesario que la mayor cantidad de estas herramientas se encuentren dispuestas en el mismo sitio (es decir, dentro de una misma red o incluso en un solo equipo, pero no dispersas entre varias redes), para que la comunicación entre ellas sea lo más eficiente posible y así poder evitar demoras que pudieran afectar el resultado final del proceso de desarrollo de software.

1.2 - Objetivos

Por todo lo mencionado en el apartado anterior, surge la necesidad entonces de disponer de un entorno más amigable, con menos dependencias, menos herramientas, y principalmente más veloz para el desarrollo; pero a su vez minimizando el riesgo de configuraciones erróneas durante la fase de desarrollo; requiriendo homogeneizar lo mayor posible ambos entornos de ejecución, de desarrollo y de producción, mientras se busca optimizar la velocidad del proceso de desarrollo en general.

Para esto se migran la mayor cantidad posible de recursos, a la ubicación donde residen aquellos más lentos, con mayor cantidad de accesos y que generan la mayoría de las

demoras en la fase de desarrollo de software (base de datos de desarrollo y unidades de red con archivos compartidos). De este modo se optimiza la comunicación entre estos recursos, a la vez que durante el desarrollo de este trabajo, se brinda una interfaz amigable al desarrollador para interactuar con este nuevo entorno, integrada en la misma herramienta CASE que ya conoce. De esta forma el desarrollador solo se limita a la edición de código fuente en su entorno local de trabajo, sincroniza su copia local (modificada) con la de la locación remota (pequeña transferencia de datos) mediante un proceso de transmisión incremental diseñado para tal fin (este proceso será explicado más adelante) y despliega la aplicación modificada en el entorno remoto, con todos los recursos involucrados locales a la aplicación, evitando demoras entre estos; mientras que la visualización de los resultados se realiza nuevamente en el entorno local, con otra diminuta transferencia de datos.

Puntualmente, el objetivo a alcanzar, es una herramienta de mejora productiva en cuanto a conectividad y herramientas de desarrollo. La herramienta optimiza la transferencia de datos requerida entre las distintas fuentes de recursos de la aplicación en el entorno de desarrollo, para que el desarrollador pueda iniciar los servicios en su terminal más velozmente (servidor web y conexión a base de datos por ejemplo), y a su vez, pueda aplicar los cambios necesarios en el software y visualizar los resultados de estos de una forma más ágil también. El acceso a los recursos remotos y/o centralizados (base de datos de desarrollo, repositorios de código fuente, unidades de red con archivos de configuración) debe ser a su vez, más veloz también, para que tanto el desarrollador como los aplicativos en sí, puedan ejecutar más velozmente, simulando un entorno con todas las piezas requeridas siendo ejecutadas de forma local.

Todos los aplicativos y herramientas que conforman el nuevo stack de tecnología requerido para montar esta nueva arquitectura de software mejorada, deben poder pre-instalarse en cada uno de los terminales de desarrollo de una forma cómoda y que no implique realizar cada uno de los pasos (instalar y configurar cada pieza de software) por separado en cada terminal.
Estas nuevas herramientas no deben estorbar en el entorno usado por el desarrollador, sino que deben proveer transparentemente los servicios necesarios para la optimización requerida, sin que el desarrollador deba interactuar o notar siquiera la presencia de ellas.
Otra característica importante es que la herramienta mejorada emula de una forma más fiel el entorno productivo de ejecución del software, en la misma terminal del desarrollador, sin la necesidad de que este último deba migrar a un nuevo sistema operativo o nuevas herramientas, sino con la adhesión de pequeñas utilidades en la misma herramienta CASE que actualmente usa para su labor diaria; brindando acceso a los servicios en el nuevo stack de tecnología con una interfaz sencilla y amigable, en la misma aplicación de software que ya conoce y usa diariamente. La ganancia de esto (además de lo mencionado en cuanto a conectividad), resulta en un entorno más parecido al productivo, minimizando en gran medida la posibilidad de que surjan errores de desarrollo en etapas de despliegue de software, ya que al contar ahora con un entorno más símil al productivo, los errores de configuración pertinentes a la arquitectura sobre la que ejecuta el software, deben aparecer durante la fase

de desarrollo del ciclo de desarrollo de software, y no durante la fase de pruebas del mismo en el entorno productivo.

Los cambios realizados en la arquitectura de la estación de trabajo del desarrollador y de la herramienta CASE utilizada, resultan entonces en la herramienta de mejora productiva requerida, mejorando la velocidad de todo el proceso de desarrollo, incluyendo la mejora en los tiempos de entrega de las piezas de software. Resulta a su vez, en una configuración inicial de cada nuevo terminal de trabajo mucho más ágil, permitiendo a los nuevos integrantes del equipo de desarrollo volverse productivos más rápido aún y requiriendo menos conocimientos iniciales; mientras que se minimizan los errores de configuración del nuevo software desarrollado, o bien se trasladan a fases más tempranas de desarrollo, en donde es más fácil encontrarlos y repararlos; sin siquiera mencionar que el riesgo de que esta clase de errores ocurra es casi nulo en el entorno de desarrollo, mientras que pueden inferir un alto costo si surgen en etapas finales del ciclo de desarrollo, en los entornos de prueba o productivos. Estos errores, por ejemplo, podrían implicar sucesivos armados de integración de nuevas características de software para su despliegue (builds), hasta dar con la configuración adecuada para que todas las nuevas piezas de software interactúen correctamente en la nueva versión del producto; generando demoras en la entrega de la nueva versión o hasta freezado temporal de integración de nuevas características.

1.3 - Antecedentes

Lo mencionado en los puntos anteriores constituye un problema que afecta a muchas empresas y/o equipos de desarrollo. Distintos equipos ven el mismo problema desde diferentes puntos de vista y por lo tanto es muy difícil que haya una única solución que sea aplicable a todos los casos. Por tales motivos existen varios proyectos en curso (como los mencionados a continuación) que tratan de solucionar de la mejor forma lo planteado.
Si bien estas soluciones alternativas son tentadoras, sólo se limitan a facilitar el acceso a recursos remotos. Crean copias locales de archivos remotos al momento de ser requeridos por el usuario y reflejan los cambios que se hagan a estos en el sistema remoto. No permiten integrar toda una gama de herramientas y partes de un sistema de software como el que se requiere. Además el intercambio de datos para mantener las copias consistentes es elevado, ya que copian archivos completos de un extremo a otro de la arquitectura global. Son proyectos muy ambiciosos que, por el momento, soportan compilación remota sólo para C/C++ y que para trabajar con otros recursos no locales a través de Eclipse File System necesitan de una red rápida y que tenga baja latencia. Esto, en el caso de aplicación de este trabajo, no está disponible y el sistema está basado en Java, por lo tanto no es factible usar estas posibles soluciones. De todas formas se hace mención de algunas de estas herramientas para mostrar otras perspectivas que abordan la problemática y las soluciones que desarrollaron para los casos de uso que aplican.

“**Remote System Explorer**” (RSE, Explorador de sistema remoto) [EDocRSE] es una parte

del proyecto "Target Management" de Eclipse que busca integrar sistemas de cómputo heterogéneos bajo una interfaz unificada de desarrollo; suministrando los servicios subyacentes de comunicación y configuración de sistemas remotos, sus conexiones y servicios; junto con un conjunto de aplicaciones útiles al desarrollador, como explorador de archivos (con posibilidad de edición, búsqueda y comparación de contenidos), intérprete de comandos (para lanzar una compilación remota por ejemplo, o iniciar servicios) y administrador de procesos; para el manejo transparente del equipo remoto. Además, desarrolladores terceros pueden extender la herramienta para integrar soporte para casi cualquier clase de sistema o protocolo, a través de un framework extensible de plugins diseñado para tal fin; o proveer nuevas utilidades sobre los protocolos implementados. Algunas que ya fueron implementadas de forma extraoficial son, por ejemplo, debuggers y visualización remota de escritorio (remote VNC).

Se probó esta herramienta en el marco de este trabajo, pero la misma simplemente facilita el acceso y la manipulación de los recursos remotos, siempre en la computadora remota. Bajo este panorama y para las necesidades de la aplicación concreta sobre la que se trabajó, el entorno sigue siendo lento, ya que no basta con simplemente dar acceso a los recursos; al estar remotos la latencia sigue siendo significativa; no permite trabajar con un subconjunto de los datos de forma local, de modo de acelerar las labores de desarrollo (tan sólo expone localmente los recursos remotos).

"**Parallel Tools Platform**" (PTP, Plataforma de herramientas paralelas) [EPTP] es otra suite de herramientas que otorga acceso remoto a recursos, y los presenta al usuario como si fueran locales. Brinda acceso a sistemas de archivos, compiladores, linkeadores, sistemas de consultas, sistemas operativos e incluso al hardware subyacente; todos remotos. Combina funcionalidad existente dentro de la plataforma de Eclipse (como las herramientas de desarrollo para C/C++) con nuevos servicios diseñados específicamente para proveer una interfaz transparente con los sistemas de cómputo paralelos, además de proveer soporte para lenguajes de programación de ámbito científico (como Fortran). PTP está específicamente orientado en lograr una mayor eficiencia en el desarrollo de aplicaciones paralelas y se focaliza en simplificar la interacción del usuario con los sistemas paralelos; soportando la integración de un gran rango de utilidades de éste ámbito (arquitecturas, sistemas y debuggers paralelos por ejemplo) [Watson05].

Si bien la aplicación web tomada como marco de referencia en este trabajo es altamente concurrente, no está diseñada de modo paralelo (ni tampoco es la intención migrarla a este modelo de cómputo). Por ende este conjunto de herramientas no se torna efectivo para el caso de implementación concreto de este trabajo, y debe ser descartada como solución.

"**Remote Development Tools**" (RDT, "Herramientas de desarrollo remoto") [EDocRDT] es un conjunto de herramientas distribuidas como parte de Eclipse PTP (herramienta presentada anteriormente), y provee un framework, y una implementación de referencia para dicho framework, que facilita el uso de un IDE local, para realizar tareas de desarrollo en otra

máquina remota.

La motivación principal detrás de este conjunto de herramientas, es el desarrollo de software sobre sobre computadoras científicas de alta performance, o bien sobre computadoras de tipo “mainframe” (computadora grande, potente y costosa; usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos; por ejemplo, para el procesamiento de transacciones bancarias). Las características propias de desarrollar software para este tipo de máquinas, que requieren especialmente de un ambiente de desarrollo remoto, son varias [EDocRDT]:

- Las máquinas en sí usualmente se encuentran centralizadas en una ubicación remota, y no pueden ser accedidas físicamente por sus usuarios.
- La experiencia de uso de las herramientas de un IDE como Eclipse, tiende a ser mucho mejor y más gratificante, que aquellas que ejecutan directamente sobre un intérprete de línea de comandos.
- La performance de mostrar una herramienta rica gráficamente como Eclipse, sobre un protocolo de gráficos remoto (como X11 por ejemplo), suele resultar muy lenta, especialmente para conexiones con alta latencia, que se encuentran geográficamente distribuidas.
- Los servidores pueden no poseer ninguna capacidad gráfica nativa, con la cual mostrar herramientas gráficas de desarrollo de software.

El concepto general detrás de RDT, es utilizar el IDE Eclipse como un cliente gráfico liviano, y delegar la mayor parte del procesamiento a los servicios ejecutando en la máquina remota. A través de esto, puede obtenerse un conjunto rico de herramientas de desarrollo, mientras que se mantiene un aceptable nivel de performance comparado a ejecutar un IDE completo localmente (en la máquina remota) [EDocRDT].
RDT provee inicialmente servicios para desarrollo remoto con C/C++, pero su framework está diseñado para soportar cualquier lenguaje arbitrario, a través del desarrollo de extensiones.

Si bien hubiera resultado como una posible elección este framework, realizando la correspondiente extensión para soportar herramientas de desarrollo sobre la plataforma de Java; sólo se limita a “conectar” el IDE Eclipse con dichas herramientas de desarrollo remotas (compilador, depurador, etc.), no permitiendo una configuración más extensa acerca de los recursos que son necesarios transferir entre una máquina y otra, y la forma en que deben transmitirse dichos recursos (a fines de optimizar la transferencia). Además delegando la totalidad de la responsabilidad de la comunicación con la máquina virtual a Eclipse, se pierden importantes oportunidades de desarrollo de más herramientas para interactuar con los servicios remotos, como aquellas que se desarrollaron durante la implementación de este trabajo, y que se presentan oportunamente en los capítulos correspondientes.

Otro enfoque totalmente distinto para atacar el problema, lo constituye **Oracle TimesTen** (TT) [OraTT]. Esta es una base de datos relacional optimizada para ser alocada en la memoria principal de una computadora (RAM), en vez de situarse en memoria secundaria

(disco rígido) como es lo habitual. De esta forma se orienta a minimizar los tiempos de respuesta de las aplicaciones que demanden una gran carga de operaciones o consultas de bases de datos y maximizar el throughput de las mismas (“throughput” refiere al volumen de trabajo o de información neto, que fluye a través de un sistema). TT puede funcionar simplemente como una caché en memoria de la base de datos real de la aplicación, o bien ser populada (poblada) con las tablas y registros necesarios para ser operada 100% en memoria a través de sentencias SQL; yendo a disco sólo para los volcados de cambios en la base de datos cada un determinado período de tiempo o bajo ciertas circunstancias predefinidas en la configuración de la herramienta (por ejemplo, una determinada cantidad de registros escritos).

Si bien inicialmente se vio como una opción tentadora a implementar, las primeras pruebas mostraron lo contrario. En el caso de la aplicación descrita, no resultó en la ganancia esperada por varias razones. La modalidad de caché es individual por cada servidor web, esto significa que TT debe instanciarse en cada terminal de trabajo con la demanda de memoria que eso requiere. Para lograr un beneficio en tiempos de consulta se deben hacer peticiones similares de forma reiterada, requiriendo las primeras hacer el fetch de los datos a la base de datos principal por cada usuario de la aplicación (en cada una de sus terminales); por lo que no optimiza el tiempo global, ni tampoco el startup (inicio) del entorno del usuario; sino que la mejora se comienza a notar luego de reiteradas operaciones a la base de datos.
Por otro lado, si queremos operar con sentencias de manipulación en la base en memoria, será necesario hacer una precarga de los datos junto con el startup de la aplicación. Esto, además de la demora inicial que supone, requiere una gran cantidad de memoria en la estación de trabajo del desarrollador si se quiere cargar toda la base de datos de pruebas o bien la mayoría de las entidades más significativas. Esto, para el caso de aplicación concreto de este trabajo, tampoco es posible debido al significativo tamaño de la base de datos de desarrollo, y la gran cantidad de esquemas, tablas, datos y demás recursos que almacena.
Sumado a estos inconvenientes, la configuración de la herramienta es bastante compleja para una tarea como la demandada para el entorno de desarrollo requerido, por lo que finalmente se descartó el uso de esta aplicación.

Por último, resulta oportuno hacer mención de **Vagrant** [Hashimoto13], que si bien no es precisamente un antecedente a este trabajo, ya que fue liberado posteriormente a la implementación del mismo (en paralelo con su redacción); bien merece la pena ser mencionado aquí, ya que se perfila como una de las mejores opciones actuales para la rápida y fácil puesta en marcha de ambientes de desarrollo de software.
Básicamente, Vagrant es una herramienta de software libre para la creación y configuración de entornos de desarrollo virtualizados [Palat12]. Puede ser visto como un wrapper (software que funciona como un adaptador entre un sistema y otro) entre software de virtualización como VirtualBox, KVM y VMware; y software de administración de configuraciones como Chef, Salt o Puppet.

Lo realmente interesante de Vagrant, es la velocidad con la que se puede montar un entorno

virtualizado en una máquina, con unos pocos pasos:

1. Bajar e instalar la distribución de Vagrant para el sistema operativo actual (Windows, Mac OS-X y varias distribuciones de Linux están soportadas).
2. Crear un simple archivo de texto plano (puede formar parte de un repositorio de código fuente de algún proyecto) para describir el tipo de máquina deseada, el software que se requiere instalar en ella, y el modo en que se querrá acceder a la misma.
3. Ejecutar una simple instrucción de línea de comandos (vagrant up) y esperar a que Vagrant baje e instale todo el software necesario para disponer de la máquina virtual deseada, completamente configurada de modo automatizado.

Dicho en otros términos, Vagrant elimina la clásica excusa de “en mi máquina funciona”, usada frecuentemente entre desarrolladores, cuando algo que originalmente funciona de una manera en el equipo del desarrollador que lo implementó, luego funciona de otra manera (o no funciona en absoluto) en otro equipo distinto. Esto se debe a que Vagrant puede crear entornos de desarrollo idénticos, para cada integrante del equipo.

Como se ve, es una solución tentadora y actualizada a los tiempos modernos; lamentablemente como se describió, es posterior al desarrollo de este trabajo, por lo que fue necesario realizar investigaciones por otras vías (que oportunamente se presentan en este informe) para desarrollar una solución al problema mencionado. De todos modos se presenta aquí debido al potencial que muestra para el mercado actual, y para mostrar que siguen desarrollándose soluciones profesionales, destinadas a atacar algunas de las problemáticas aquí expuestas.

2 - Herramientas CASE y patrones

Aquí se presentan todos los fundamentos teóricos que suscitaron durante la investigación e implementación de este trabajo. Asimismo, se condensa todo lo relacionado a lo que se ha escrito e investigado sobre el objeto de investigación de este trabajo. Este comprende la rama de la ingeniería de software ocupada del desarrollo de tecnologías CASE, y específicamente hablando de la aplicación particular de este trabajo, aquellas que constituyen a facilitar la virtualización de hardware mediante software de aplicación.

2.1 - Herramientas CASE

Las herramientas CASE (Computer Aided Software Engineering, ingeniería de software asistida por computadora) comprenden diversas herramientas destinadas a mejorar la efectividad de la producción de software y reducir el costo del proceso y el esfuerzo requerido; brindando asistencia a los desarrolladores e ingenieros de software en todos los aspectos del ciclo de vida del desarrollo de software (como análisis, diseño, codificación y pruebas), y en las actividades asociadas a los procesos de desarrollo de software (automatización de actividades de gestión de proyectos, gestión de los productos desarrollados, etc.) [Kuhn89, LouKa95]. Estas herramientas pueden proporcionar nuevas formas de observar la información obtenida en el proceso, llevando a tomar mejores decisiones y conseguir una mayor calidad en el producto generado; como así también elaborar resultados adicionales, automatizados y personalizados, que no serían fáciles de producir sin el soporte adecuado de estas herramientas [Press05]. Dicho de otra forma, ayudan a garantizar que la calidad se alcance antes de construir el producto mismo.

2.1.1 - Taxonomía

Para comprender mejor la amplitud de las herramientas CASE y para apreciar más fácilmente en qué procesos de desarrollo de software pueden aplicarse, se torna necesario contar con una taxonomía de herramientas. Una categorización que refleje áreas precisas para cada tipo de herramientas con las que el desarrollador pueda contar (una herramienta podría brindar asistencia en más de una área). Las herramientas CASE se pueden clasificar por su función, por su utilización en los distintos pasos del proceso de ingeniería de software, por la arquitectura del entorno (hardware y software) que les presta su apoyo, por su origen, etc. [Press05]. A continuación se mencionarán aquellas categorías de la taxonomía (utilizando como criterio principal la función) en las que se podría considerar que pertenece la herramienta desarrollada.

- Herramientas de software de sistema: CASE es una tecnología a ser aplicada en las estaciones de trabajo de los desarrolladores y demás actores implicados en el proceso de desarrollo de software. Por tanto, el entorno CASE deberá adaptarse a un software de sistema en red de alta calidad, al correo electrónico, aplicaciones de mensajería y a otras posibilidades de comunicarse. La herramienta propuesta se implementa de forma que pueda ser desplegada individualmente en la estación de trabajo de cada desarrollador, permitiendo la comunicación de dicho equipo con el ambiente remoto correspondiente en forma eficiente.
- Herramientas de gestión de configuración de software: La gestión de configuración de software (GCS) se encuentra en el núcleo de todos los entomos CASE. Estas herramientas pueden ofrecer su asistencia en varias tareas principales de configuración del software. La herramienta desarrollada puede agilizar procesos manuales de verificación de versiones y control de cambios; permitiendo desplegar rápidamente una u otra versión de la aplicación web, para probar los escenarios necesarios antes y después de un cambio significativo en el software.
- Herramientas de programación: Esta categoría de herramientas CASE abarca los compiladores, editores y debuggers (depuradores) disponibles para apoyar a la mayoría de los lenguajes de programación convencionales. Además, en esta categoría también residen los entornos de programación orientados a objetos, los lenguajes de cuarta generación, los entomos de programación gráfica, los generadores de aplicaciones y los lenguajes de consulta de bases de datos. Dado que la herramienta desarrollada se acopla al entorno de desarrollo integrado (IDE) usado por el equipo de IT (tecnologías de la información), y permite usar las capacidades de debug sobre instancias remotas de la aplicación o de procesos programados (jobs), también puede ser nombrada bajo esta categoría.
- Herramientas de pruebas cliente/servidor: El entorno cliente/servidor (C/S) requiere de herramientas especializadas que ejerciten los requisitos de comunicaciones en red para el cliente y el servidor. La arquitectura de la herramienta desarrollada permite alcanzar un diseño más similar al productivo en los ambientes de desarrollo y testeo; pudiendo desplegar la arquitectura del sistema de forma separada con un cliente en la estación de trabajo local (como un navegador web o una consola de comandos) y un servidor remoto en donde reside la aplicación web; en lugar de que ambas partes del sistema residan en la estación del desarrollador. Esto resulta en que el desarrollador puede llevar a cabo sus tareas en un entorno sustancialmente distinto al productivo, con el riesgo que eso supone en las pruebas (principalmente disponer de una comunicación real entre ambas entidades a través de conexiones de red).

2.1.2 - I-CASE

Los entornos CASE integrados (I-CASE) combinan varios de los beneficios individuales de las distintas clases de herramientas CASE (que abarcan actividades de ingeniería de software separadas) en una interfaz o herramienta unificada. Entre los beneficios del I-CASE

se incluyen [Press05]:

1. Una transferencia regular de información (modelos, programas, documentos, datos) entre una herramienta y otra, y entre un paso de la ingeniería y el siguiente.
2. Una reducción del esfuerzo necesario para efectuar actividades globales (que se practican durante todas las fases del ciclo de desarrollo de software), tales como la gestión de configuraciones de software, el control de calidad y la producción de documentos (documentación).
3. Un aumento del control del proyecto que se logra mediante una mejor planificación, monitorización y comunicación.
4. Una mejor coordinación entre los miembros del equipo de desarrollo que estén trabajando en grandes proyectos de software.

Para que estas características no resulten contraproducentes, las representaciones de la información de la ingeniería del software consecuentes, así como las interfaces entre las herramientas deben estar estandarizadas, y el mecanismo para la comunicación entre el ingeniero del software y todas sus herramientas debe ser homogéneo. Esto hará posible que I-CASE se desplace a lo largo de distintas plataformas de hardware y distintos sistemas operativos en donde requiera implementarse. Los distintos bloques de la herramienta I-CASE, como también de la arquitectura sobre la cual se sostiene (plataforma de hardware y sistema operativo) deberán acoplarse a través de un conjunto de servicios de portabilidad que ayuden a alcanzar los objetivos requeridos.

La fracción de software de la herramienta desarrollada se brinda como un plug-in (extensión) de la herramienta I-CASE Eclipse (IDE para la plataforma Java) usada por el equipo de desarrollo. La integración se logra entonces al disponer de la herramienta dentro del mismo entorno usado habitualmente por el desarrollador. Al estar diseñado sobre el mismo framework de desarrollo de extensiones de Eclipse, la integración es absoluta, con una interfaz idéntica del resto de las herramientas nativas del IDE. Además la comunicación entre la herramienta y el resto de la arquitectura se realiza a través de comandos y tecnologías abiertas y estandarizadas, a través de intérpretes de comandos y scripts diseñados para tal fin. Más adelante, en las secciones de tecnología se ahondará más en detalle en las cuestiones de implementación técnica de la herramienta.

De esta forma, la herramienta CASE diseñada, dá soporte al desarrollador y abarca la mayoría de las capas del marco de referencia provisto, con el objeto de producir software de calidad (facilitando la transferencia de información desde y hacia cada capa del modelo) [Press05]:

- Capa de interfaz de usuario: esta capa proporciona una ruta consecuente entre las acciones efectuadas por el usuario (el desarrollador de software) y las herramientas que están dentro del entorno (herramientas CASE y demás aplicativos). Incorpora un conjunto de herramientas de interfaz estandarizado, con un protocolo de presentación común. El kit de herramientas de interfaz contiene software para la gestión de la interfaz hombre-máquina, y una biblioteca (repositorio) de objetos de visualización. Ambos proporcionan un mecanismo consecuente para la comunicación entre la interfaz y las herramientas CASE individuales. El protocolo de presentación es el

conjunto de líneas generales que proporciona un mismo aspecto a todas las herramientas CASE. Las convenciones del diseño de pantalla, nombres y organización del menú, íconos, nombres de los objetos, utilización de los dispositivos de hardware de entrada (como teclado y mouse), y el mecanismo para acceder a las herramientas; se definen todos ellos como parte del protocolo de presentación.

Como las características que se adicionan a la herramienta CASE se diseñaron y desarrollaron sobre la misma arquitectura extensible de plugins de Eclipse (más adelante se describe en detalle), se mantiene el protocolo de presentación de la herramienta CASE actual; agregando algunos accesos al menú de herramientas para lograr las acciones deseadas en el aplicativo. La interfaz generada en las ventanas de diálogo consecuentes son consistentes con el resto de la interfaz habitual, dando al usuario la sensación de la integración nativa deseada.

- Capa de herramientas: coordina la utilización de las herramientas CASE incorporando un conjunto de servicios de gestión de herramientas con las herramientas CASE en sí. Estos servicios controlan el comportamiento de las herramientas dentro del entorno. Efectúan la sincronización y comunicación multitarea, coordinan el flujo de información desde el repositorio y el sistema de gestión de objetos a las herramientas, realizan las funciones de seguridad y auditoría y recogen métricas acerca de la utilización de las herramientas.

  La herramienta desarrollada combina distintos aplicativos y servicios a fin de coordinar el intercambio de datos entre el desarrollador, su entorno local de desarrollo, y la arquitectura virtual remota montada en la red local de los recursos de acceso lento y costoso. Mediante Putty/plink, Cygwin y scripts batch, se coordina el intercambio de información entre las distintas capas del entorno, integrando transparentemente las herramientas en el entorno CASE utilizado (las tecnologías implementadas se explican en detalle más adelante en este informe, durante las secciones de tecnología y arquitectura).
- Capa de gestión de objetos: por último, esta capa gestiona los cambios efectuados en la información (los datos; para el caso de uso de este trabajo, archivos de configuración o programas informáticos). En esencia, el software de esa capa de la arquitectura de marco de referencia, proporciona el mecanismo para la integración de herramientas (un conjunto de módulos estandarizados que acoplan las herramientas con el repositorio de datos; para este caso de aplicación, un controlador de versiones de código fuente). Además, esta capa proporciona los servicios de gestión de configuración, haciendo posible la identificación de todos los objetos (archivos) de configuración, llevando a cabo el control de versiones y proporcionando soporte para el control de cambios, auditorías y contabilidad de estados.

  Mediante las interfaces diseñadas y la herramienta de sincronización incremental de archivos rsync (junto con la configuración definida para esta herramienta) incorporadas a la arquitectura de la herramienta CASE; se localizan todos los componentes necesarios del software que sufrieran cambios por parte del desarrollador, y se sincronizan bajo demanda con el entorno remoto de desarrollo de modo veloz y transparente, para que el desarrollador pueda probar y verificar sus

cambios en el software de modo ágil. Por último, usa la herramienta habitual de control de versiones para impactar los cambios realizados en el software en el repositorio compartido de código fuente, sin necesidad de realizar pasos adicionales sobre el proceso corriente de desarrollo de software, o sobre los datos en sí.

2.2 - Patrones arquitectónicos

Las arquitecturas de software viables se construyen de acuerdo a ciertos principios estructurales básicos. Estos principios se describen como patrones arquitectónicos. Un patrón arquitectónico expresa un esquema organizativo de la estructura fundamental de los sistemas de software. Provee un conjunto predefinido de subsistemas de software, especifica sus responsabilidades, e incluye reglas y lineamientos para organizar las relaciones entre ellos. Estos patrones son plantillas para arquitecturas de software concretas, es decir, son los patrones de más alto nivel. Especifican las propiedades estructurales a lo largo de todo el sistema de una aplicación y tienen impacto en la arquitectura de sus subsistemas. [Wiley96]

Cada patrón arquitectónico ayuda a los desarrolladores o ingenieros de software a alcanzar una propiedad global específica en el sistema (como puede ser la usabilidad de una interfaz de usuario). Los patrones que ayudan a soportar propiedades similares se agrupan en categorías. Se expone aquí la categoría que engloba a los sistemas distribuidos, dado que es la categoría correspondiente a la arquitectura definida para el sistema descripto; y tal como se realizó con las herramientas CASE, se describe el patrón correspondiente para la herramienta desarrollada en este trabajo, en este caso, el patrón Broker.

2.2.1 - Sistemas distribuidos

La computación distribuida es un modelo para resolver problemas de computación masiva utilizando un gran número de computadoras organizadas en granjas dentro de una infraestructura de telecomunicaciones distribuida.

Un sistema distribuido se define como una colección de computadoras separadas físicamente y conectadas entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema (no necesita conocer qué servicios brinda cada máquina, ni dónde residen los mismos). De esta forma, el usuario accede a los recursos remotos de la misma manera en que accede a recursos locales. [Tennen96]

Los sistemas distribuidos pueden ser muy confiables, ya que si un componente del sistema falla, otro componente puede ser capaz de reemplazarlo (esto se denomina tolerancia a fallos). El tamaño de un sistema distribuido puede ser muy variado, ya sean decenas de

hosts (red de área local), centenas de hosts (red de área metropolitana), y miles o millones de hosts (Internet); es decir que puede crecer hasta casi cualquier magnitud (esto se denomina escalabilidad). [Tennen96]

Algunas de las características comunes y deseables en los sistemas distribuidos son:

- Para cada uno de los usuarios, las tareas desarrolladas en el sistema distribuido deben ser similares al trabajo hecho en el sistema centralizado (es decir, su propia máquina o sistema local).
- Es necesaria una capa adicional de seguridad interna en el sistema distribuido.
- El sistema se ejecuta en múltiples computadoras.
- Tiene varias copias del mismo sistema operativo o de diferentes sistemas operativos, que proveen los mismos servicios a los usuarios finales.
- El entorno de trabajo debe ser cómodo; en sentido que el uso de múltiples procesadores y el acceso remoto deben ser lo más transparente posible al usuario. [Tennen96]
- El sistema es dependiente de alguna forma de red (LAN, MAN, WAN, etc.).
- Se requiere de compatibilidad entre los dispositivos conectados (ya sea compatibilidad nativa de los componentes, o simulación dada por software adicional).
- Los diferentes equipos del sistema interactúan entre sí.
- El software debe ser diseñado de modo que sea compatible con varios usuarios y sistemas operativos.

La computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier computadora individual, mientras se mantiene la flexibilidad de trabajar en múltiples problemas más pequeños. Por lo tanto, este modelo de computación es naturalmente un entorno multiusuario; por ello, las técnicas de autorización segura son esenciales antes de permitir que los recursos informáticos sean controlados por usuarios remotos.

2.2.1.1 - Broker

El patrón arquitectónico Broker puede ser usado para estructurar sistemas de software distribuidos con componentes desacoplados que interactúan a través de invocaciones a servicios remotos. Un componente broker es responsable de coordinar la comunicación, como puede ser reenviar requerimientos, transmitir resultados e informar excepciones. [Wiley96]

Construir un complejo sistema de software como un conjunto de componentes desacoplados, en lugar de una aplicación monolítica, resulta en mayor flexibilidad, facilidad de mantención, adaptabilidad y escalabilidad.

Ahora bien, cuando los componentes distribuidos interactúan unos con otros, se requiere

alguna clase de comunicación inter-proceso. Si los componentes manejan la comunicación ellos mismos, el sistema resultante se enfrenta a severas dependencias y limitaciones. Por ejemplo, el sistema se vuelve dependiente del mecanismo de comunicación usado y los clientes requieren conocer la ubicación de los servidores.

También hay que proveer servicios para agregar, remover, activar o localizar componentes. Las aplicaciones que usen estos servicios no deben depender de detalles específicos del sistema para garantizar portabilidad e interoperabilidad, incluso en una red heterogénea.

En resumen, se puede usar el patrón Broker cuando:

- los componentes deben acceder a servicios provistos por otros componentes, a través de invocaciones a servicios remotos, de forma transparente a su ubicación.
- se debe poder agregar, quitar o cambiar componentes en tiempo de ejecución (dinámicamente).
- la arquitectura debe ocultar los detalles específicos de implementación de los componentes y servicios a los usuarios finales.

Para el caso de aplicación de este trabajo, la herramienta implementada actúa de broker al sistema de software y hardware desplegado, ya que el usuario no necesita saber en dónde se encuentran los servidores con los que interactúa ni tampoco cuales sirven cada servicio. Este proceso es transparente, el usuario interactúa con la herramienta desplegada en su terminal, a través de la cual instruye la ejecución de los respectivos servicios en donde corresponda.

Los administradores pueden crear nuevas instancias de máquinas virtuales que provean los servicios necesarios, e incluso realojar una instancia determinada en otro servidor distinto, sin que el usuario deba alterar sus procesos. Sólo bastará que los administradores realicen la configuración de red pertinente.

Además los scripts utilizados por la herramienta CASE para comunicarse con los servicios y ejecutar determinadas acciones, pueden ser reemplazados en cualquier momento por otros equivalentes, que realicen alguna acción adicional o incluso ser reescritos en otro lenguaje de programación según convenga. Mientras que se mantenga la interfaz con la herramienta, el usuario no deberá cambiar su metodología habitual ni preocuparse en absoluto por conocer los detalles de implementación.

2.3 - Patrones de diseño

Los subsistemas de una arquitectura de software, como así también las relaciones entre ellos, consisten a menudo de muchas unidades pequeñas de arquitectura. Para describir a estas pequeñas unidades usamos los patrones de diseño. Un patrón de diseño provee un

esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe una estructura reusable y comúnmente recurrente de comunicación entre componentes, que resuelve un problema de diseño general dentro de un contexto en particular.

Estos patrones son más pequeños en comparación a los patrones arquitectónicos, pero tienden a ser independientes de un lenguaje o paradigma de programación en particular (por lo general pueden implementarse fácilmente con lenguajes orientados a objetos, pero de todos modos son bastante generales como para poder ser adaptados a la mayoría de las prácticas de programación). Otra propiedad importante de los patrones de diseño es que son independientes de un dominio de aplicación en particular; ya que tratan con la estructura de la funcionalidad de la aplicación, pero no con la implementación en sí misma de dicha funcionalidad. La aplicación de estos no tiene efecto en la estructura fundamental de un sistema de software, pero puede tener cierta influencia en la arquitectura de un subsistema determinado.

Muchos patrones de diseño proveen estructuras para descomponer componentes o servicios complejos en equivalentes más sencillos; mientras que otros se enfocan en la cooperación efectiva entre ellos. [Wiley96]

Como los patrones arquitectónicos, los patrones de diseño se agrupan en categorías que denotan comportamiento similar o para agrupar patrones que se enfocan en resolver problemáticas afines. Así como se realizó en las secciones anteriores, se describen aquí los patrones de diseño que poseen relación directa o que han sido implementados en la herramienta desarrollada, como así también se mencionan las categorías a las que corresponden estos patrones.

2.3.1 - Control de acceso

Estos patrones protegen y controlan el acceso a recursos o componentes críticos o escasos. A menudo un componente o incluso todo un subsistema no puede o no debería ser accedido directamente por sus clientes; por distintas razones de seguridad, consistencia de datos, falta de disponibilidad local, etc. Los patrones de esta categoría coordinan o transforman estos recursos, de forma que puedan ser accedidos por los clientes (o simular el acceso a los mismos) de una forma controlada y segura.

2.3.1.1 - Proxy

El patrón Proxy permite a los clientes de una aplicación, comunicarse con una representación de un componente, en lugar de comunicarse con el componente mismo. Esto sirve a muchos propósitos, incluyendo una eficiencia mejorada, un acceso más simple o protección contra

accesos o acciones no autorizadas.

Este patrón es ampliamente usado. Prácticamente todo sistema distribuido o infraestructura para sistemas distribuidos usa el patrón para representar componentes remotos localmente. Por ejemplo, la herramienta implementada en este trabajo, permite la visualización remota de logs (bitácoras informativas). Estos logs no son accedidos directamente por el cliente, sino que son transformados en una representación adecuada para poder ser visualizados localmente en una terminal de texto. Esto permite que el cliente acceda al log, pero por ejemplo no pueda modificarlo. De esta forma procesos del background (tareas ejecutando remotamente en segundo plano) ajenos al desarrollador pueden realizar tareas de mantenimiento con estos logs, como backup o purga de logs antiguos, sin preocuparse por posibles modificaciones o pérdida de datos, ya que el usuario final puede visualizar una representación local de estos logs, pero no los logs en sí, impidiendo su manipulación.

2.3.2 - Comunicación

Los patrones de esta categoría ayudan a organizar la comunicación entre los componentes de la aplicación.

Dado que en la actualidad es impensado el hecho de que un sistema completo ejecute en una simple computadora en lugar de una red de computadoras, la distribución de las aplicaciones impone un importante requerimiento: los sistemas distribuidos deben colaborar, y para ello necesitan mecanismos de comunicación entre ellos.

Algunos de los aspectos más importantes de la comunicación inter-proceso, y a los cuales se enfocan los patrones de diseño bajo esta categoría, son:

- Encapsulación: encapsular las facilidades de comunicación significa ocultar los detalles de los mecanismos de comunicación subyacentes a los usuarios.
- Transparencia de ubicación: permite a las aplicaciones acceder a componentes remotos sin ningún conocimiento de su ubicación física.
- Consistencia: cuando muchos componentes colaboran en la resolución de una tarea particular, surge otro problema de comunicación a resolver; se torna necesario mantener consistentes los datos compartidos.

2.3.2.1 - Forwarder-Receiver

Provee comunicación transparente inter-procesos en sistemas de software con un modelo de interacción punto a punto (peer-to-peer).

Los puntos (componentes por ejemplo) son los responsables de las tareas de la aplicación. Para llevarlas a cabo necesitan comunicarse con otros puntos, que pueden estar ubicados en

otros procesos u otros equipos de la red. Cada punto conoce el nombre del otro punto con el que necesita comunicarse. Entonces usa un forwarder (promotor) para enviar mensajes al otro punto, y un receiver (receptor) para recibir los mensajes desde los otros puntos. Estos mensajes son requerimientos que un punto envía a otros puntos remotos, y respuestas que un punto transmite a quienes originaron los requerimientos.

Los componentes forwarders envían mensajes a través de todo el sistema, proveyendo una interfaz general que es una abstracción de un mecanismo particular de comunicación inter-proceso. También poseen capacidades de marshaling (codificación) de mensajes según el protocolo de comunicación usado; y tienen mapeos de nombres de componentes a direcciones físicas. Para enviar un mensaje a un punto remoto, identifican su ubicación en base a su nombre, realizan el marshaling correspondiente y utilizan el mecanismo implementado para la comunicación. Los componentes receivers son los responsables de recibir los mensajes. Al igual que los forwarders ofrecen una interfaz abstracta y un mecanismo de unmarshalling (decodificación) de mensajes; es decir que implementan la vía opuesta en la comunicación.

En la herramienta desarrollada, se implementan estos mecanismos para ofrecer facilidades de comunicación con los servicios remotos. Por ejemplo, para reiniciar el servidor web remoto, no hace falta entablar una comunicación directa con él, ni tampoco conocer el comando específico que realiza dicha acción. Se hace uso de un control en la interfaz de usuario que invoca al forwarder correspondiente. Este localiza la ubicación del servidor web según su configuración, y convierte la acción del usuario en el comando correcto (marshaling). El receiver espera por la respuesta del servidor web remoto, y traduce su estado en una descripción entendible para el usuario (unmarshalling), como “Reinicio exitoso” o “Reinicio erróneo”.

2.3.2.2 - Client-Dispatcher-Server

Este patrón describe una comunicación entre procesos transparente de la ubicación de los mismos, en una estructura cliente/servidor. Logra esto agregando una capa intermedia entre los clientes y los servidores: el despachador (dispatcher). Este componente provee transparencia de ubicación, reemplazándola por nombres de servicio; además de ocultar los detalles de establecer la comunicación o conexión entre clientes y servidores. Para proveer su servicio de nombres, el despachador implementa funciones para registrar y localizar servidores.

En el caso de aplicación de este trabajo, el administrador configura por única vez las direcciones físicas (direcciones IP) de los recursos asignados al usuario (como servidor web, base de datos, etc.) en la etapa de despliegue de la herramienta, en la terminal del mismo. Luego el usuario no necesita conocer la ubicación de los servicios que accede o invoca, ya que al realizar acciones sobre la interfaz de la herramienta, esta resolverá la dirección

adecuada al servicio invocado en base a su configuración (despachador), y creará el enlace correcto hacia el servicio en cuestión.

2.3.2.3 - Publisher-Subscriber

Este patrón ayuda en la tarea de mantener consistente la información entre procesos cooperativos (ayuda a mantener sincronizado el estado de los mismos). Para lograr este propósito realiza una propagación de una vía (en un sentido) de los cambios realizados. Así, el productor (publisher) notifica los cambios en su estado a cualquier número de suscriptores (subscribers).

El productor mantiene un registro de todos los componentes actualmente suscriptos y ofrece una interfaz para suscripción y cancelación de la suscripción. Cuando el estado del productor cambia, envía una notificación a todos sus suscriptores, los cuales reciben las modificaciones en los datos y actúan frente a ello (en caso de que la modificación sea de su interés).

Para el caso de este trabajo, dado que se desarrolló una herramienta destinada a facilitar tareas de desarrollo remoto de software; el estado a mantener consistente es el software mismo, el software que permanentemente es modificado por el desarrollador. Para garantizar la consistencia de la copia local del software (aquella siendo modificada) con la copia remota (aquella siendo ejecutada), se provee un mecanismo de sincronización de una vía (local a remota) para aplicar en los servicios remotos (servidor web por ejemplo) los cambios hechos por el desarrollador. Esto significa que bajo demanda, el desarrollador puede acceder a un control en su interfaz, que invoca el mecanismo de sincronización de recursos, a fin de aplicar los cambios realizados por el desarrollador, en la arquitectura de software desplegada remotamente. Más adelante, en la sección correspondiente, se explica con más detalle el mecanismo de sincronización implementado.

3 - Tecnologías implementadas

En esta sección se presentan las bases tecnológicas sobre las que se apoya el desarrollo de este trabajo. Se hace una descripción exhaustiva de aquellas tecnologías que se adoptaron y combinaron para lograr los objetivos propuestos. Se muestran cuáles son los aspectos principales de cada una, qué lleva a considerar estas tecnologías, cuáles son las ventajas de su aplicación en un entorno como el descripto (o en general, qué beneficios implican en grandes equipos de desarrollo o empresas) y, a su vez, algunos ejemplos de uso reales de estas tecnologías. Luego, en el próximo capítulo, se describe la implementación específica de estas tecnologías en la herramienta CASE desarrollada.

3.1 - Virtualización de equipos

La virtualización refiere a la creación (a través de software de aplicación) de una versión virtual de algún recurso tecnológico, como puede ser una plataforma de hardware, un sistema operativo, un dispositivo de almacenamiento u otros recursos de red [Turban08].
Dicho de otra manera, se refiere a la abstracción de los recursos de una computadora, que crea una capa de abstracción entre el hardware de la máquina física (host / anfitrión) y el sistema operativo de la máquina virtual (guest / huésped), dividiéndose el recurso en uno o más entornos de ejecución paralelos.
Esta capa de software maneja, gestiona y arbitra los cuatro recursos principales de una computadora (unidad de CPU, módulos de memoria, dispositivos periféricos de entrada/salida y conexiones de red), y así podrá repartir dinámicamente dichos recursos entre todas las instancias de las máquinas virtuales definidas en la computadora host central. Esto hace que se puedan tener varias computadoras virtuales ejecutándose en la misma computadora física, compartiendo sus recursos transparentemente.
La virtualización se encarga de crear una interfaz externa que encapsula una implementación subyacente mediante la combinación de recursos en localizaciones físicas diferentes, o por medio de la simplificación del sistema de control. La máquina virtual, en general, simula una plataforma de hardware autónoma incluyendo un sistema operativo completo que se ejecuta como si estuviera totalmente instalado en la instancia virtual guest.

Básicamente, y puesto en términos menos técnicos, la virtualización es poder ejecutar sobre una misma máquina más de un sistema operativo y sus aplicaciones correspondientes, de forma simultánea. De esta forma, se podría tener en una misma máquina ejecutándose a la vez, un sistema operativo Linux para servir una página web, y un Windows Server para soportar una intranet (o servidor de archivos, o servidor de impresión, o un programa de contabilidad, etc). Y como si fueran máquinas separadas, con diferente dirección IP, con diferentes accesos, con diferentes perfiles de seguridad [Loogic08].

La meta general de la virtualización es centralizar las tareas administrativas, mientras se mejora la escalabilidad y la utilización total de los recursos de hardware.

Existen diferentes formas de virtualización: es posible virtualizar el hardware en un servidor, el software de un servidor, virtualizar las sesiones de los usuarios de un sistema, virtualizar aplicaciones y también se pueden crear máquinas virtuales en una computadora de escritorio [Micro11].
A los fines de este trabajo, interesa particularmente la “virtualización de plataforma”, que refiere a la simulación de máquinas virtuales. Esta se lleva a cabo en una plataforma de hardware mediante un software host, que es un programa de control (usualmente llamado Hypervisor o Virtual Machine Monitor) que simula un entorno computacional (máquina virtual) para su software guest. Este software guest, que generalmente es un sistema operativo completo, se ejecuta como si estuviera instalado en una plataforma de hardware autónoma. Típicamente muchas máquinas virtuales son simuladas en una máquina física dada. Para que el sistema operativo guest funcione, la simulación debe ser lo suficientemente grande como para soportar todas las interfaces externas de los sistemas huéspedes, las cuales pueden incluir (dependiendo del tipo de virtualización) los drivers de hardware.

Si bien la virtualización no es un invento reciente, con la consolidación del modelo de la “computación en la nube” (cloud computing), la virtualización ha pasado a ser uno de los componentes fundamentales de la informática moderna; especialmente en lo que se denomina infraestructura de nube privada, permitiendo a las organizaciones darle la agilidad de la arquitectura “de nube” a sus propios centros de datos (datacenters) [MicroTCV1].

3.1.1 - Ventajas

La virtualización proporciona oportunidades inmensas para consolidar la arquitectura de grandes sistemas de cómputo y de información. Se detallan entonces, algunos de los retos que afronta esta tecnología, y las ventajas de su aplicación bajo estos entornos:

- Reutilización de hardware existente (para utilizar software más moderno) y optimizar el aprovechamiento de todos los recursos de hardware [MicroTCV2].
  Esto genera índices de utilización de hardware de servidor y de almacenamiento más altos (menor cantidad de recursos ociosos); ya que las cargas de trabajo pueden ser encapsuladas y transferidas a los sistemas inactivos o sin uso; lo cual significa que los sistemas existentes pueden ser consolidados, y de esta forma retrasar o evitar las compras de capacidad adicional para servidores, resultando en ahorros de costo y mayor eficiencia (no sólo aporta el beneficio directo en la reducción del hardware necesario, sino también en los costos asociados).
- Rápida incorporación de nuevos recursos para los servidores virtualizados, y sin la necesidad de suspender el suministro del servicio en general durante la instalación.
- Reducción de los costos de consumo necesario, de forma proporcional al índice de

consolidación logrado.
La electricidad requerida para que funcionen los centros de datos de clase empresarial ya no está disponible en suministros ilimitados (incluyendo el costo de los servidores en función y de los sistemas de refrigeración), y el costo de la energía no hace más que aumentar constantemente. Utilizando virtualización para consolidar los recursos disponibles, se torna posible cortar el consumo total de energía y ahorrar dinero de una manera significativa.

- Reducción de los costos de espacio necesario, de forma proporcional al índice de consolidación logrado.
  La extensión de servidores permanece como un serio problema en la mayoría de los datacenters empresariales, siendo la expansión del mismo datacenter no siempre una opción, por los altos costos de construcción y la dificultad de obtener el espacio adecuado. La virtualización puede aliviar la tensión mediante la consolidación de muchos sistemas virtuales en menos sistemas físicos, logrando un importante ahorro de espacio.
- Administración global centralizada y simplificada, reduciendo la carga total de trabajo administrativo y cortando el total de costos de operación.
- Permite gestionar el centro de procesamiento de datos como un pool de recursos (fuente de recursos compartidos) o agrupación de toda la capacidad de procesamiento, memoria, red y almacenamiento disponible en la infraestructura.
- Mejora en los procesos de clonación y copia de sistemas.
  Mayor facilidad para la creación de entornos de testeo que permiten poner en marcha nuevas aplicaciones sin impactar a la versión productiva del producto, **agilizando el proceso de las pruebas**. La virtualización permite crear fácilmente nuevos ambientes de desarrollo, o mantener ambientes de pre-producción (formalmente denominados "ambientes de stage o staging"), que se sincronicen automáticamente con los servidores productivos (el paraíso del desarrollador de software).
- Aislamiento, un fallo de sistema general de una máquina virtual no afecta al resto de las máquinas virtuales en ejecución.
- Mejora y agiliza los procesos de backup.
  La restauración de una copia de seguridad ante un desastre, en cualquier otro sistema virtualizado, es rápida y sencilla. Lo cual asegura que ante una falla importante, es posible recuperarse en un plazo bastante más reducido que con un sistema normal (no virtualizado) [Loogic08].
  Esto se debe a que la virtualización permite el acceso a técnicas de “snapshotting”. Un snapshot (o copia instantánea de un volúmen) es el estado de una máquina virtual y sus dispositivos de almacenamiento, en un punto exacto de tiempo. Los snapshots pueden “tomarse” con una simple orden al hypervisor, y pueden ser revertidos a demanda, con el efecto de que la máquina virtual aparenta ser exactamente la misma que lo fue en el instante en que la copia fue tomada (el punto del tiempo exacto y específico). Esta capacidad es extremadamente útil como técnica de rápido backup, en lugar de otras operaciones mucho más riesgosas.
- Reduce los tiempos de parada del servicio a usuarios finales (downtime aplicativo).

La virtualización puede incrementar la disponibilidad de los índices del nivel de servicio (SLA, Service Level Agreement / acuerdo de nivel de servicio) en general y proporcionar nuevas opciones de soluciones para la recuperación de desastres, asegurando o mejorando la continuidad del negocio.

- Migración en caliente (hot swap) de máquinas virtuales (sin pérdida de servicio) de un servidor físico a otro, eliminando la necesidad de cortes de servicio planificados por mantenimiento de los servidores físicos (los “snapshots” descritos con anterioridad pueden ser movidos de una máquina host a otra con facilidad, lo que se conoce como “teleportación”).
- Balanceo dinámico de máquinas virtuales entre los servidores físicos que componen el pool de recursos, garantizando que cada máquina virtual ejecute en el servidor físico más adecuado y proporcionando un consumo de recursos homogéneo y óptimo en toda la infraestructura.
- Contribución al medio ambiente por menor consumo de energía en servidores físicos [MicroTCV2].
- Las compañías pueden administrar de una forma mejor las actualizaciones de software y los cambios en las aplicaciones y sistemas operativos, sin distraer o interrumpir la labor de los empleados (usuarios del sistema).

Claro está que virtualizar la infraestructura de una compañía y lograr beneficiarse con la ventajas que acarrea y que se mencionaron, conlleva algunos retos y dificultades al comienzo. En primer lugar, adquirir el software adecuado para virtualizar, que en ocasiones puede tener un costo (o puede optarse por versiones de software libre como en el caso desarrollado en este trabajo), y especialmente en este tipo de casos, debería ser instalado y mantenido por profesionales en esta tarea. En definitiva, existe el ahorro mencionado en máquinas, pero existiendo un costo adicional en licencias y posiblemente en mantenimiento; pero en todo caso, un costo menor que comprar un nuevo servidor y mantenerlo con cada nueva instalación o ampliación del sistema (ver "scale out vs scale up" en la próxima sección).
A su vez, pueden existir picos de capacidad, es decir que en determinados momentos, el servidor (host) no pueda atender a todos sus entornos virtuales por causa de algún proceso en particular. Si bien es posible configurar qué y cuánto (en recursos) pueden usar cada uno de los entornos virtuales, al final serán menos recursos que si estuviera en una máquina dedicada para cada uno de ellos. En definitiva, se debe estar atentos al dimensionamiento si no se desean problemas en este sentido [Loogic08].

3.1.2 - Scale out vs scale up

La **escalabilidad** es la habilidad de un sistema, red o proceso, para manejar un aumento en la cantidad de trabajo de una manera capaz o fiable, o su habilidad para ser agrandado para acomodar dicho crecimiento [Bondi00]. La escalabilidad, como una propiedad de los

sistemas, es generalmente difícil de definir [Hill90], y en cada caso en particular se torna necesario definir los requerimientos específicos para la escalabilidad en esas dimensiones donde se crea que son importantes. Un sistema cuyo rendimiento aumenta luego de haberle añadido más capacidad de hardware, de modo proporcional a la capacidad añadida, se dice que es un **sistema escalable**.

Particularmente para el caso desarrollado en este trabajo, se puede definir que la arquitectura es escalable si es factible aumentar la cantidad de máquinas virtuales de desarrollo a medida que se suman más desarrolladores de software en el equipo de trabajo, implicando un agregado proporcional de hardware subyacente (proporcionalmente más pequeño ya que se están virtualizando varias máquinas virtuales en unos pocos equipos físicos) y sin ver degradada la performance general de todas las máquinas virtuales de la arquitectura, como así también la velocidad de acceso a los recursos remotos. Es decir, que se busca alcanzar lo que formalmente se define como **escalabilidad administrativa**, y refiere a la habilidad de que un número incremental de usuarios compartan fácilmente un sólo sistema distribuido [Willey05].

Existen dos principales categorías de métodos para agregar más recursos a un sistema o una aplicación en particular: escalado horizontal o vertical [IPaDPS07].

- Escalabilidad horizontal: un sistema escala horizontalmente (**scale out**) si al agregar más nodos al mismo, el rendimiento del sistema mejora. Por ejemplo, al añadir una computadora nueva a un sistema puede mejorar el rendimiento de todo el sistema (que balancee la carga de una aplicación de software distribuido entre las antiguas máquinas y la nueva). Aquí, el tamaño de la escalabilidad (**size scalability**) es el máximo número de procesadores que un sistema puede manejar [Willey05]; y pese a que se trate de un número específico para cada arquitectura implementada y que el software y otros parámetros de la infraestructura siempre imponen límites, en la actualidad se entiende este valor como literalmente infinito.
    - Ventajas:
        - Mucho más económico que escalar verticalmente
        - Más fácil de montar un esquema tolerante a fallas
        - Más fácil de actualizar el sistema en su conjunto
    - Desventajas:
        - Mayores costos de licenciamiento, debido a la necesidad de adquirir licencias para cada equipo físico, para cada equipo virtual, y para el software Hypervisor
        - Mayor complejidad en la infraestructura física del datacenter
        - Mayores costos en electricidad y refrigeración
        - Generalmente requiere más equipamiento de red para llevar a cabo la interconexión de los equipos
- Escalabilidad vertical: un sistema escala verticalmente o hacia arriba (**scale up**), cuando al añadir más recursos a un nodo particular del sistema, este mejora en

conjunto. Por ejemplo, añadir más capacidad de memoria, un disco duro más grande o más rápido, o un procesador más veloz a una computadora en particular, puede mejorar el rendimiento del sistema global.

- Ventajas:
  - Menor consumo de memoria que al ejecutar múltiples servidores
  - Los costos de refrigeración son menores que escalando horizontalmente
  - Generalmente, este modelo es más fácil de implementar
  - Menores costos de licencias
  - A menudo requiere el uso de menos hardware de red, que escalar horizontalmente
- Desventajas:
  - El costo de adquirir múltiples equipos
  - El riesgo de que una falla de hardware cause indisponibilidad en el servicio brindado es mayor
  - Generalmente, las posibilidades de upgrade futuro (mejoras/ampliación de hardware) en los equipos quedan muy limitadas al vendedor específico original (lo que se conoce formalmente como **vendor lock-in**)

Escalar sistemas existentes permite hacer un uso más eficiente de las tecnologías de virtualización, ya que provee más recursos al sistema operativo anfitrión (en el caso de scale up) o al Hypervisor (en el caso de scale out) para compartir entre los módulos aplicativos. Tomar ventaja de estos recursos puede ser llamado también “escalar”, como así también “escalabilidad de aplicación” refiere a una mejora en el rendimiento de las aplicaciones de software cuando están ejecutando en una versión escalada del sistema [Willey05].

Ahora bien, considerando la definición hecha de escalabilidad para la arquitectura de software descripta, y los dos métodos posibles de hacerla escalar… ¿Cuál es el método que conviene implementar en dicha arquitectura?
Hay ventajas y desventajas entre ambos modelos. Una mayor cantidad de computadoras implica un incremento en la complejidad administrativa de las mismas; así como un modelo de programación más complejo y problemas como throughput (rendimiento, cantidad de datos o información siendo transmitida por el sistema) y latencia entre los nodos del sistema; sin mencionar el hecho de que algunas aplicaciones requeridas pueden no prestarse a ejecutar en un modelo de computación distribuida.
En el pasado, la diferencia de costo (monetario) entre ambos modelos favorecía el “scale up” para aquellas aplicaciones o sistemas que soporten este paradigma, pero los avances de la última década en tecnologías de virtualización han removido dicha ventaja, haciendo que desplegar un nuevo nodo virtual sobre un Hypervisor (en aquellos casos en que esto sea posible) es casi siempre más económico que realmente adquirir e instalar un equipo físico.

Es por esto que entonces, **se decide aplicar la estrategia de “scale out” cuando sea**

necesario, ya que al analizar los hechos en retrospectiva, considerando los problemas conocidos en el campo de las comunicaciones, las tareas de los profesionales del mantenimiento de sistemas y redes de información, y comparando las aptitudes de los dos modelos, se llega a la conclusión de que:

- Es posible automatizar el proceso de despliegue de nuevos nodos a través de software aplicativo, facilitando la tarea administrativa requerida para el profesional de la materia; además de tornarla mucho más veloz.
- En lugar de requerir adquirir y desplegar nuevo hardware cada vez que se torna necesario escalar para entregar un equipo a un nuevo desarrollador, sólo deberá hacerse cuando el sistema en su conjunto comience a ver degradada su performance.
- El precio actual del hardware de servidor y de una solución de virtualización basada en un Hypervisor, es más económico que el costo de adquirir múltiples equipos económicos individuales (y en el caso del Hypervisor, existen soluciones profesionales gratuitas y libres).
- Escalar la arquitectura con nuevos nodos virtuales, no implica cortes de servicio (downtime) sobre los nodos actuales, una diferencia importante sobre el scale up, que durante la tarea de mantenimiento o mejora de un equipo (upgrade), éste puede presentar downtime al usuario (desarrollador de software) si uno de los componentes reemplazados es crítico (por ejemplo, el procesador o la memoria).
- Puede moverse con relativa facilidad una máquina virtual de un nodo físico a otro, a través del software de virtualización implementado, si un equipo en particular presentara fallas de hardware.
- Puede automatizarse el proceso de backup de máquinas virtuales (o de los discos rígidos montados sobre las mismas) y facilitar y acelerar el proceso de recupero ante problemas.
- Es posible seguir escalando agregando nuevos equipos (incluso heterogéneos entre sí) y actualizarlos fácilmente, una posibilidad muy limitada en el modelo vertical: “Escalar verticalmente es fracasar verticalmente. Usualmente llegas a un punto en donde o bien los costos económicos se tornan imprácticos o ya no existe más un hardware mayor.” [Abbott11]

3.1.3 - Algunas soluciones existentes

Entre los principales proveedores de software que han desarrollado tecnologías de virtualización integrales (que abarcan todas las instancias: servidor, aplicaciones, escritorio) se encuentran, por ejemplo VMware y Microsoft. Estas compañías han diseñado soluciones específicas para virtualización, como VMware Server y Windows Server Hyper-V para la virtualización de servidores, respectivamente.

Si bien en la actualidad existen varias formas distintas de licenciar o adquirir software de modo legal, básicamente se puede seguir dividiendo todo el espectro de software en dos grandes grupos: los de pago y los gratuitos.

Como se mencionó, dentro de los programas de pago se encuentra el VMware, que es uno de los programas de virtualización referentes del mercado, como así también Windows Server Hyper-V [MicroTCV2] cuya función de virtualización está incluida sin cargo en la licencia del servidor Windows Server. Existe una versión más básica de VMWare que es gratuita (VMware Player), que permite virtualizar a través de una máquina virtual ya configurada. Algo novedoso de la actualidad, es que también existen páginas web como EasyVMX, que permiten especificar la configuración deseada en un formulario, y descargar una máquina virtual acorde a dichas necesidades, lista para ser ejecutada en VMware Player. Parallels Virtuozzo Containers, es otro de los programas de pago más famosos, que permite la virtualización a nivel de sistema operativo o hardware. Típicamente suele emplearse para virtualizar Windows y, en menor medida, GNU/Linux.

Dentro de los programas gratuitos existe Virtual PC de Microsoft, que es un producto de Windows, compatible con versiones avanzadas de XP, Vista y Windows 7.
A su vez, también existen soluciones gratuitas y de código libre. Dentro de este grupo están el Xen, OpenVZ y VirtualBox, que funcionan tanto en Mac OS-X, como en Windows y GNU/Linux, y todos permiten virtualizar estos tres sistemas operativos. Particularmente en la implementación de este trabajo, se usó Xen de Oracle para el despliegue de la arquitectura requerida de máquinas virtuales. Un poco más adelante, en el capítulo 3.2, se detalla parte de la arquitectura de hardware virtual implementada, y cómo contribuye Oracle Xen a orquestar dicho hardware.

3.1.4 - Aplicaciones y usos

La virtualización no es una rama de la informática exclusiva de las grandes compañías de tecnología, ni de aquellos que usen o ejecuten hardware de servidor increíblemente poderoso. La realidad, es que con el correr del tiempo, la virtualización de hardware o de software ha llevado a crear una multitud de soluciones y aplicaciones a los más diversos problemas, alcanzando a casi cualquier profesional de la tecnología de la información, y a muchas otras áreas también. No es ilógico pensar que en la actualidad, prácticamente todos los que poseen acceso a la tecnología (aunque sea hogareña o incluso internet), hacen uso de aplicaciones o ambientes virtualizados, sean conscientes de ello, o no.

A continuación, se listan algunas aplicaciones de la virtualización en la vida real. Muchas de ellas no son sólo conceptos o ideas interesantes, sino que la mayoría goza ya de soluciones profesionales, enfocadas a resolver de modo práctico el problema o reto general en sí.

- **Ejecutar aplicaciones antiguas**: muchas veces se posee algún programa de aplicación que no ejecuta correctamente (o no ejecuta en absoluto) en versiones actuales de los sistemas operativos, pero sí ejecuta correctamente en versiones pasadas, incluso muy antiguas. Esto es usual en empresas que aún utilicen software

administrativo antiguo (comúnmente denominado “legacy”), que por alguna u otra razón, nunca actualizaron. Otras veces se torna necesario ejecutar este software debido a la inexistencia de versiones corrientes, porque surge la necesidad de abrir o editar archivos en formatos antiguos, o bien por simple nostalgia.

En estos casos, basta con instalar la versión necesaria del sistema operativo (aquella versión antigua que ejecute correctamente, o para la cual fue diseñado originalmente el software en cuestión) en una máquina virtual; y luego, instalar la aplicación necesaria en él.

Algunas soluciones de virtualización, como por ejemplo VMware Player, permiten ir un poco más allá, permitiendo que las aplicaciones que ejecutan en la máquina virtual, luzcan como si en realidad estuvieran ejecutando nativamente en la computadora host. Esto es, dotando al contexto de ejecución de su propia ventana, barras de botones o herramientas, menúes y demás, del sistema operativo anfitrión. Así se crea la noción en el usuario final de que el entorno de virtualización en realidad no existiera, y la experiencia de uso mejora, principalmente en usuarios inexpertos.

- **Acceder a datos corrompidos**: las amenazas a la seguridad informática son un serio problema desde hace décadas, y probablemente gran parte de los usuarios de computación fueron víctimas de cuando menos un virus informático, afectando algún archivo de datos o programa.

  Si por alguna razón se requiriera acceder a un archivo al que una solución antivirus por ejemplo, hubiera informado que posee virus; se lo puede realizar de una forma segura a través de una máquina virtual. La mayoría de las soluciones de virtualización ofrecen capacidades de snapshotting (ver capítulo 3.1.1), de modo que se puede crear una máquina virtual con un estado “congelado” o seguro de su sistema operativo y disco rígido. Así, abrir el archivo infectado en dicha máquina, y si el virus llegase a causar daños en el sistema, entonces simplemente restaurar la máquina virtual al estado seguro anterior (o eliminar dicha máquina si ya no fuera necesaria), y listo.
- **Probar actualizaciones o nuevas configuraciones de software**: la técnica del punto anterior no sólo puede aplicarse a software malicioso (malware). También puede usarse la máquina virtual para probar nuevo software, actualizaciones o incluso nuevas configuraciones de software, antes de aplicarlas en el sistema operativo real. Esto es muy útil principalmente para los administradores de sistemas, como método para probar si los cambios en el software pueden causar algún daño o efecto colateral indeseado en el sistema, sin realizar la prueba en el sistema real.
- **Ejecutar un sistema operativo sobre otro distinto**: simplemente por el hecho de querer probar un nuevo sistema operativo, o bien para poder usar aplicaciones que no están diseñadas para el sistema operativo corriente de una computadora.
- **Respaldar un sistema operativo o servidor completo (hacer back-up)**: dado que el sistema operativo de una máquina virtual se encuentra totalmente contenido en unos pocos archivos en la máquina real, realizar un respaldo de dicha instalación es un proceso tan simple como backupear cualquier otro archivo común y corriente.

  Crear de este modo, backups periódicos de instalaciones de servidores existentes,

puede proveer una redundancia de datos vital, por si ocurre una catástrofe en el servidor que brinda el servicio actualmente.

- **Emular otras arquitecturas de hardware**: aprovechar las capacidades de cómputo de una arquitectura puntual para simular mediante software la de otra, es algo muy frecuente dentro de la rama de la virtualización de hardware.
  Particularmente, este concepto es ampliamente usado en la emulación de consolas de videojuegos, en donde el emulador permite simular el comportamiento de la consola en cuestión, sobre otra arquitectura (otra consola distinta o una computadora).
- **Reutilizar hardware antiguo**: virtualizando un equipo viejo, se lo puede convertir en un conjunto de clientes livianos, removiendo así la necesidad de gastos en actualizaciones de workstations del sector de IT (tecnología de la información) de una compañía. Los clientes acceden a sus escritorios personales en el servidor, ocurriendo unas pocas diferencias notables, comparado con ejecutar el sistema operativo y sus aplicaciones localmente.
  Mediante la aplicación de algunas soluciones profesionales, como Citrix XenDesktop, es posible incluso que los empleados accedan a su escritorio personal desde sus hogares o dispositivos móviles.
- **Actualizar el hardware de los servidores sin ocasionar downtime**: combinar un servidor virtual con un dispositivo de almacenamiento compartido, permite realizar la migración de una máquina física a otra, sin dejar de ofrecer el servicio brindado ni un segundo (lo que se denomina formalmente como “independencia de hardware”).

3.2 - Administración remota del equipo

Para el despliegue de la arquitectura de máquinas virtuales, así como su administración y manutención, se decide usar el paquete de tecnologías Oracle VM. La decisión se basa en su gran penetración en el mercado profesional de virtualización, la documentación y soporte brindados, y la gran cantidad de características que posee todo el paquete de software que dispone.
Oracle VM es una plataforma que provee un ambiente totalmente equipado, con todos los últimos beneficios de las tecnologías de virtualización. Permite desplegar sistemas operativos y software de aplicación dentro de alguno de los ambientes virtuales soportados por la herramienta. El software provisto aísla a los usuarios y administradores de la tecnología de virtualización subyacente, a la vez que facilita las operaciones de rutina al ofrecer interfaces de usuario orientadas a objetivos concretos y comunes [OraVM14]. A continuación se muestran los componentes de una implementación usual de Oracle VM:

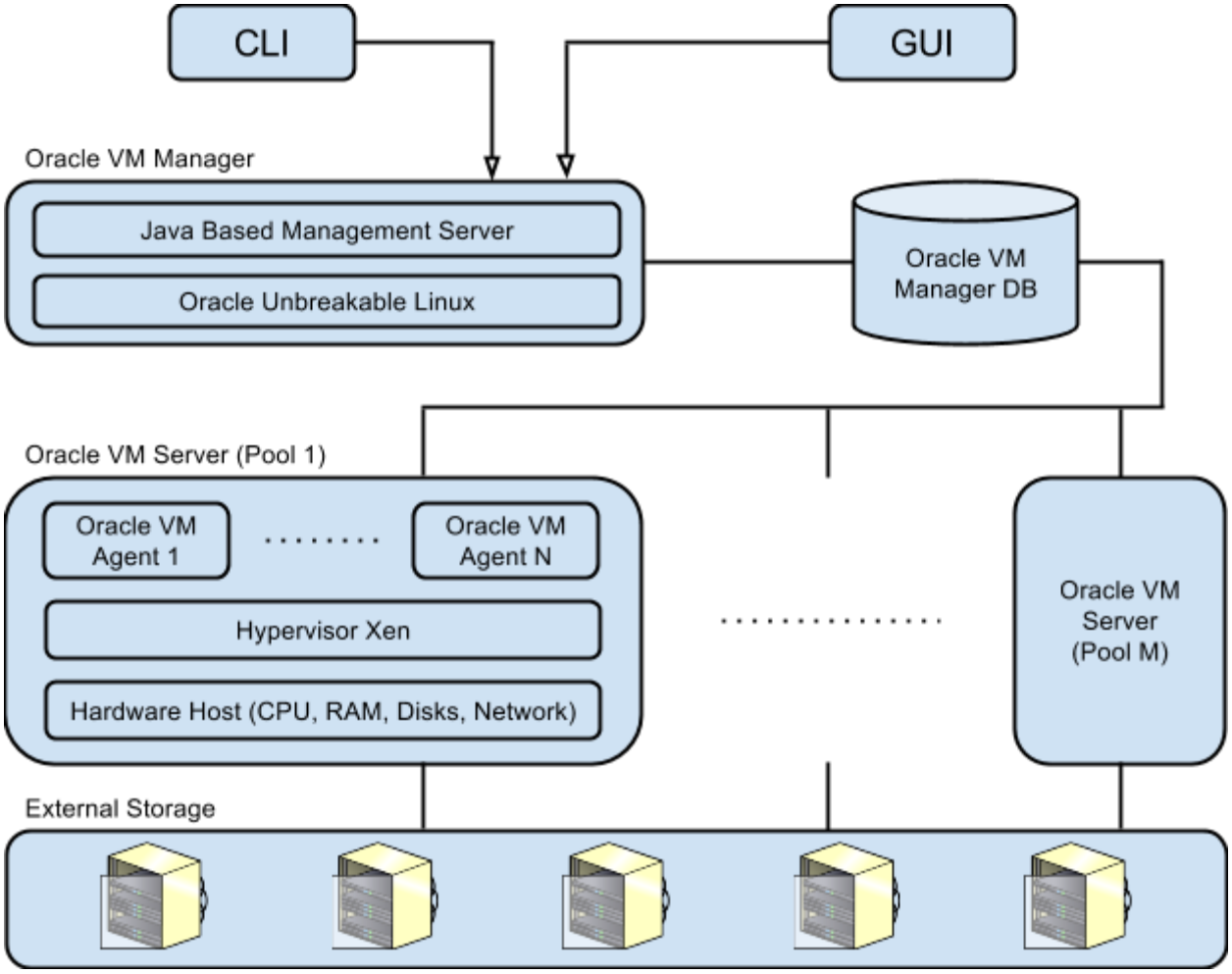

Figura 3.1: Arquitectura de hardware y software de una instalación típica de Oracle VM.

- **Oracle VM Manager**: provee una interfaz de línea de comandos (CLI), como así también una interfaz gráfica de usuario (GUI) basada en un navegador web, para realizar las tareas relativas a la administración de las máquinas virtuales y de toda la infraestructura de virtualización en general; a saber (pero no limitado a):
    - Configurar y administrar servidores Oracle VM (y pooles de servidores)
    - Configurar y administrar redes e infraestructura de red
    - Configurar y administrar el soporte de almacenamiento
    - Configurar y administrar recursos como imágenes de máquinas virtuales (imágenes de disco de tipo ISO), plantillas (templates) de máquinas virtuales, discos de instalación, etc
    - Crear máquinas virtuales a partir cualquiera de los recursos anteriores (a su vez, clonar máquinas virtuales)
    - Administrar las máquinas virtuales (algunas operaciones disponibles por ejemplo son el encendido y apagado, login, borrado de instancias y migración en caliente de las mismas)

    - Importar a la arquitectura corriente, máquinas virtuales creadas con Oracle VM u otras tecnologías de virtualización de servidores
    - Realizar balanceo de carga de las máquinas virtuales en los pooles de servidores
    - Administrar políticas de disponibilidad del servicio, de seguridad, de consumo de energía, etc.
- **Oracle VM Manager DB**: base de datos Oracle XE que mantiene toda la configuración del ambiente Oracle VM, tanto físico como virtual.
- **Oracle VM Server**: es un ambiente de virtualización administrado sobre un servidor físico, proveyendo una plataforma de servidor ligera y segura que ejecuta máquinas virtuales. Al menos se requiere un servidor Oracle VM, pero para hacer uso de las ventajas de clustering del software, se requieren varios (el clustering refiere a la técnica de enlazar o interconectar muchas computadoras entre sí, para actuar como si fueran una sola). Así es como se implementa la tecnología en la arquitectura desplegada para este trabajo (N servidores con M máquinas virtuales cada uno, donde M no necesariamente debe ser igual para todos los servidores).
  Cada servidor posee varios agentes (instancias virtuales) controlados por el Hypervisor de Oracle VM (Xen), que incluye el soporte para administrar el hardware subyacente del equipo físico.
- **External Storage**: capa abstracta que provee acceso a los dispositivos de almacenamiento disponibles para la arquitectura, así como las funcionalidades necesarias para descubrir, registrar y usar nuevo soporte de almacenamiento.

Un aspecto no menor es la seguridad en el acceso al ambiente de virtualización. Como toda red que pudiera llegar a transmitir y/o almacenar información de negocio crítica, es necesario implementar y mantener ciertas políticas de control de acceso para asegurar que los datos o recursos alojados en este ambiente, no sean vulnerados (eliminados, modificados de forma maliciosa, robados, etc).
Oracle VM ofrece mecanismos de seguridad para controlar el acceso y las operaciones realizadas sobre el ambiente virtualizado. Los usuarios de este software, son administradores que mediante el Oracle VM Manager y su interfaz de línea de comandos pueden configurar y mantener el ambiente físico y virtual. Los elementos esenciales que definen el nivel de seguridad en Oracle VM son [OraVM14]:

- La instalación en sí del ambiente, siguiendo la guía oficial para realizarlo de modo de implementar la mayor cantidad de prácticas de seguridad
- El control estricto de los privilegios administrativos
- Segregación de la red e isolación de los equipos físicos, para prevenir la exposición de los servicios no requeridos
- Separar el acceso de los usuarios finales (los desarrolladores de software para el caso de uso implementado en este trabajo) a las máquinas virtuales, e isolación de las máquinas virtuales de la infraestructura subyacente de Oracle VM

3.2.1 - Acceso de usuarios

Las cuentas de administrador poseen acceso completo a toda la funcionalidad y todos los recursos controlados a través de Oracle VM Manager y sus herramientas de administración (como la GUI). Aún así, es altamente recomendable que tan sólo sean asignadas unas pocas cuentas administrativas a las personas que son responsables de la configuración y de la administración de rutina del ambiente. Los administradores también deben tener acceso a los sistemas operativos de las instancias virtuales, y para ello deben usar la consola de la máquina virtual provista por la interfaz de Oracle VM Manager. Obviamente, no todo usuario con acceso a alguna máquina virtual debe ser un administrador. Esto violaría el principio del mínimo privilegio, que propone que en un ambiente computacional, cada módulo (como ser procesos, usuarios, o programas) debe tener permitido tan sólo el acceso a la información y recursos que son necesarios para su legítimo propósito [Saltzer75 y Denning76].

Dependiendo del despliegue en particular de la arquitectura de Oracle VM, se querrá asignar permisos de acceso a las máquinas virtuales de diferentes modos, dentro de los siguientes tres métodos de control de acceso a las mismas:

- **Consola de Oracle VM Manager**: es el método estándar para conectarse a una máquina virtual alojada en el ambiente de Oracle VM, y el que puede ser usado en cualquier momento por un administrador. Dado que las máquinas virtuales, en esencia, son servidores que ofrecen a los usuarios finales aplicaciones y servicios con los que interactuar, estos rara vez ingresan (se logean) a la máquina virtual en sí, por lo que no requieren de este servicio.
  La lógica sugiere que desde el punto de vista de la seguridad, tan sólo unas pocas cuentas administrativas posean acceso a las máquinas virtuales a través de esta consola, mientras que los recursos de Oracle VM se mantienen ocultos y protegidos del resto de los usuarios. Para el caso implementado en este trabajo, tan sólo las personas encargadas de la instalación y mantenimiento de la infraestructura de red en el centro de desarrollo de software, poseen este tipo de acceso.
- **Conectividad o acceso remoto directo**: si ciertos usuarios requieren acceso administrativo a las máquinas virtuales, pero no son administradores en sí, lo recomendable no es crear más cuentas administrativas. En su lugar, un administrador debería configurar la máquina virtual para que se pueda establecer una conexión remota, sin la necesidad de acceder a través de Oracle VM Manager; es decir, facilitar el acceso de línea de comandos mediante SSH a la misma.
  Este es el método de acceso por defecto otorgado a los desarrolladores de software, en la implementación de la arquitectura de desarrollo remoto de este trabajo; debido a la conveniencia y consideraciones de seguridad descritas anteriormente (además, la mayoría de los desarrolladores de software saben cómo interactuar con una computadora remota, a través de la línea de comandos).
- **Acceso basado en roles**: los despliegues a gran escala de Oracle VM tienen requerimientos diferentes en lo que respecta a administración de usuarios y control de

acceso. A medida que la cantidad de máquinas virtuales crece a razón de cientas o miles de instancias, se torna inmanejable la combinación de los dos métodos de acceso anteriores. Esto se debe a que diferentes categorías de usuarios requerirán de distintos niveles de acceso a grupos virtualizados de recursos. En un ambiente así de complejo, se requieren facilidades de control de acceso basadas en roles, y la integración de un servicio de directorio de usuarios (como por ejemplo, “Active Directory” de Microsoft).
Debido a que la arquitectura desplegada en la implementación de este trabajo no es así de grande y compleja, este método de acceso no se implementó. Aún así, se presenta la noción del mismo para contemplar el caso de infraestructuras mayores, y de cómo deberían manejar el acceso en este último caso.

3.2.2 - Consideraciones de seguridad

Desde muchos puntos de vista, los servidores virtuales poseen los mismos requerimientos de seguridad que los servidores físicos. Lo mismo aplica a las aplicaciones y servicios que estos brindan. La virtualización provee ciertos beneficios sobre la seguridad: cada máquina virtual posee un contexto de seguridad privado, potencialmente con reglas de autenticación y autorización separadas, y con procesos separados. Desplegar aplicaciones en máquinas virtuales separadas, provee un mejor control sobre la seguridad, comparado a ejecutar múltiples aplicaciones sobre el mismo sistema operativo: comprometer el sistema operativo de una máquina virtual (que exista una brecha en la seguridad del mismo) no necesariamente compromete las cargas de trabajo y los datos que residan en otras máquinas virtuales. De todos modos, algunas prácticas de seguridad deben mantenerse en consideración para prevenir que la misma virtualización introduzca vulnerabilidades en la seguridad del sistema [OraVM14].

Un aspecto es la seguridad física. La infraestructura virtual no es tan “visible” como la infraestructura física. Equipos físicos sirviendo instancias virtualizadas de trabajo, deben ser aislados en los datacenters, del mismo modo que el resto de los servidores con requerimientos de seguridad extremos. Incluso sin áreas físicas segurizadas, muchas compañías mantienen cargas de trabajo y de datos de diferentes clases de niveles seguridad en distintos servidores. Las mismas reglas de aislamiento deben aplicarse para las máquinas virtuales. Para el caso implementado a través de OracleVM, esto implica mantener pooles de servidores separados, cada uno con su propio grupo de servidores.

Estas reglas de aislamiento deben ser aplicadas también a la infraestructura de red y a las conexiones entre los equipos; ya que no hay indicadores visuales (no hay cables) que ayuden a asegurar que el tráfico de aplicación, administración y respaldo (backup) se mantengan funcionales y separados. Así que en lugar de conectar cables de red en interfaces y switches distintos, el administrador de OracleVM debe asegurarse de que las interfaces virtuales de red, se encuentren conectadas a redes virtuales separadas (VLAN,

Virtual Local Area Network) las unas de las otras. Asimismo, asignar redes virtuales separadas para el tráfico diferenciado sobre las máquinas virtuales (administración, almacenamiento, backup, etc.). Todo esto puede controlarlo el administrador desde la consola de OracleVM.

También deben ponerse en consideración cuidados adicionales sobre las imágenes de disco de las máquinas virtuales. En muchos casos, estas se encuentran disponibles sobre la red para, por ejemplo, propósitos de migración. Usualmente, estas imágenes son archivos, los cuales pueden ser copiados o robados fácilmente, si la seguridad en el hardware de almacenamiento de la red es comprometida. Por ende, es esencial restringir los ambientes virtuales y prevenir los accesos no autorizados. Un intruso con acceso de root a una máquina virtual en la red de almacenamiento, podría copiar, borrar o alterar sus contenidos. Deben usarse redes separadas para la transmisión entre los servidores de almacenamiento y los equipos virtuales, para asegurarse de que el tráfico no sea público y sujeto a ser comprometido.

Todos estos pasos de securización requieren acceso de control sobre la red virtual (a través del Oracle VM Manager). Los accesos a estos equipos deben realizarse tan sólo a través de redes privadas y las cuentas de usuarios con permisos para loguearse en los servidores, deben ser rigurosamente controladas, y limitadas a la menor cantidad posible de individuos.

3.3 - Sincronización de recursos

Como se mencionó con anterioridad, mantener consistentes las copias del software ejecutando en ambientes distintos a lo largo de la arquitectura desarrollada, es todo un problema en sí. Esto se vuelve particularmente crítico con las copias de código fuente de cada desarrollador, teniendo en cuenta que es normal un escenario en donde cada desarrollador se encuentre editando segmentos o módulos distintos del código fuente aplicativo, e incluso, realizando sucesivas ediciones en períodos cortos de tiempo. Todo esto con la necesidad de ver el resultado de las modificaciones realizadas, de modo instantáneo en su equipo.

Dado que, bajo la arquitectura de software remoto desarrollada, el servidor web que ejecuta la aplicación en el entorno de desarrollo del programador, ya no corre sobre su propia máquina local, sino sobre una instancia virtual remota; hay que trasladar toda la copia de código fuente hacia este equipo virtual, el particular de cada desarrollador. Con cada cambio significativo, que el desarrollador considere que ya está en condiciones de ver aplicados los cambios realizados, hay que volver a migrar el código fuente al servidor web remoto. Esto puede suceder concurrentemente entre muchos desarrolladores de forma sucesiva en el tiempo.

Es así, que tras análisis y pruebas de las opciones posibles para atacar este particular problema (ver capítulo 5.3.1), se selecciona una solución de software a aplicar en ambas locaciones de la arquitectura de software (el broker local, y la máquina virtual remota), que resuelve eficientemente el problema, sobre la base de que esta solución:

- Debe poder ser ejecutada e instalada fácilmente y de forma automatizada, tanto en el entorno local del desarrollador, como en la instancia virtual remota.
- Debe poder ser invocada bajo demanda del desarrollador, para no consumir constantemente recursos valiosos de hardware y degradar la performance general de toda la arquitectura.
- Debe ser eficiente en términos de consumo de recursos, de modo de que un proceso de sincronización de código fuente, no anule ni degrade las capacidades de respuesta del equipo del desarrollador que solicitó la sincronización (tanto su equipo local como su máquina virtual remota).
- Debe ser eficiente en términos de uso de la red o cantidad de datos transferidos y ancho de banda consumido; de modo que procesos de sincronización concurrentes de varios desarrolladores no perjudiquen la performance de toda la red, o del cluster de máquinas virtuales.
- Debe ser lo suficientemente rápida, para que el desarrollador pueda probar sucesivas modificaciones pequeñas de código fuente y ver inmediatamente los resultados obtenidos con sus cambios.

Por estas consideraciones, es que se desarrolla la solución basada en un algoritmo de “delta encoding”, realizando sincronización incremental de una sola vía, capacidad provista por el software Rsync (ver capítulo 5.3.2). Estos términos y la base teórica y técnica correspondientes se detallan a continuación.

3.3.1 - Delta Encoding

“Delta Encoding” es una forma de almacenar o transmitir datos en la forma de diferencias entre datos secuenciales en lugar de archivos completos (esta técnica también es conocida como “compresión delta” o “compresión diferencial”, particularmente cuando es utilizada en el contexto de “compresión compacta” de archivos o en software de control de versiones de archivos).

Una “función delta” hace referencia a la diferencia entre dos instancias de un determinado objeto, en el caso de los archivos (a transmitir entre distintas fuentes, almacenar en un histórico de cambios o respaldar en un proceso de backup), una función delta para estos refiere a la diferencia que existe dentro de un mismo archivo en dos versiones distintas. Las diferencias son almacenadas en archivos discretos llamados “deltas” o “diffs”, los cuales mantienen la secuencia de operaciones de cambios elementales que, aplicados a una versión del archivo, resultan en la versión sucesiva del mismo (nótese la correspondencia

con los “transaction logs” de las bases de datos relacionales).

El algoritmo de Delta Encoding tiene como objetivo obtener solamente los bytes que han sido modificados desde la última versión del archivo, permitiendo reducir considerablemente el tamaño de éste y lograr una optimización del uso de la red al momento de realizar la transferencia o respaldo.
En situaciones en donde las diferencias son pequeñas (por ejemplo cambios en unas pocas palabras de un documento de texto, cambios a unas pocas líneas en archivos de código fuente o modificaciones en algunos registros de una gran tabla de base de datos), la técnica de Delta Encoding reduce enormemente la redundancia de datos. Las colecciones de deltas “únicos” resultan sustancialmente más eficientes en espacio, que sus equivalentes no procesados.

Una vez que se tiene comprimido un archivo, se puede obtener el archivo original teniendo la versión de referencia del archivo y el archivo generado por el algoritmo de Delta Encoding (desde un punto de vista lógico, la diferencia entre los dos archivos, es la información requerida para obtener uno en base al otro).

3.3.2 - Sincronización incremental

Se llama “codificación incremental” (incremental encoding) a una variación del algoritmo de delta encoding, el cual codifica las diferencias entre los prefijos o sufijos de las cadenas de texto dentro del archivo a procesar (almacena los prefijos o sufijos comunes y sus longitudes, de modo que no requieran duplicarse en el archivo procesado). A un algoritmo de sincronización de archivos que realice el proceso mediante codificación incremental, se dice que es un algoritmo de sincronización incremental. Este método es particularmente efectivo para listas de textos ordenadas con pequeñas diferencias entre las cadenas adyacentes (como lo es por ejemplo un diccionario).

3.3.3 - Ejemplos de usos

Debido a la facilidad de implementación de las técnica de delta encoding y de sincronización incremental, y su versatilidad para ser llevadas a la práctica en diversos campos, es que son bastamente aplicadas en casi cualquier área de la computación o de transmisión de datos digitales en general entre diversas fuentes. Se citan aquí algunos ejemplos de la vida real para ilustrar el por qué en la selección de un proceso de sincronización que hace uso de estas técnicas, los beneficios que conlleva, y la adopción a lo largo de una amplia variedad de temáticas y áreas de implementación.

- **Actualizaciones/patches de software**: “Binary Delta Compression” es una tecnología usada en el proceso de despliegue de software para la distribución de

patches (actualizaciones de programas de software que usualmente reparan bugs o vulnerabilidades, o bien mejoran la usabilidad o performance del software). Esta tecnología permite una gran reducción en el tamaño de descarga de las actualizaciones, transfiriendo sólamente las diferencias entre los archivos viejos y los nuevos durante el proceso de actualización; logrando disminuir la cantidad de datos descargados de internet y por ende, los problemas de tráfico de red correspondientes.

- **Codificación de archivos de audio y video**: el formato de archivos de audio 8SVX (8-Bit Sampled Voice), aplica delta encoding a los datos de sonido planos antes de aplicarles compresión. Esta técnica suele lograr una buena relación de disminución de tamaño en muestras de sonido de 8 bits. Para muestras con bitrate (número de bits que se transmiten por unidad de tiempo a través de un sistema de transmisión digital) de 16 bits (formato 16SVX) o superiores, la usabilidad del delta encoding disminuye, pero afortunadamente, los algoritmos de compresión a menudo eligen aplicar delta encoding sólo cuando se logra una versión reducida en tamaño con la compresión.
  En la compresión de video, el delta encoding reduce considerablemente las diferencias entre los frames (cuadros de película) adyacentes, logrando disminuir así el frame size (tamaño de cada frame) y consiguiendo entonces altas tasas de compresión. Esta técnica es usada prácticamente en todos los códecs (especificación capaz de transformar un archivo con un flujo de datos o una señal) de compresión de video.
- **Índices de motores de búsquedas**: la codificación incremental es comúnmente usada durante el proceso de recuperación de información en búsquedas de texto, para comprimir los lexemas presentes en los documentos indexados. Estos índices listan todas las palabras de todos los documentos, junto con un puntero en cada una hacia una lista de locaciones con sus ocurrencias. Generalmente, este algoritmo de codificación, logra comprimir los índices a razón de un 40% [Witten99].
- **GNU locate**: la utilidad de búsqueda de archivos en sistemas de archivos de Unix, también hace uso del algoritmo de codificación incremental, como punto de partida en un índice que mantiene con nombres de archivos y directorios, para reducir el espacio requerido al almacenar los prefijos en las rutas más populares (o con mayor cantidad de archivos).
- **Delta encoding en HTTP**: otra posible instancia de uso del delta encoding es en la transmisión de páginas web. Actualmente existe un esfuerzo de desarrollo de una norma que propone que los servidores HTTP puedan ser capaces de enviar a los clientes (navegadores web) páginas web actualizadas, en la forma de diferencias entre versiones (deltas), lo cual debería disminuir el tráfico de internet, dado que la mayoría de las páginas cambian poco o lentamente en el tiempo, en lugar de ser complementamente re-escritas de modo repetitivo [RFC3229]. Así, las solicitudes causarían sólo la descarga de instancias de recursos con pequeñas modificaciones, sobre la versión que el cliente ya tuviera almacenada en su caché local.
- **Sistemas de control de versiones de código fuente**: Git utiliza compresión mediante delta encoding para ahorrar espacio en disco cuando los archivos de código fuente o de datos cambian incrementalmente entre sucesivos commits.

- **Utilidades de comparación de archivos**: muchos formatos, algoritmos y programas de comparación de datos o archivos (tanto de texto como binarios), utilizan delta encoding como parte de su especificación o implementación para lograr altas tasas de compresión en los resultados. Algunos ejemplos son VCDIFF, open-vcdiff, Xdelta, GDIFF, Diff y bsdiff.

3.3.4 - El caso puntual de Rsync

En las transmisiones a través de una red, codificadas mediante el algoritmo de delta encoding, en donde tan sólo una copia del archivo se encuentra disponible en cada extremo del canal de comunicación, suelen usarse códigos de control de errores especiales para determinar qué partes del archivo han sido alteradas desde su última versión. En este contexto, Rsync utiliza una forma de delta encoding mediante un algoritmo de “rolling checksum” (una función de hash en donde la entrada es calculada rápidamente en una ventana o segmento del archivo que se va moviendo a través de toda la entrada en forma cíclica) basado en el método checksum adler-32 de Mark Adler [RFC1950]. Este método es utilizado por Rsync para determinar qué archivos y más específicamente, qué partes de cada archivo cambiaron, y así minimizar la transferencia de datos a través de la red [Maxino06].

Así, el cliente de Rsync (la copia del aplicativo que invoca la sincronización) segmenta cada archivo en la memoria en un conjunto de bloques de tamaño fijo, y calcula el rolling checksum de cada bloque, para luego enviar la lista de checksums a la copia servidor de Rsync. El servidor puede entonces buscar en la instancia corriente aquellos bloques que posean checksums que concuerden con los de la lista y enviar de regreso una codificación de dicha instancia que no incluye ninguno de los bloques que ya posee el cliente.
El particular cálculo de checksum que realiza Rsync, es lo suficientemente versátil para permitir al servidor comparar el patrón de los bloques que aparezcan en desplazamientos arbitrarios en la nueva instancia, de modo que la codificación usada puede ser muy compacta en tamaño, si el cambio fuera una pequeña inserción o remoción de datos (o texto en el caso de código fuente).

Como se ve, Rsync requiere enviar un bloque entero al servidor, aunque tan sólo un byte de dicho bloque hubiera sufrido alteraciones. Podría reducirse el impacto usando tamaños de bloque más pequeños, pero entonces esto requeriría calcular y enviar una lista de checksums más grande. Aún así, Rsync logra el balance adecuado al ser más fácil de implementar que realizar el procesamiento manual de delta encoding, y logrando una eficiencia y velocidades considerables en la transmisión de pequeñas y sucesivas diferencias entre archivos [Mogul00].

La eficiencia y versatilidad de Rsync, lo han llevado a estar presente “de fábrica” en la mayoría de las distribuciones de sistemas operativos basados en Unix, como distribuciones de Linux y OS-X de Apple. De hecho la adopción de este software va mucho más allá de

simplemente ser brindado como una utilidad del sistema operativo, a ser un requisito para la correcta instalación o configuración de piezas de software populares de la actualidad.
Hadoop es un caso de uso digno de mención de aplicación de Rsync.

Hadoop es un framework de software libre, diseñado para almacenar y procesar grandes conjuntos de datos a gran escala (literalmente, permite trabajar con miles de nodos y petabytes de datos), dispersos en clusters de “commodity hardware” (gran cantidad de componentes de computación ya disponibles en una arquitectura o instalación de hardware corriente, para realizar cómputo paralelo obteniendo el mayor throughput posible a bajo costo [Dorband10]). Hadoop implementa un paradigma computacional llamado Map/Reduce para procesar los datos y obtener los resultados solicitados. Mediante Map/Reduce, la aplicación se divide en muchos pequeños fragmentos de trabajo, cada uno de los cuales se pueden ejecutar o volver a ejecutar en cualquier nodo del clúster. Además, proporciona un sistema de archivos distribuido que almacena los datos en los nodos de cómputo, produciendo un alto ancho de banda agregado en todo el clúster. Ambos, Map/Reduce y el sistema de archivos distribuidos, están diseñados de manera que las fallas de nodo se gestionan automáticamente mediante el framework de software. Esta es una característica importante del software, ya que todos sus módulos están diseñados bajo la asunción fundamental de que las fallas de hardware son comunes, y este escenario debe ser automáticamente manejado y resuelto por el mismo framework, que proporciona a las aplicaciones de forma transparente, fiabilidad y movilidad de datos [Hadoop14].
Hadoop no posee una ubicación única y centralizada para sus archivos de configuración. En su lugar, cada nodo Hadoop en el cluster posee su propio conjunto de archivos de configuración, y se delega a los administradores el asegurar que dichas copias se mantengan sincronizadas a través de todo el sistema. Hadoop provee facilidades rudimentarias para sincronizar las configuraciones mediante Rsync (alternativamente, también ofrece scripts y herramientas de línea de comandos que pueden ayudar en dicha tarea). Los scripts de control de Hadoop pueden distribuir los archivos de configuración a todos los nodos del cluster mediante Rsync (esta opción no viene habilitada por defecto, pero puede definirse fácilmente en el archivo principal de configuración del nodo master de Hadoop, de forma de que cada uno de los worker daemons sincronicen el árbol de archivos bajo el nodo master, al nodo local de su instalación cuando el demonio es inicializado) [White12]. Vale la pena mencionar, que un demonio (o en inglés, daemon) es un servicio, proceso o programa residente no interactivo, es decir que se ejecuta en segundo plano en lugar de ser controlado por el usuario.

3.4 - Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado (o IDE, por sus siglas en inglés para “Integrated Development Environment”) es una aplicación de software que provee a los desarrolladores, de un conjunto de utilidades de fácil comprensión para el desarrollo de software de aplicación

(un marco de trabajo amigable), para un lenguaje de programación específico o bien para varios. En este sentido, un IDE se distribuye como una sola pieza de software o paquete que usualmente consiste de un editor de código fuente (un editor de textos con capacidades avanzadas de edición), herramientas de construcción automatizadas (algunos contienen un compilador, un intérprete, o ambos, aunque otros ninguno de los dos) y un depurador de código. La mayoría de los IDEs modernos también ofrecen características inteligentes como herramientas de construcción de interfaces visuales de usuario (GUI), auto-completado de código y sugerencias estáticas para la mejora del código escrito, en términos de su manutención y eficacia de ejecución. En ocasiones se incluye soporte para sistemas de control de versiones de código fuente (programas de versionado); y en aquellos IDEs modernos con soporte para lenguajes orientados a objetos, también se suele incluir navegadores de clases y objetos, o diagramas automatizados de la jerarquía de clases que componen las aplicaciones construidas. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva; sin la necesidad de trabajar editando archivos de texto de código fuente, sino instruyendo directamente a un intérprete de línea de comandos para realizar las tareas requeridas.

Los IDEs se diseñan de forma de maximizar la productividad del desarrollador de software, proveyendo al mismo de componentes altamente acoplados y con interfaces de usuario similares. Es decir que se presentan como un único programa en donde todo el desarrollo pueda ser realizado (todas las fases del ciclo de vida del desarrollo de software, convirtiéndose en una herramienta I-CASE). Esto contrasta con el desarrollo de software mediante herramientas no interrelacionadas, como aún realizan muchos desarrolladores por ejemplo en entornos Unix; utilizando Vim (editor de textos), GCC (compilador) y Make (scripting de construcción o despliegue del software); todos programas altamente difundidos y efectivos, pero que no se encuentran acoplados en el sentido de facilitar su utilización conjunta. Aún así, los programadores de Unix pueden combinar las herramientas POSIX de línea de comandos (diversos programas que son provistos nativamente en toda instalación de sistemas operativos basados en Unix) como si fueran un completo entorno de desarrollo, capaz de desarrollar grandes programas como el kernel Linux y su ambiente de ejecución [Rehman02]. Además, en los últimos años, esto ha trascendido la frontera de Unix llegando a otras plataformas también, debido a que las herramientas de software libre de GNU (como GNU Compiler Collection, GNU Debugger y GNU Make entre otras) se encuentran ya disponibles para muchas más plataformas, incluyendo Windows [GNUEmacs13].

Uno de los principales propósitos de los IDEs, es reducir la configuración necesaria para distribuir conjuntamente múltiples utilidades de desarrollo, en lugar de proveer el mismo conjunto de capacidades como una unidad coherente. Reducir el tiempo de configuración del entorno de desarrollo puede incrementar la productividad del programador, en los casos en los que aprender a utilizar el IDE sea más rápido que integrar manualmente todas las herramientas individuales. Una mayor integración de todas las tareas de desarrollo, logran el potencial de mejorar toda la productividad en su conjunto, en lugar de tan sólo facilitar las

tareas de configuración del ambiente.

3.4.1 - Eclipse como I-CASE

Eclipse es un IDE que ofrece un espacio de trabajo base predefinido, y un sistema de plug-ins extensible para personalizar el ambiente usado para desarrollar aplicaciones. Está escrito principalmente en Java, lo que lo hace multiplataforma y le permite ser ejecutado en prácticamente cualquier arquitectura para la cual está provista una Máquina Virtual de Java (JVM). Si bien, el kit de desarrollo de software de Eclipse (Eclipse SDK), originalmente, fue concebido para el desarrollo de software sobre la plataforma de Java; los usuarios pueden extenderlo a través de plug-ins escritos sobre su plataforma. De esta forma puede ser usado además, para desarrollar aplicaciones en otros lenguajes de programación, como por ejemplo (pero sin limitarse a) Ada, C, C++, Clojure, COBOL, Erlang, Fortran, Groovy, Haskell, JavaScript, Perl, PHP, Prolog, Python, Ruby, Scala y Scheme. Debido a su naturaleza modular y a la versatilidad y variedad de lenguajes para los que ofrece soporte a través de plug-ins, Eclipse se provee en diversos paquetes ya personalizados (con los plug-ins necesarios ya instalados) para el desarrollo de software para diversas plataformas de ejecución (o lo que es lo mismo, podría decirse que se ofrecen distintas versiones del IDE según la plataforma sobre la que se desea desarrollar). Algunos de estos paquetes de Eclipse son: Eclipse Java Development Tools (JDT) para desarrollar sobre Java, Eclipse CDT para el desarrollo sobre C/C++, Eclipse PDT para PHP, Eclipse ADT para el desarrollo de aplicaciones para la plataforma móvil Android, entre otros.
Tal y como lo definieron sus creadores inicialmente, “Eclipse es una plataforma buena para todo y para nada en particular, cuyo objetivo es soportar herramientas integradas de desarrollo, contenido arbitrario, aplicaciones de proveedores externos al proyecto e interfaces de usuario” [EMission].

3.4.2 - Arquitectura expandible de plug-ins

Los plug-ins son piezas de software usadas para agrupar código ejecutable en unidades modulares, extensibles y fácilmente distribuibles. Específicamente en el lenguaje de programación Java, un plug-in técnicamente es un Archivo Java (JAR) que es auto-contenido y auto-descriptivo (self-contained y self-describing en inglés). Es **self-contained** porque empaqueta todo el código fuente y demás recursos que el plug-in requiere para ser ejecutado. Es **self-describing** porque además, incluye información acerca de qué es dicho plug-in en sí, qué requiere del mundo exterior para su correcto funcionamiento, y qué le contribuye al mundo [Aniszczyk08].

Eclipse utiliza plug-ins para proveer toda la funcionalidad escrita sobre su sistema de ejecución, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. De hecho, con la excepción de su pequeño kernel

(núcleo del sistema), todo en Eclipse es un plug-in. Esto significa que todo plug-in desarrollado por terceras partes, se integra con Eclipse exactamente de la misma forma que los plug-ins "oficiales", o aquellos que son distribuidos en conjunto con el mismo software.

El Entorno de Desarrollo de Plugins de Eclipse (o PDE por sus siglas en inglés para Plug-in Development Environment) provee un conjunto comprensivo de herramientas para crear, desarrollar, testear, debuggear, construir y desplegar plug-ins, fragmentos de software, herramientas, características y actualizaciones para Eclipse. Así, además de poder usar la plataforma de Eclipse para el desarrollo de software sobre varios lenguajes, puede extenderse para trabajar sobre otros espectros (a través de plug-ins desarrollados por terceras partes, tanto libres como comerciales); como con sistemas de composición de textos como LaTeX [TeXlipse14], soporte a lenguajes de modelado como UML, aplicaciones de networking como Telnet, sistemas de administración de bases de datos (DBMS) como DB Explorer, sistemas de control de versiones de software como CVS, etc. La arquitectura de plug-ins es lo suficientemente amplia para soportar la escritura de casi cualquier extensión deseada para el ambiente.

El componente PDE de Eclipse provee un avanzado mecanismo para testear y debuggear los plug-ins a desarrollar o desarrollándose en un entorno de ejecución real del mismo Eclipse (esto usualmente es llamado **self-hosting** en computación). La instancia desde la que se lanza Eclipse es el host, mientras que la instancia lanzada es el **runtime workbench** (entorno de trabajo de tiempo de ejecución) [Alwis02]. De esta forma, uno puede escribir plug-ins sobre la plataforma, y testearlos y debuggearlos inmediatamente sobre una instancia real y totalmente funcional de Eclipse sobre la cual el plugin desarrollándose está instalado, pero siendo depurado desde la instancia original en la que el desarrollador escribe el código.

Los componentes del framework de PDE se dividen en tres categorías principales:

- PDE Build: facilita la automatización de procesos de construcción de plug-ins, produciendo scripts para Ant (aplicativo para mecanizar procesos de compilación y/o construcción de software) basados en información provista en tiempo de desarrollo.
- PDE UI: modelos, asistentes, editores y otras herramientas para facilitar el desarrollo de plug-ins en el IDE Eclipse.
- PDE API Tools: herramientas integradas para el proceso de construcción, a fin de mantener y documentar una completa API (Interfaz de Programación de Aplicaciones) de Eclipse para el desarrollador de software.

3.4.2.1 - Extensiones y puntos de extensión

Una regla básica para desarrollar sistemas de software modulares y extensibles, es evitar el alto acoplamiento entre componentes. Si los componentes se encuentran estrechamente integrados, se torna difícil ensamblar las piezas en diferentes configuraciones, o reemplazar un componente por otra implementación distinta, sin causar una catarata de cambios a través

de todo el sistema.
Eclipse logra deshacerse parcialmente del acoplamiento entre componentes, a través del **mecanismo de extensiones y puntos de extensión**. La metáfora más simple para describir estos conceptos son los tomacorrientes eléctricos hogareños. El enchufe es el punto de extensión y el conector o aparato que se conecta a él es la extensión. Al igual que los tomacorrientes, los puntos de extensión vienen en distintos tamaños y formas, y sólo las extensiones que están diseñadas para un punto de extensión en particular, encajarán en él.
Cuando un plug-in requiere permitir a otros plug-ins que lo extiendan o personalicen parte de su funcionalidad, entonces declara un punto de extensión. Este último especifica un contrato, generalmente una combinación de marcado XML e interfaces Java, que las extensiones deben concretar (los plug-ins que deseen conectarse a dicho punto de extensión, deben implementar el contrato del mismo). El punto principal de esto, es que el plug-in que está siendo extendido desconoce en lo absoluto los detalles acerca del plug-in que se está conectando a él, más allá del alcance del contrato de interfaz del punto de extensión. Esto permite que los plug-ins que sean desarrollados por diferentes personas o compañías, puedan interactuar a la perfección, incluso desconociendo muchos aspectos el uno del otro. Un plug-in puede proveer uno o más puntos de extensión, de modo que otros plug-ins puedan agregar funcionalidad sobre él; además, puede proveer extensiones para conectar a otros plug-ins existentes.

La plataforma de Eclipse incluye muchas aplicaciones concretas de los conceptos de extensiones y puntos de extensión. Algunas extensiones son meramente declarativas, en la forma en que no aportan código ejecutable en lo absoluto. Por ejemplo, un punto de extensión provee configuraciones editables para atajos de teclado, y otro define anotaciones personalizadas para archivos (llamadas “markers” en Eclipse); ninguno de estos puntos de extensión requieren de código implementado en sus extensiones.
Otra categoría de puntos de extensión, son aquellos que permiten sobrescribir el comportamiento por defecto de un componente. Por ejemplo, las JDT (Java Development Tools) de Eclipse incluyen un formateador de código fuente, pero a su vez también ofrecen un punto de extensión para que formateadores de código de terceras partes puedan ser desarrollados. El plug-in “resources” (recursos) de Eclipse posee un punto de extensión que permite a ciertos plug-ins reemplazar la implementación de algunas operaciones básicas de archivos, como su copia, traslado o borrado.
Por último, otra categoría de puntos de extensión que vale la pena mencionar (y que como se verá en el capítulo 5.4, se hace uso extensivo de estos puntos de extensión para la implementación de la herramienta CASE de este trabajo), es aquella que agrupa elementos relacionados en la interfaz del usuario. Esto es, puntos de extensión para proveer vistas de usuario, editores de texto, y asistentes (wizards) en la interfaz del IDE. Esta categoría de puntos de extensión, permite al plug-in base de interfaz de usuario, agrupar características comunes (como ser el ubicar todos los asistentes de importación de proyectos en un único cuadro de diálogo), y definir una forma consistente de presentar las contribuciones de interfaz de usuario, proporcionadas por una gran variedad de otros plug-ins.

4 - Diseño de la herramienta CASE

Antes de abocarse de lleno en la implementación de la herramienta CASE, que cumple con los objetivos descritos, y siguiendo la metodología idónea de la ingeniería de software, se debe transitar por la fase de diseño del software requerido desarrollado.
Para esto se muestra y describe en el siguiente diagrama, el diseño general de la arquitectura, poniendo énfasis en qué aplicaciones y responsabilidades son puestas en cada extremo de la arquitectura final montada (entorno de desarrollo local del desarrollador de software, versus entorno remoto de máquina virtual) y cómo se realiza la interacción entre ambas plataformas; prestando también especial atención en qué papel juega la herramienta CASE desarrollada en este trabajo, en dicha arquitectura:

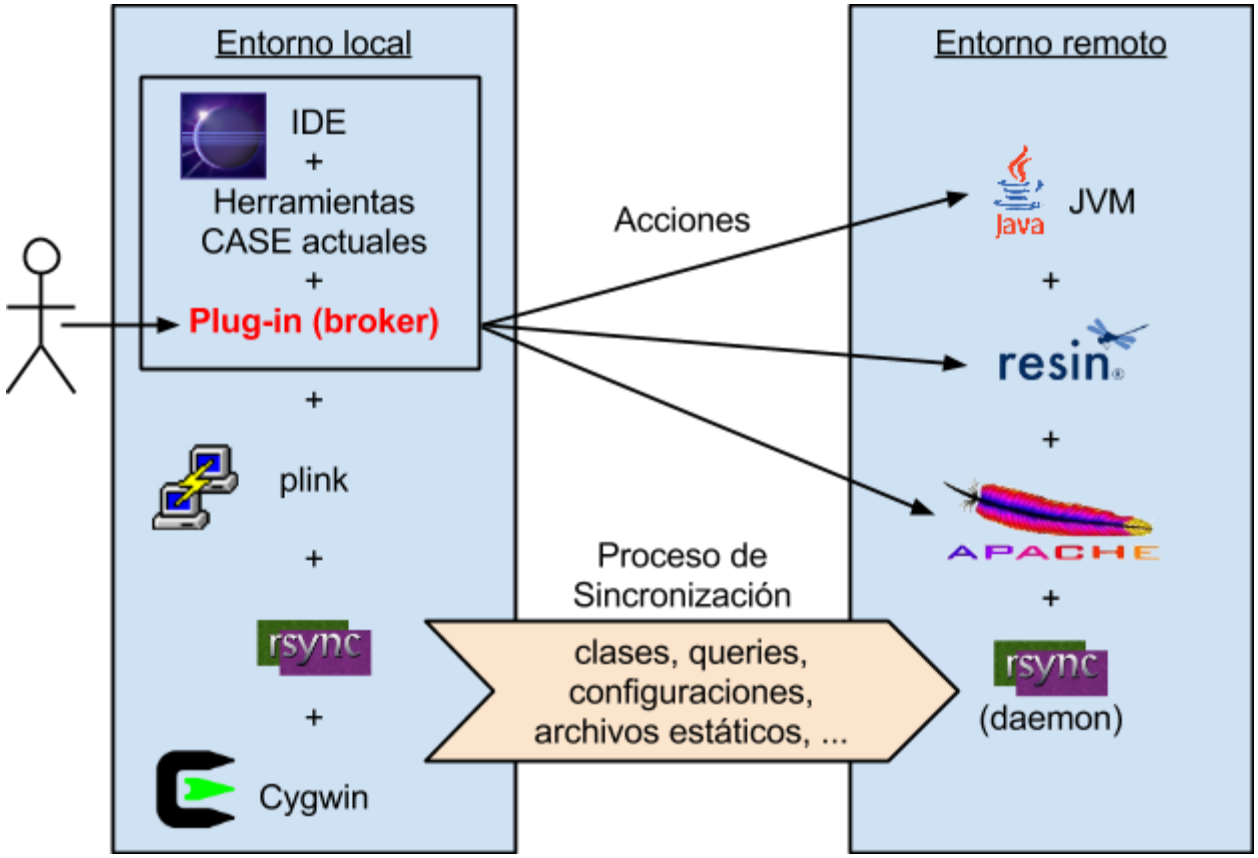

Figura 4.1: Diseño general de la arquitectura de la herramienta CASE.

Aquí es entonces donde se aprecia claramente cuáles herramientas del actual entorno de desarrollo del desarrollador de software (workstation) son llevadas a la máquina virtual remota, y cuáles herramientas extra fueron necesarias instalar en ambos entornos; además de la herramienta CASE desarrollada para servir de broker a toda la arquitectura.

El desarrollador mantendrá en su máquina local el IDE (Eclipse para este caso de aplicación) que habitualmente usa, junto con los demás aplicativos CASE que frecuenta en su labor diaria de desarrollo de software (herramientas de edición de texto, versionador de repositorios de código fuente, aplicaciones de chequeo y resolución de conflictos/diferencias entre archivos, entre otras); a los que se añaden ahora:

- PuTTy Link (plink): necesario para abrir túneles SSH entre ambas máquinas y ejecutar comandos en la máquina remota desde la máquina local.
- Rsync: herramienta de sincronización incremental eficiente para el intercambio de archivos y mantener así las copias de código fuente consistentes en la máquina virtual, según los cambios hechos por el desarrollador en su máquina local. La copia instalada aquí ejecuta en modo de aplicación cliente, para lanzar el proceso de sincronización de archivos bajo demanda del desarrollador de software.
- Cygwin: máquina virtual para la ejecución de Rsync (programa diseñado para ejecutar en entornos Unix) bajo la plataforma Windows del desarrollador.
- Finalmente, la herramienta CASE desarrollada en cuestión, para la coordinación de todas las herramientas antes descritas, y para presentar las nuevas acciones de las que a partir de ahora goza el desarrollador (sincronización de archivos entre ambos entornos, reinicio/ejecución de servicios remotos, visualización de logs remotos), en una interfaz integrada dentro de Eclipse, tal como cualquier otra herramienta que el mismo IDE ya provee de forma nativa.

Por el otro lado, es necesario migrar a la instancia remota de máquina virtual, los servicios y aplicativos necesarios para montar el servidor web en el cual se ejecutan remotamente (en la misma red que la base de datos, unidades de red y demás recursos lentos mencionados anteriormente) los cambios que el desarrollador realice en el código fuente de la aplicación, para su visualización inmediata en su máquina local, tal como siempre lo hizo usualmente. Puntualmente se necesita llevar a una imagen de máquina virtual que pueda instalarse fácilmente en cada instancia virtual que desee entregarse (una por cada desarrollador idealmente):

- Apache: el servidor web que actualmente sirve archivos estáticos de la aplicación, como ser imágenes, archivos de configuración varios y scripts, entre otros.
- Resin: el servidor de aplicación Java, que es el que sirve la aplicación web principal.
- La Máquina Virtual de Java, donde ejecuta Resin.
- Y por último es necesario instalar también en este entorno una copia del software Rsync (por lo general viene ya provista en la mayoría de las distribuciones de Linux, como es en este caso de aplicación), para que ejecute en modo daemon (demonio), a la escucha de los eventos/pedidos de sincronización que ocurran sobre el puerto en que la herramienta escucha las solicitudes.

Ahora es que particularmente se pone el foco de atención en el diseño de la herramienta CASE, ya que el resto de los componentes de software que fueron combinados para lograr el objetivo final ya existen previamente. De este modo, luego en la sección de implementación

se ahondará en más detalle acerca de cómo se interconectan estos aplicativos, mientras que ahora se irá algo más a bajo nivel en el diseño de la herramienta CASE desarrollada.

De aquí en adelante, y para clarificar todo el diseño hecho, y también para hacer de guía y facilitar las tareas subsiguientes de desarrollo que se realizaron para la herramienta CASE; es que se construyen y exponen los modelos desde alto a bajo nivel (desarrollo de modelos en modalidad top-down) de la herramienta CASE. Se hace uso en este caso del Lenguaje Unificado de Modelado (UML), dada su versatilidad e idoneidad en la materia, la claridad en su descripción y la facilidad que aportan luego los modelos desarrollados para construir la herramienta, partiendo desde el diseño modelado bajo este lenguaje. UML ofrece una basta cantidad de diagramas categorizados principalmente bajo tres tipos bien definidos de diagramas: diagramas de estructura, diagramas de comportamiento y diagramas de interacción. A su vez, cada una de estas categorías cuenta con sus propios tipos de diagramas para cubrir todas las necesidades de modelado de software a casi cualquier escala. Si bien no hacemos uso de todos los tipos de diagramas posibles, ya que a veces resulta en un despropósito y no se torna necesario ahondar hasta niveles de detalle tan específicos, sí se implementaron aquí los 3 tipos de categorías de diagramas, con los diagramas específicos que se consideran más idóneos en cada categoría. Esto es, a los fines prácticos de explicar lo mejor posible cómo se compone la herramienta CASE, y de modo que sirvan a su vez, para construir la herramienta en sí.

4.1 - Estructura

Para entender cómo se compone la estructura de la herramienta CASE desarrollada, es necesario entender primero qué papel juega dentro de toda la arquitectura de software y dónde se ubica respecto del resto de los componentes. Para esto se comienza con uno de los diagramas más básicos de UML, pero que permite ver al más alto nivel la arquitectura expuesta. A continuación el **diagrama de despliegue** de la arquitectura:

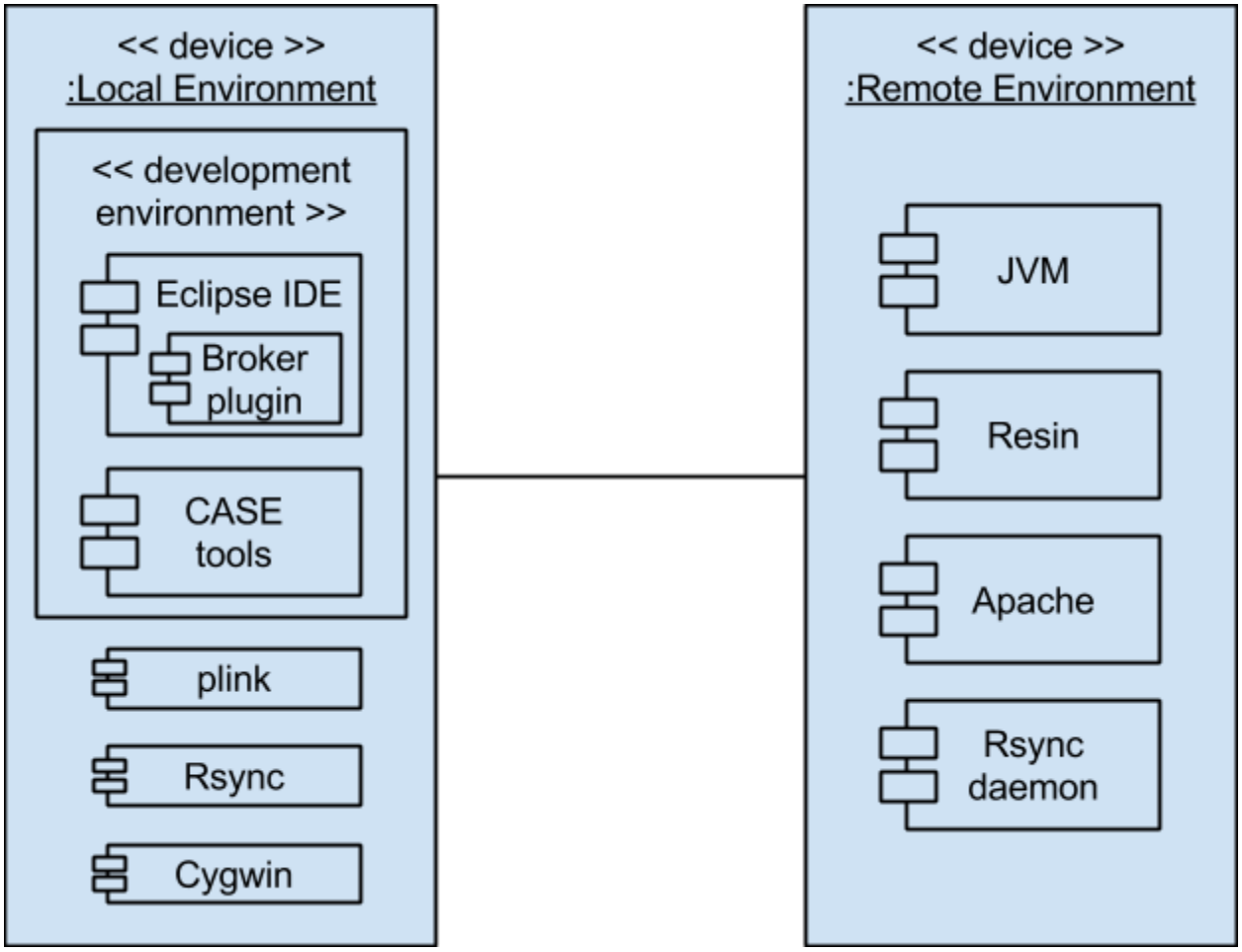

Figura 4.2: Diagrama de despliegue de la herramienta CASE.

Como se ve, es muy similar al diagrama anterior presentando el diseño general de la arquitectura, pero lo importante aquí, es poder especificar claramente la plataforma sobre la que se ejecuta el software; los dispositivos (tanto dispositivos físicos como lógicos) que intervienen en la arquitectura (modelar su disposición física a través de la arquitectura montada), qué componentes o artefactos involucra cada dispositivo (o dicho de otra manera, en qué dispositivo está desplegado cada componente) y con qué dispositivos y componentes interactúa directamente el usuario final, es decir, el desarrollador de software para el caso.
Es entonces que se plantea a alto nivel la arquitectura, como una plataforma formada por dos nodos principales (por cada desarrollador), su ambiente local (nodo físico, es la terminal de trabajo corriente del desarrollador, su propia computadora) y un nuevo ambiente remoto (formado por una instancia de máquina virtual instalada en la ubicación residente de los recursos más lentos del stack de tecnología para el desarrollo, para el caso de aplicación de este trabajo, la oficina de Buenos Aires). Si bien el desarrollador interactúa directamente con la arquitectura a través de su propia máquina, se encapsula en un nuevo nodo o dispositivo lógico, un subconjunto del software instalado en su máquina (su ambiente de desarrollo de

software), que son los aplicativos con los que realmente trabaja o interactúa diariamente (el IDE para el desarrollo de software, otras herramientas CASE, y lo más importante, la herramienta CASE broker desarrollada en este trabajo, para la coordinación y ejecución transparente del resto de los componentes de la arquitectura. Por ende el desarrollador no necesita conocer qué otros componentes están instalados y ejecutan en su máquina para la correcta ejecución de toda la plataforma (en este caso, Plink, Rsync y Cygwin), ya que el broker instalado lo abstrae de su interacción; mientras que tampoco necesita un conocimiento profundo de qué artefactos de software fueron desplazados hacia una máquina virtual remota, ni interactuar directamente con ellos, ya que lo realiza transparentemente a través de las facilidades suministradas por el plugin del broker, dentro del mismo CASE IDE que ya conoce.

Ahora sí se ahonda específicamente en el diseño del plugin de Eclipse (la herramienta CASE desarrollada) a modo de **broker** de esta arquitectura. Para ello se hace uso de la API brindada por el propio Eclipse para el desarrollo de la herramienta CASE mencionada; utilizando de aquellas interfaces de expansión que provee Eclipse, los componentes que mejor cumplen los requisitos necesarios para implementar los casos de uso (estos se verán más adelante en la sección de diseño del comportamiento de la herramienta). Este es entonces el **diagrama de componentes** del plugin:

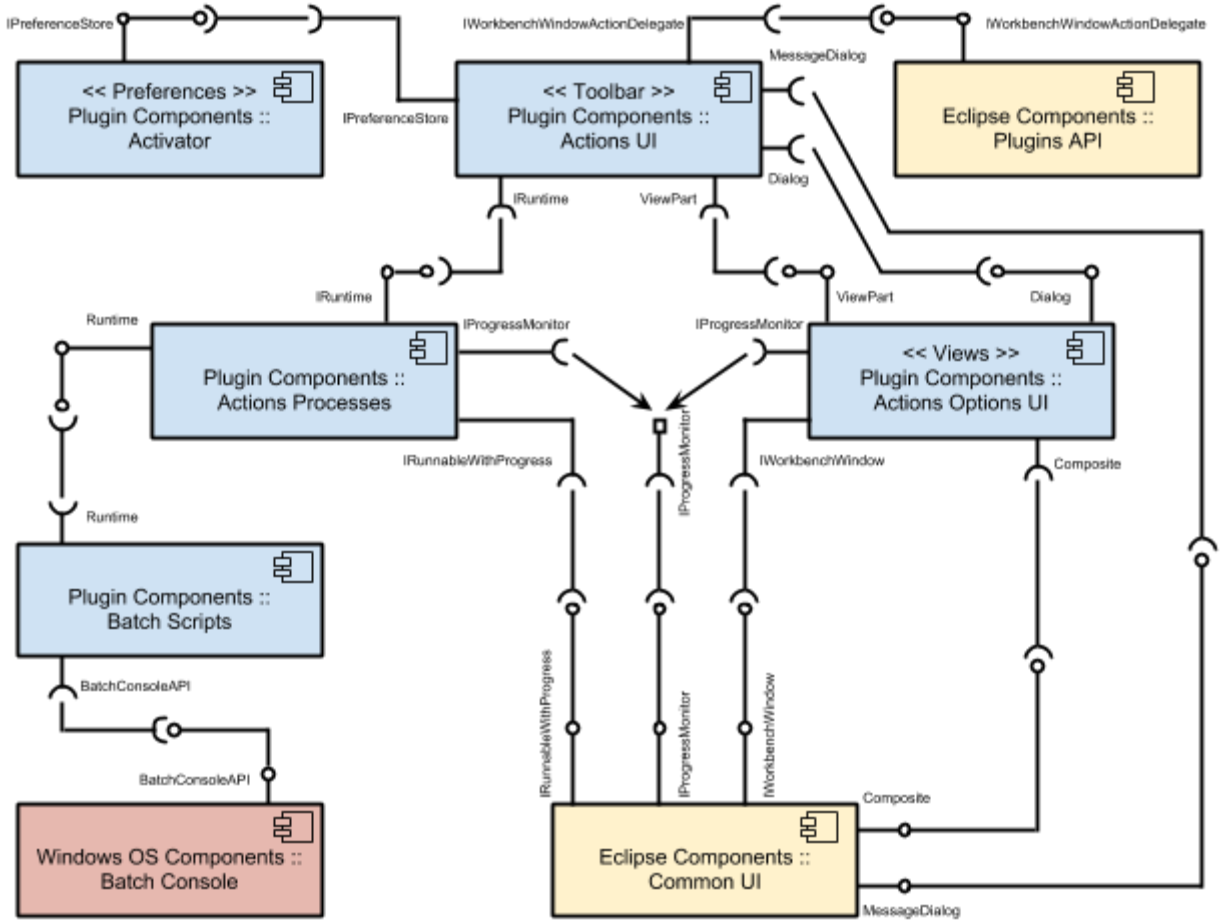

Figura 4.3: Diagrama de componentes del plug-in broker.

La separación por colores se hace para facilitar la identificación de qué componentes forman parte del sistema operativo (rojo, Windows OS Components), qué componentes forman parte nativa del IDE (amarillos, Eclipse Components) y qué componentes se desarrollaron en el plugin broker para incorporar al IDE en la herramienta CASE mejorada (azules, Plugin Components).

Como se ve, los componentes del plugin están segmentados en cinco tipos de piezas de software, cuatro de las cuales siguen los principios de desarrollo de extensiones de Eclipse IDE para su correcta integración (como se describieron en la sección 3.4.2 de “Arquitectura expandible de plug-ins”, de “Entorno de Desarrollo Integrado (IDE)”). Estos se agrupan en:

- **Activator**: es el almacén de preferencias y configuraciones que Eclipse brinda a un plugin. Aquí están contenidos los métodos para leer y guardar las opciones brindadas al desarrollador desde el plugin; como así también está el módulo con la integración de dichas preferencias en el panel de opciones generales de Eclipse a través de la API provista por el mismo, para que el desarrollador pueda encontrar las opciones de configuración, en el mismo lugar que cualquier otra herramienta nativa del IDE las

ofrece usualmente.

- **Actions UI**: es el punto de entrada principal de la interfaz de usuario brindada por la herramienta CASE al desarrollador. Para facilitar la integración al IDE y agilizar la localización de las nuevas funcionalidades brindadas por la herramienta; se integra directamente como nuevos botones en la barra de herramientas principal (toolbar) de Eclipse, a través de la interfaz provista por Eclipse para añadir acciones al toolbar.
- **Actions Options UI**: para aquellas funcionalidades que requieren ingreso de datos por parte del desarrollador (como por ejemplo, el nombre de un job a ejecutar), y/o requieren salida de datos, resultados o información (archivos de logs, resultado del proceso de sincronización, entre otras); se construyeron las vistas de interfaz de usuario necesarias, a partir de las interfaces de vistas, diálogos, controles de progreso, y demás de la API de Eclipse.
- **Actions Processes**: una vez recolectadas las preferencias y los datos de entrada necesarios para ejecutar la funcionalidad seleccionada por el usuario, se crearán y dispararán los procesos adecuados para realizar la tarea requerida. Nuevamente se desarrollaron los procesos bajo este módulo, siguiendo el diseño expansible de Eclipse, y valiéndose de las interfaces de monitoreo de procesos, y de informe de progreso de los mismos a la interfaz de usuario, según cada caso lo requiere.
- **Batch Scripts**: es el único componente del broker que no forma parte del framework de Eclipse (no es parte propia del plugin de Eclipse en sí), sino que son scripts a ejecutar directamente en el sistema operativo local para comunicarse con la máquina virtual remota, a través de las nuevas piezas de software instaladas en ambos extremos de la arquitectura descrita. Son invocados a través de las interfaces de tiempo de ejecución del sistema operativo implementadas en los procesos que se desarrollaron.

El componente **Batch Console** es parte de la API misma del sistema operativo local (Windows para el caso desarrollado), exponiendo las directivas (programas de línea de comandos) que son usadas en los scripts para la comunicación entre ambos dispositivos local y remoto. Por último, **Plugins API** contiene las librerías propias de Eclipse que son utilizadas para construir el plugin y **Common UI** contiene las interfaces expuestas por Eclipse para poder integrar las funcionalidades en la misma interfaz de usuario, dando la impresión de que son herramientas nativas del propio IDE.

4.1.1 - Patrón arquitectural

Debido a la complejidad del aplicativo desarrollado, y a la necesidad de tener múltiples componentes para servir a varios propósitos de desarrollo distintos, que deben a su vez entablar comunicaciones con servicios ejecutando en un entorno remoto; se decide hacer uso del patrón arquitectural **Broker**, de modo de tener un mecanismo específico y aislado del resto de los componentes para comunicarse con el sistema remoto de máquina virtual. Así se deja toda la responsabilidad e implementación de la comunicación específica subyacente al

componente broker, y se abstrae al resto de los componentes de detalles de implementación de mensajería; mientras que se facilita su manutención, al tener toda la lógica de comunicación implementada en sólo un componente específico y no distribuida entre todos los artefactos de software. Esto no sólo facilita el mantenimiento del software, si la tecnología subyacente cambia (se decide cambiar el lenguaje de scripting o implementar el broker para otro sistema operativo), entonces los detalles de implementación de mensajería sólo tendrán que ser modificados en el componente broker y no en el resto de los componentes. También si a futuro se decide incorporar nuevos componentes que resuelvan otras necesidades de desarrollo, pueden estructurarse de la misma forma que los componentes desarrollados y no tener que lidiar con aspectos de implementación de mensajería ya resueltos en el broker, es decir, el software se desarrolla a partir de una arquitectura escalable.

El broker desarrollado posee un archivo de configuración donde se listan las ubicaciones locales de los scripts que sirven a la comunicación con los servicios remotos (estos scripts pueden considerarse aquí parte del broker en sí), de modo que si fuera necesario modificar las instrucciones de scripting, o bien por ejemplo cambiar por completo el lenguaje de los scripts, sólo debería actualizarse la configuración leída por el broker de modo totalmente transparente a los componentes de software que hacen uso de los servicios provistos por el broker para la comunicación remota (solicitan al broker la ejecución de estos scripts, pero sin conocer su ubicación ni sus detalles de implementación).
A su vez, el broker también posee como parte de su configuración, la dirección IP de la máquina virtual remota de desarrollo asignada al desarrollador de software y las credenciales para acceder de forma segura a la misma; así los componentes individuales no requieren conocer en absoluto estos detalles sino simplemente saber solicitar al broker que ejecute la acción determinada en la máquina virtual, sin especificar explícitamente cómo se resuelve la comunicación contra la instancia remota, ni de qué instancia remota específica se trata, estas cuestiones las resuelve interna y transparentemente el broker.

Si fuera necesario editar las rutas de los scripts con las instrucciones para realizar la comunicación y acciones contra la máquina virtual remota, o el archivo de clave privada para la comunicación segura con la misma; puede hacerse fácilmente desde el archivo de configuración de tipo “Properties” de texto plano (archivos usados usualmente en tecnologías Java para almacenar los parámetros configurables de una aplicación) del cual se nutre el plugin del broker al iniciar el IDE Eclipse (Activator.properties) para inicializar su configuración interna, por medio de los servicios desarrollados en el Activator.
Por otro lado la configuración pertinente a especificar la dirección IP de la instancia virtual asignada y el usuario con que se identificará el broker en la misma, pueden editarse desde el panel de preferencias del mismo CASE IDE, en una nueva sección desarrollada específicamente para las necesidades de este plugin. Así la configuración inicial puede realizarse de modo sencillo una vez que el desarrollador es informado acerca de cuál máquina virtual se creó para él, e incluso puede cambiarse entre máquinas virtuales distintas ante propósitos específicos de testeo, sin más inconvenientes que abrir las preferencias del IDE, y editar la opción mencionada.

Según lo especificado, se realiza un boceto conceptual de la organización de los componentes del aplicativo en forma de una arquitectura de broker:

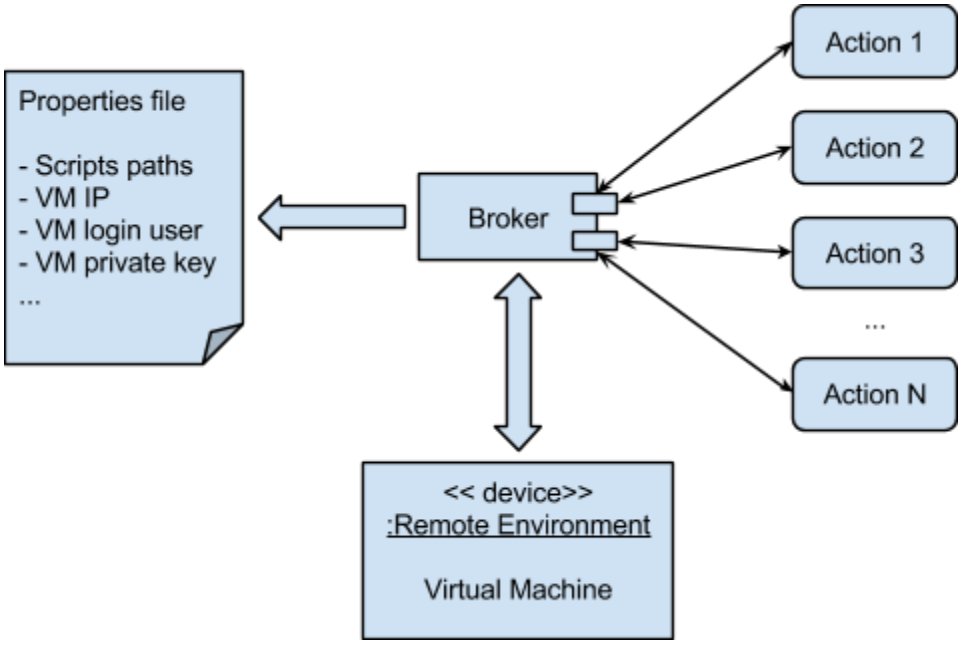

Figura 4.4: Arquitectura del broker desarrollado.

Como se ve, cuando el desarrollador de software ejecuta las acciones provistas en el plugin, estas interactúan con el broker para solicitarle la ejecución de scripts en la máquina virtual remota, configuraciones que sólo conoce y administra el componente broker.

4.1.2 - Patrones de diseño

Continuando con las conveniencias del desarrollo de software que aplica las buenas prácticas de la ingeniería de software, es que se desarrollaron algunos de los componentes del plugin de la herramienta CASE, siguiendo algunos de los patrones de diseño idóneos para cada caso. Esto permite modelar el software usando componentes estructurados de modo que ya ha sido probado que las respectivas soluciones sirven al propósito, y facilitan su comprensión y mantenimiento. Se detallan aquí los patrones de diseño aplicados, en qué componentes fueron aplicados y cómo/por qué:

- **Proxy**: se puede considerar a todo el plugin desarrollado como un proxy a la arquitectura de máquina virtual remota, abstrayendo y facilitando el acceso a los recursos remotos, mientras que seguriza el acceso a los recursos críticos de la arquitectura. Puntualmente el componente de Actions (acciones del plugin) es el que

por medio de una interfaz de usuario amigable, oculta al desarrollador toda la implementación subyacente que ocurre luego de presionar el botón adecuado de la barra de herramientas del IDE, para ejecutar la acción deseada. Cada una de estas acciones, ofrece un proxy a distintos recursos o aplicativos remotos, como ser el servidor web o servidor de aplicación remoto, archivos de logs remotos, código fuente remoto e incluso la base de datos de desarrollo remota a través del aplicativo desplegado. Dado que algunos de estos recursos son críticos o bien no deberían ser accedidos directamente por el desarrollador, es que el proxy adecuado media en la interacción, ofreciendo una representación útil y más simple del componente en cuestión, pero segura de ser accedida por el desarrollador (representaciones locales de logs remotos por ejemplo).

- **Forwarder/Receiver**: para llevar a cabo las tareas que ofrece el plugin del broker, se requiere de comunicación transparente interprocesos. Así es que se modelaron como “forwarders” a los Processes (procesos en el modelo de plugins de Eclipse) del plugin desarrollado. Estos son los responsables de solicitar al almacén de preferencias (Activator) la dirección física de los archivos de scripting locales y de la máquina virtual remota asignada al desarrollador. Una vez que los forwarders poseen las direcciones en cuestión, proceden a recolectar los parámetros necesarios para la ejecución de la acción seleccionada y realizar marshalling con estos datos, a fin de construir un mensaje que es enviado a través de la arquitectura desplegada. Una vez que la tarea es ejecutada remotamente, los “receivers” (en este caso son las Views de la arquitectura de plugins de Eclipse) capturan los mensajes de retorno y realizan el proceso opuesto de unmarshalling, para luego mostrar el resultado en la interfaz de usuario desarrollada en cada view.
- **Client-Dispatcher-Server**: como se mencionó recién, los Processes del plugin no requieren conocer de la ubicación física de los servicios/equipos remotos, para ello está el despachador en medio de la arquitectura cliente/servidor. Mediante su configuración, este componente (Activator en la arquitectura de Eclipse) registra las ubicaciones físicas de los servicios, y las ofrece de modo transparente como nombres de servicios o componentes. De esta forma, los Processes pueden solicitar el acceso a la máquina virtual remota, a archivos de scripting locales o a archivos de log remotos, sin necesidad de conocer los detalles subyacentes de ubicación o comunicación. Nótese por ejemplo, que en este caso de implementación, los recursos locales se encuentran ubicados bajo una arquitectura Windows, mientras que los recursos remotos están desplegados en una arquitectura Linux. Aún así los Processes son agnósticos a estos detalles y simplemente acceden a los recursos en cuestión de modo transparente, a partir del enlace provisto por el Dispatcher. Para facilitar y asegurar la flexibilidad en la configuración del Dispatcher, es que se modeló una interfaz de usuario para la configuración de la dirección IP y de las credenciales de acceso a la máquina virtual remota, dentro de las mismas preferencias de usuario de Eclipse.
- **Publisher-Subscriber**: dado que en la arquitectura de desarrollo de software remoto implementada se torna necesario mantener consistentes las copias del software

desplegado en sí (en particular, el software ejecutando en el servidor web remoto), es que se modelan un componente productor (publisher) en la instancia local de desarrollo, y uno suscriptor (subscriber) en la instancia remota. Así, estos componentes pueden mantener actualizado el software editado localmente y ejecutado remotamente, a través del proceso de sincronización incremental provisto por el software Rsync. Para este propósito es que se instala Rsync en el equipo local para ser invocado bajo demanda a través del proceso adecuado del plugin, al momento de requerir de la sincronización del software desarrollado. Este módulo promotor, mantiene en un archivo de configuración local la lista de componentes (recursos o archivos locales) cuyo estado debe ser comunicado al suscriptor desplegado en la instancia remota (otra copia de Rsync). Este último se mantiene ejecutando en la máquina virtual a modo de daemon (demonio), listo para escuchar las notificaciones enviadas por el productor para ser impactadas en la instancia remota, y luego ejecutar acciones adicionales, como por ejemplo, el reinicio del servidor web para que ponga a disposición del desarrollador el nuevo código implementado.

4.2 - Comportamiento

Expuesto ya el diseño general de la arquitectura y la estructura de cómo se desarrolló el broker de nuestra herramienta CASE, es importante explicitar ahora qué facultades son brindadas al usuario (desarrollador de software) con esta herramienta, cuáles son las acciones que ahora tiene disponibles y cómo es la interacción entre desarrollador y la herramienta CASE en cada una de estas acciones. Planteamos entonces el siguiente **diagrama de casos de uso**:

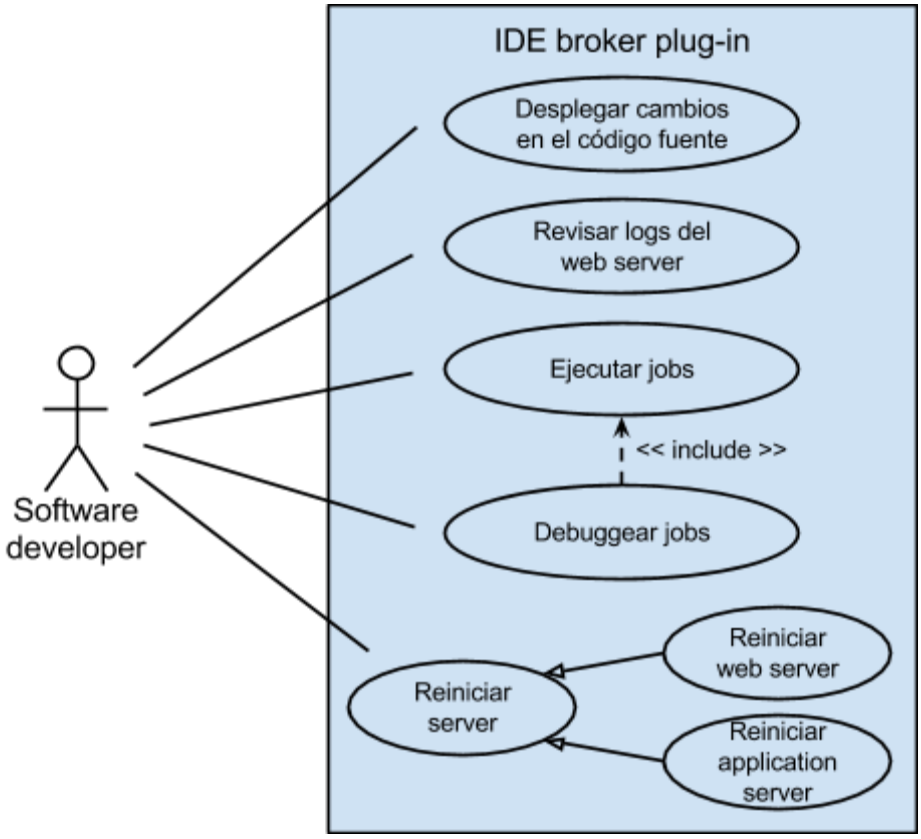

Figura 4.5: Diagrama de casos de uso para la herramienta CASE.

Como se ve, los casos de uso en detalle constan de:

- **Desplegar cambios en el código fuente**: acción fundamental durante el ciclo de desarrollo de software planteado. Luego de editar el código fuente de la aplicación en el entorno local de desarrollo, es necesario proveer el mecanismo para que el usuario pueda propagar estos cambios en el servidor web del entorno remoto, de modo que los cambios efectuados tomen efecto en su ambiente de pruebas.
- **Revisar logs del web-server**: el logging (traceo de mensajes que ayudan a depurar una aplicación de software) es una herramienta útil y frecuente durante el ciclo de desarrollo de software. Dado que con la nueva arquitectura implementada, el servidor web que ejecuta la aplicación ya no está montado en la computadora del desarrollador, hay que proveer una facilidad para acceder y visualizar el log de la aplicación ejecutando en el entorno remoto; a fin de que el desarrollador no pierda la pista de qué sucede exactamente durante la ejecución y pruebas de los módulos que está editando.
- **Ejecutar jobs**: como se mencionó anteriormente, los jobs programados son otra herramienta de desarrollo frecuente en esta aplicación. Tenemos entonces que poder lanzar cualquier job nuevo o existente en la aplicación montada en el entorno remoto de desarrollo.
- **Debuggear jobs**: asimismo, es deseable (o más bien, esperable) que también

puedan depurarse en el mismo CASE IDE, los jobs lanzados remotamente a través del broker desarrollado, y que el desarrollador pueda especificar a través de la herramienta, cuándo desea que un job simplemente sea lanzado (ejecutado) para que efectúe un proceso en particular, y cuándo desea depurarlo para analizar en detalle el comportamiento del mismo, o para proceder a encontrar errores aplicativos en él (bug fixing).

- **Reiniciar web-server**: provee un acceso fácil para reiniciar el servidor web Apache remoto, sin necesidad de acceder o interactuar directamente con la máquina virtual remota.
- **Reiniciar application-server**: del mismo modo que en el caso de uso anterior, provee un acceso también para el reinicio del servidor aplicativo Resin, para que por ejemplo tome nuevos cambios, levante nuevas librerías, libere cachés, etc.

Es importante modelar también cuál es la interacción que sucede entre la herramienta CASE y el desarrollador en cada uno de estos casos de uso. Para ello hacemos uso de los **diagramas de actividades** de UML. Se comienza entonces con el diagrama de actividades del primer caso de uso (“Desplegar cambios en el código fuente”):

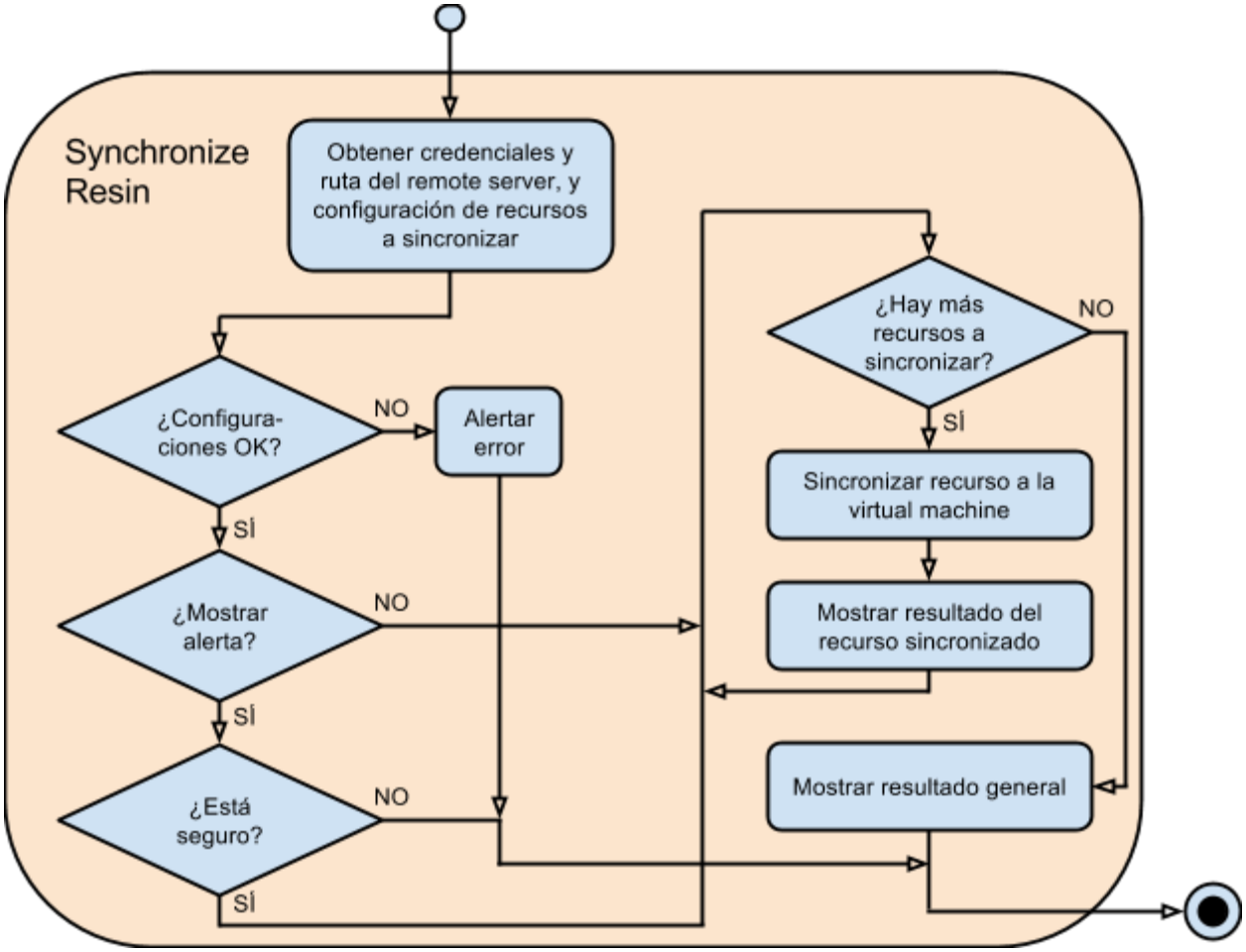

Figura 4.6: Diagrama de actividades del caso de uso “Desplegar cambios en el código fuente”.

Cuando el desarrollador decida disparar la acción pertinente a sincronizar los cambios hechos en el código fuente de su máquina local con la remota, y desplegar dichos cambios en la máquina virtual, lo primero que debe hacer el broker es localizar en su propia configuración interna, cuál es la dirección IP propia de la máquina virtual particular de dicho desarrollador y las credenciales de acceso a la misma (clave pública/privada), junto con la lectura de otro archivo de configuración que contiene cuáles son las entradas en el sistema de archivos (local) que debe sincronizar con el sistema remoto. Cualquiera de estas configuraciones que se encuentre errónea o faltante, debe mostrar un mensaje de alerta descriptivo del error y finalizar la ejecución. Si está todo en orden, entonces debe mostrar una alerta con ingreso de usuario para prevenir al desarrollador del cambio que está a punto de efectuarse y solicitar su aceptación. Esto es útil por si el desarrollador presiona involuntariamente el botón que dispara esta acción en cuestión, pero también es conveniente poder omitir esa alerta las próximas veces, una vez que el desarrollador ya se encuentra acostumbrado a la operatoria; por ende el broker debe revisar también una configuración propia del desarrollador que especifique si debe mostrarse la alerta en cuestión o si debe actuar directamente. Una vez que se decide comenzar la ejecución del proceso de sincronización y despliegue (deploy), debe utilizarse transparentemente la herramienta de software Rsync para localizar y decidir cuáles archivos requieren de sincronización, y por cada archivo sincronizado, mostrar el resultado del mismo en la ventana de salida pertinente del IDE CASE. Una vez que se hayan agotado los recursos a sincronizar, el plugin del broker debe mostrar al usuario un resumen acerca del éxito o fracaso de la ejecución, según sea el caso.

Se continúa con el diagrama de actividades del caso de uso “Revisar logs del web-server”:

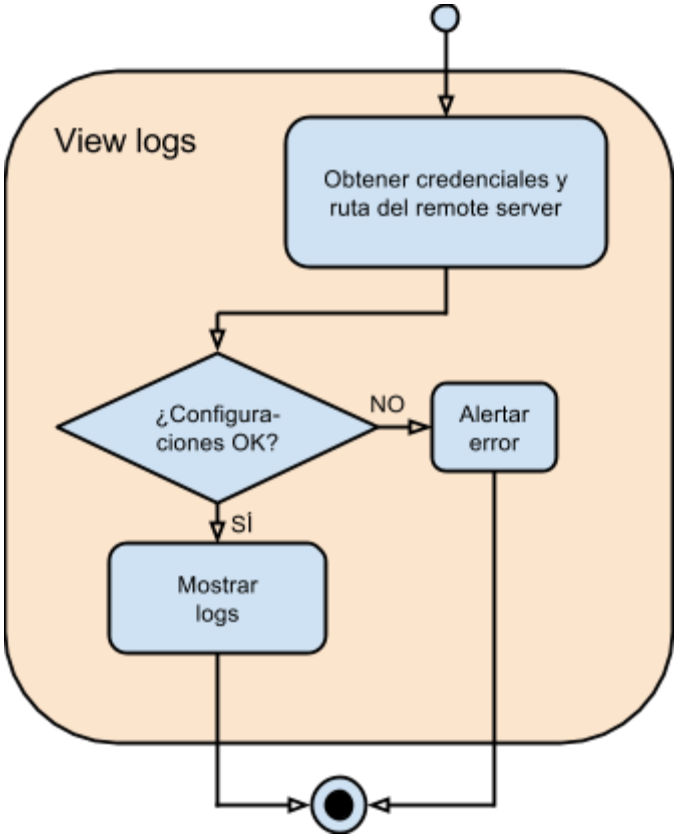

Figura 4.7: Diagrama de actividades del caso de uso “Revisar logs del web-server”.

Este caso de uso es mucho más simple que el anterior, ya que ante la acción del usuario de presionar el botón para visualizar el archivo de log del servidor web ejecutando en la máquina remota virtual, el broker simplemente tiene que obtener las configuraciones que posee pertinentes al acceso del equipo en cuestión, y si está todo en orden, ejecutar el comando que a través de un túnel SSH provisto por el software Plink, propague la salida de texto estándar del proceso del servidor de aplicaciones Resin remoto, en una ventana local del sistema operativo. El único caso excepcional es que, similar al anterior, la configuración no hubiera sido hecha aún o sea inválida, en ese caso debe mostrar un error y finalizar.

Como la interacción del desarrollador de software con los casos de uso “Ejecutar jobs” y “Debuggear jobs” son similares, se muestran ambos juntos en el siguiente diagrama de actividades, para luego explicar la salvedad entre ellos:

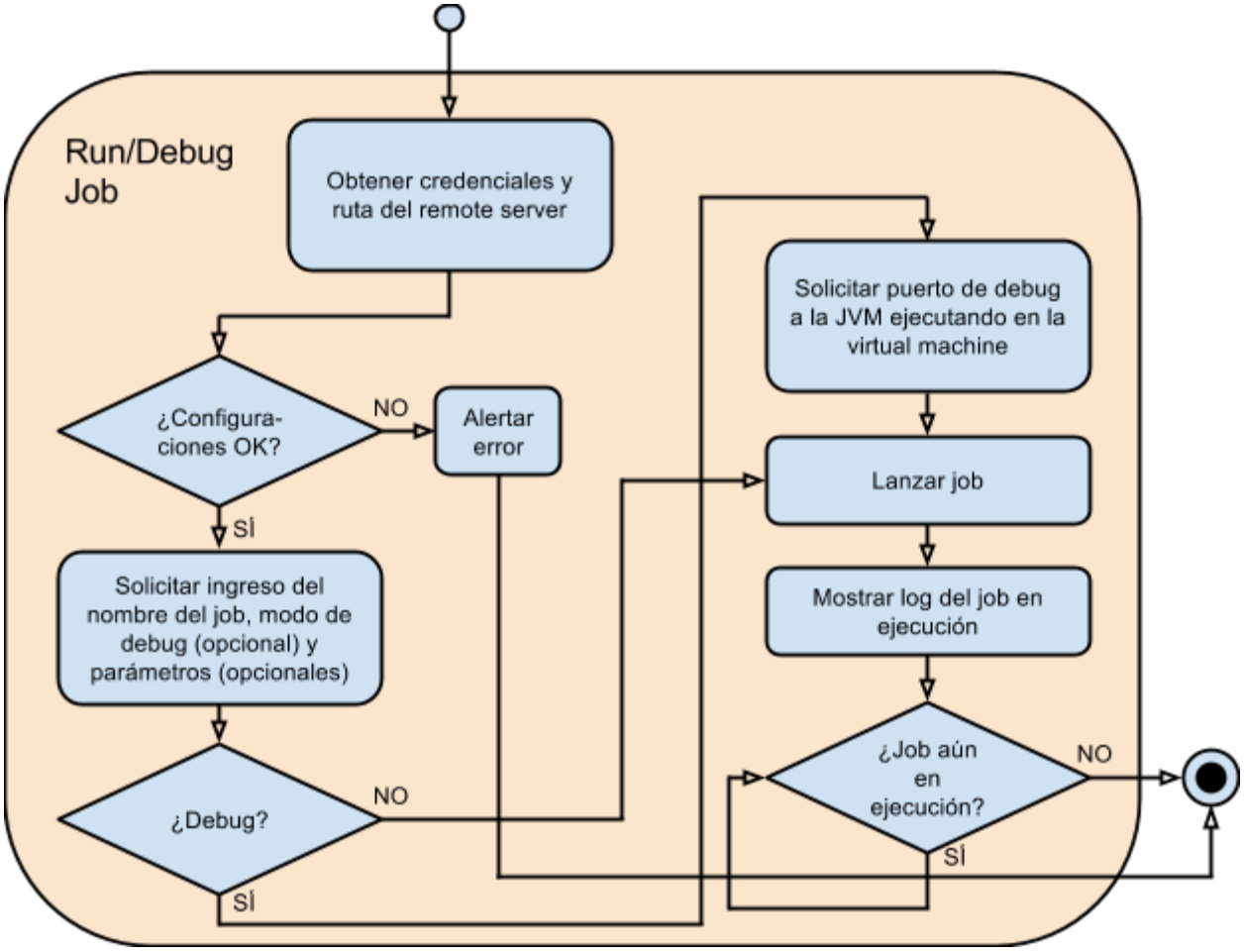

Figura 4.8: Diagrama de actividades de los casos de uso “Ejecutar jobs” y “Debuggear jobs”.

Como se ve, se comienza como en los anteriores, encargándose el plugin del broker de obtener las configuraciones de acceso al equipo remoto, controlar su integridad y mostrar un error en caso contrario. No es necesario ahora un diálogo de confirmación con el desarrollador, ya que el siguiente ingreso de datos obraría como tal (si la acción fue involuntaria, el desarrollador puede cancelarlo, dando por finalizado el proceso). Entonces llegado este punto, el plugin debe mostrar una ventana de tipo formulario, pidiendo al desarrollador que ingrese el nombre del job a ejecutar en la máquina virtual remota (el nombre de la clase Java del job), un input para especificar opcionalmente si desea debuggear el job luego de lanzado (aquí la distinción en la interacción entre los dos casos de uso), y finalmente permitir ingresar opcionalmente parámetros de texto a suministrar al job a ejecutar. Ahora viene la distinción funcional entre ambos casos de uso, ya que si el desarrollador optó por debuggear en lugar de sólo ejecutar el job, el broker debe preparar los parámetros necesarios en el comando que correrá en un nuevo túnel SSH solicitado a PLink, para especificar al puerto de la máquina virtual de Java (JVM) remota, que detenga la ejecución del job hasta que el debugger de Eclipse (software depurador del IDE) se conecte a él. Así se puede debuggear el job en la misma interfaz del CASE IDE usual. Una vez que el

comando está preparado, el broker debe ejecutarlo en la máquina virtual remota. Adicionalmente es útil al desarrollador, que el broker inicie otro comando automáticamente para mostrar localmente la salida de texto de consola del job ejecutando en la máquina virtual. Una vez que la ejecución del job o la sesión de debug finaliza, entonces el broker debe cerrar ambos túneles SSH finalizando el caso de uso.

Por último, se muestran los diagramas de actividades de los casos de uso restantes, “Reiniciar web-server” y “Reiniciar application-server”, unificados dado su similitud (ambos heredan del mismo caso de uso “Reiniciar server”):

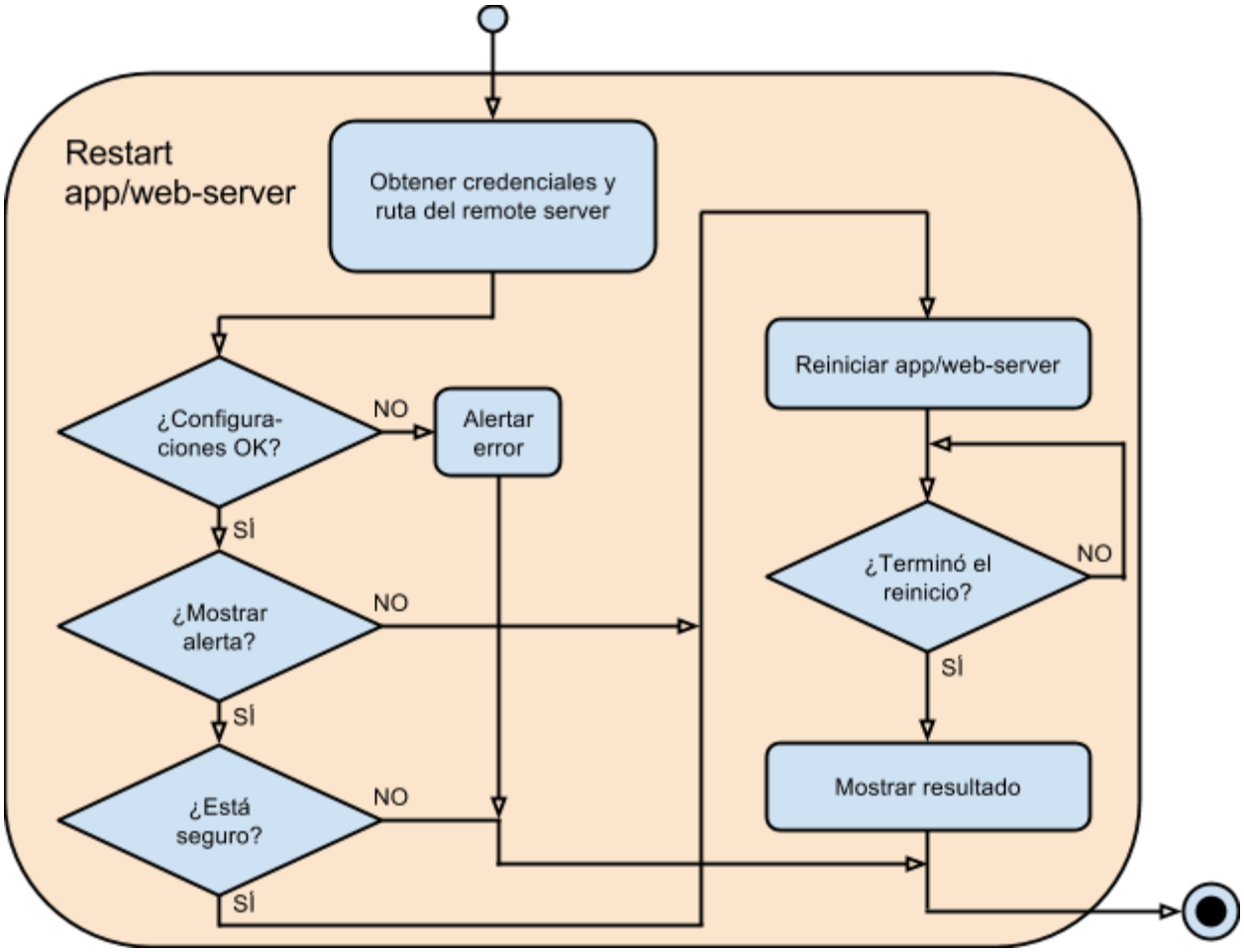

Figura 4.9: Diagrama de actividades de los casos de uso “Reiniciar web-server” y “Reiniciar application-server”.

Esta interacción es simple también, ya que es el comportamiento necesario para reiniciar un servidor remoto ante la solicitud del desarrollador. Cuando éste presione el botón correspondiente en la barra de herramientas provista por el plugin del broker, y luego de nuevamente, localizar el equipo remoto y corroborar la correctitud de las configuraciones, se pide confirmación al usuario (con opción de omitirla en futuras interacciones), para luego

proceder a que el broker ejecute el comando vía túnel SSH provisto por Plink para realizar el reinicio remoto, bien del servidor web Apache, como del servidor de aplicación Resin, según sea la elección del usuario con el correspondiente botón. Una vez finalizado el reinicio, el plugin debe mostrar un cuadro de diálogo informando al usuario del resultado de la operación.

4.3 - Interacción

Una vez que se tiene el diseño de la herramienta CASE, con sus componentes internos, cuáles se despliegan en cada extremo de la arquitectura de software, los casos de uso que cumple la herramienta de software, y cómo es el comportamiento de cada caso de uso para con el usuario; resta especificar cuál es la interacción interna entre los componentes de software en cada caso de uso. De este modo, la traducción desde los diseños proporcionados hacia la construcción del software en sí, es un tarea mucho más sencilla, además de que sirven a los propósitos de un mejor entendimiento de la herramienta y documentación de la misma también. Para esto, se hace uso ahora de los **diagramas de secuencia** de UML a fin de especificar cuál es la interacción de los componentes en los casos de uso. Se comienza entonces nuevamente por la secuencia que refiere al caso de uso “Desplegar cambios en el código fuente”:

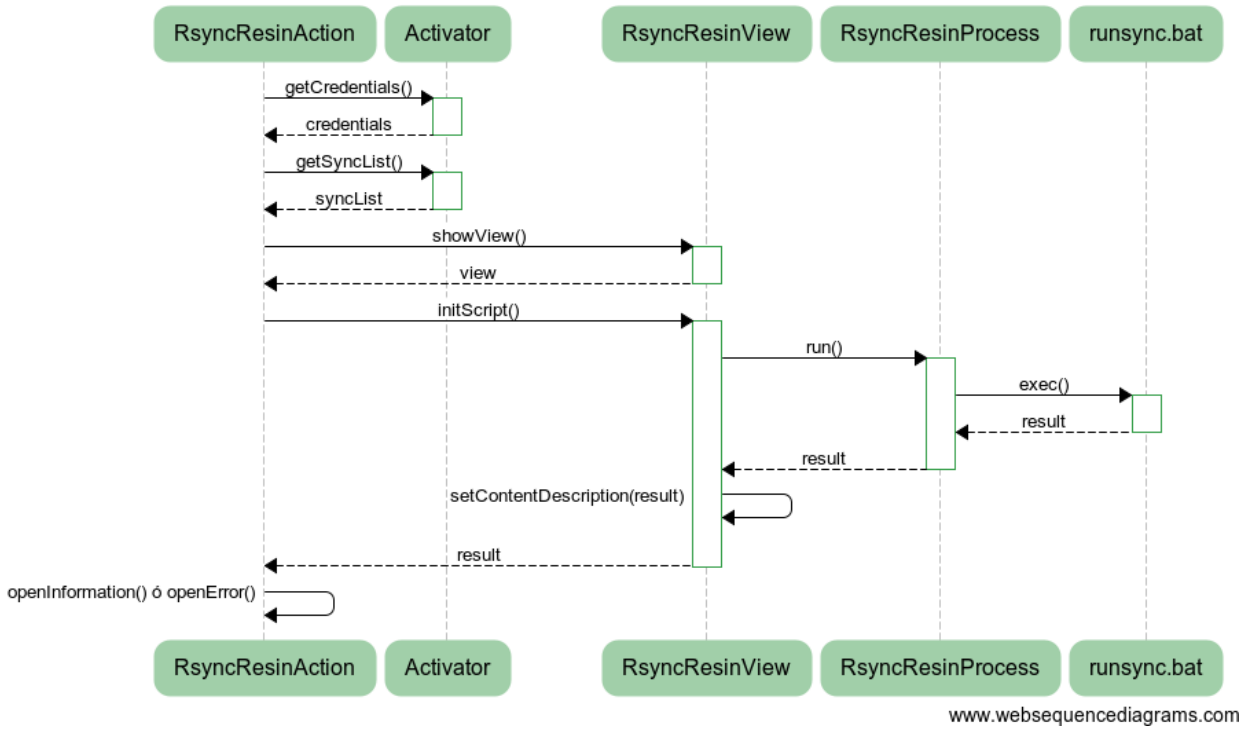

Figura 4.10: Diagrama de secuencia del caso de uso “Desplegar cambios en el código

fuente”.

La secuencia empieza cuando el desarrollador de software dá click en el botón correspondiente que se muestra en la barra de herramientas del IDE CASE Eclipse para la acción de sincronizar y desplegar los cambios en el código fuente editado. Por cómo se ha estructurado el diseño de los componentes del plugin del broker, según el framework de extensión de plugins de Eclipse, se dispara entonces el action correspondiente para atender el pedido. Este action (RsyncResinAction) consulta al componente Activator (esta es la nomenclatura dentro de la organización de componentes de Eclipse, pero refiere al almacén de preferencias, o dicho de otra forma, el componente que implementa el **patrón de diseño dispatcher**) por las credenciales necesarias para ejecutar comandos contra la máquina virtual, específicamente la IP de la misma, y la private key para autenticarse a ella. Luego pide también, cuáles son los archivos configurados para chequear el estado de sincronización contra la máquina virtual (para luego controlar si están sincronizados, o fueron editados posteriormente a la última sincronización). Una vez que cuenta con todas las configuraciones necesarias solicita al componente de vista de usuario, RsyncResinView, que muestre el panel correspondiente donde se listará el estado de cada archivo verificado para sincronizar, y el resultado de su sincronización si correspondiera. Una vez que el panel fue activado, el action debe crear el proceso RsyncResinProcess que será el encargado de ejecutar el script de línea de comandos necesario para la operación. De este modo entonces, el proceso ejecuta uno de los archivos de scripting provistos dentro del plugin, por medio de la interfaz de comandos del sistema operativo (para este caso de implementación, Windows), y propaga de retorno la salida del script hasta la view, para mostrar el listado de archivos al desarrollador. Finalmente, cuando la lista de archivos fue procesada e informada, se propaga el resultado general (éxito o error) al action original, para que abra un cuadro de diálogo informando al desarrollador del resultado del proceso.

Se continúa con el segundo de los casos de uso desarrollados, "Revisar logs del web-server":

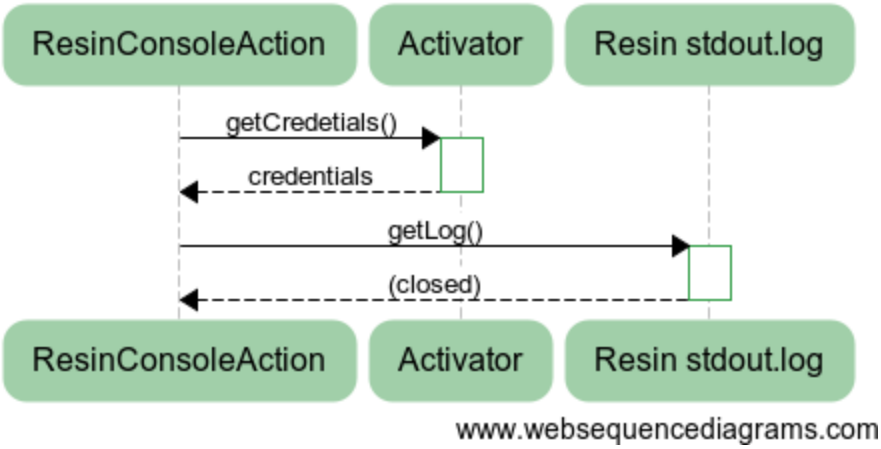

Figura 4.11: Diagrama de secuencia del caso de uso “Revisar logs del web-server”.

Esta secuencia es mucho más simple que la anterior y comienza de modo similar, cuando el desarrollador presiona el botón correspondiente de la barra de herramientas, el action adecuado (ResinConsoleAction) pide nuevamente las credenciales al Activator, e inmediatamente abre un túnel SSH (por medio de PLink) para propagar el archivo de salida estándar del log del webserver Resin, en la máquina local. Cuando el desarrollador cierra el stream (flujo de datos) de log, la secuencia finaliza automáticamente.

Nuevamente se muestran unificadas las secuencias de los casos de uso "Ejecutar jobs" y "Debuggear jobs" dada su similitud:

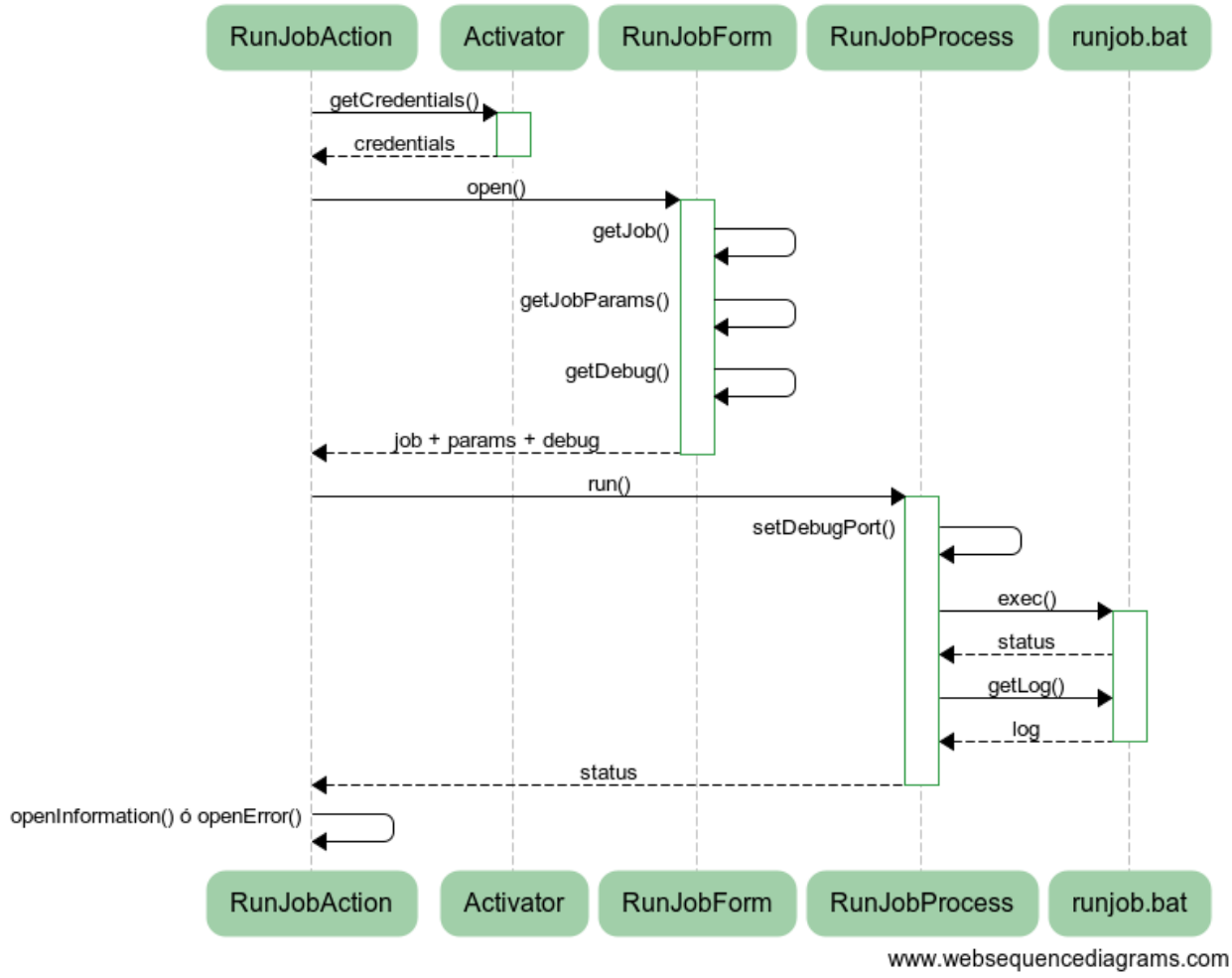

Figura 4.12: Diagrama de secuencia de los casos de uso “Ejecutar jobs” y “Debuggear jobs”.

Cuando el desarrollador presiona el botón, se dispara la acción RunJobAction, que pide las credenciales como el resto de las demás acciones al Activator; para luego pedir a otro de los

componentes de view (vistas de usuario, específicamente RunJobForm en este caso) que abra un formulario de ingreso de datos para solicitar algunos inputs al desarrollador (nombre del job a ejecutar en la máquina virtual, una lista de parámetros opcional a suministrar al job, y si desea correrse el mismo en modalidad de debugging). Una vez que todas las configuraciones fueron recolectadas, el action inicializa el proceso adecuado, RunJobProcess, con dichos parámetros. Este último es el encargado de configurar el puerto de debug en la JVM remota (si así fue solicitado), y ejecutar la secuencia de comandos del script correspondiente provisto en el plugin. Además abre un segundo túnel SSH para visualizar localmente, la salida del job ejecutando en la instancia remota. Una vez que el job finaliza, el proceso retorna el estado de finalización al action, que se encarga de abrir un cuadro de diálogo informando al desarrollador si la ejecución fué exitosa, u ocurrió algún error.

Por último, se muestra ahora la secuencia correspondiente al reinicio de un servidor remoto. Como la operatoria necesaria para reiniciar el servidor web y el servidor de aplicación son idénticas (y ambas extienden del mismo caso de uso), se unifican en el siguiente diagrama de secuencia, los casos de uso "Reiniciar web-server" y "Reiniciar application-server":

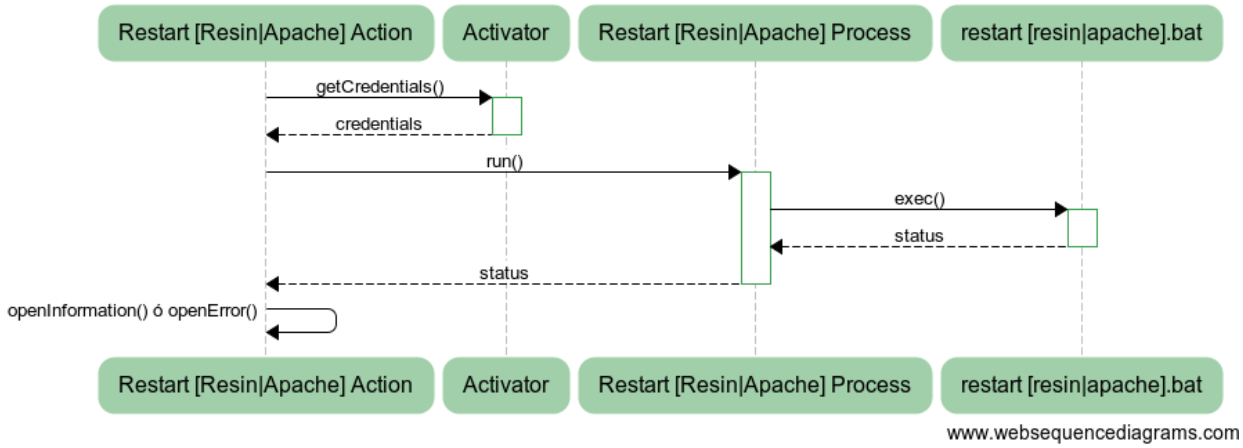

Figura 4.13: Diagrama de secuencia de los casos de uso “Reiniciar web-server” y “Reiniciar application-server”.

Luego de presionado el correspondiente botón de la barra de herramientas del plug-in, se dispara uno de los dos actions adecuados, el cual una vez más, solicita las credenciales al Activator e inicia el proceso adecuado en el framework de Eclipse. Este proceso a su vez, ejecuta el archivo de secuencia de comandos necesario para el reinicio del servicio solicitado. Una vez que el servidor reinició, esto es informado al proceso, y de retorno a su vez a la acción, que muestra un cuadro de diálogo informando del resultado al desarrollador.

5 - Arquitectura desplegada

En este capítulo se exponen los pasos más relevantes, realizados durante la implementación de la arquitectura de software lograda. Se detalla el análisis de las opciones de herramientas a utilizar en cada aspecto de la misma, sus prestaciones, por qué se seleccionaron y qué le aportan a la arquitectura desplegada y al objetivo de la herramienta en sí. Se define la arquitectura de la herramienta, de qué módulos de software y hardware consta, y cómo se construyó el software CASE, y su sucesiva puesta en producción.

Para definir la arquitectura planteada, a partir del problema original de lentitud en el acceso a los recursos y herramientas de desarrollo, bajo el panorama descripto en el capítulo 1.1; se ponen en consideración en un principio, varias soluciones creativas; sobre las que se realizó análisis, prueba y benchmarks de las mismas:

1. Implementar una herramienta de caching de base de datos (el recurso más lento de toda la arquitectura), local a la estación de trabajo del desarrollador de software; junto con la persistencia de archivos de sesiones en el mismo centro de desarrollo en el que el desarrollador de software resida (otro recurso con demoras en su acceso).
2. Virtualizar un escritorio completo para cada desarrollador de software, con el mismo sistema operativo que el desarrollador utiliza en su estación de trabajo, pero ubicado en la locación que posee los recursos más críticos en cuanto a accesos y demoras (alojarlo en esa misma red). En el caso de aplicación de este trabajo, sería un escritorio Windows virtualizado en Buenos Aires.
3. Virtualizar un escritorio completo para cada desarrollador de software, pero con el mismo sistema operativo que se utiliza en el ambiente productivo del sistema de software para usuarios finales; a su vez, alojado en la red con los recursos más lentos. En el caso de aplicación de este trabajo, sería un escritorio Linux, nuevamente virtualizado en Buenos Aires.
4. Virtualizar en la red con los recursos de acceso lento, una computadora con acceso únicamente vía Telnet o SSH (sin interfaz gráfica), y que cuente con algunas de las herramientas básicas necesarias para ejecutar el sistema en un ambiente de desarrollo (servidor web, servidor de aplicaciones, máquina virtual de Java, etc).

En particular para la primer solución planteada, Oracle TimesTen fue analizado en este aspecto, y se puede obtener una descripción de esta herramienta en el capítulo 1.3. Si bien la herramienta logra cierta ganancia en la velocidad de resolución de consultas respecto de la base de datos original, esta sólo puede ser alcanzada luego de reiterado ejercicio de la herramienta en la estación de trabajo local, y una vez que se haya poblado lo suficiente en cuanto a datos. Sumado a este problema, la configuración de la herramienta es bastante tediosa y no justifica la ganancia de velocidad. En cuanto a la persistencia de sesiones de disco locales, se expone un benchmark realizado en el anexo 1. Nuevamente, si bien se evidencia cierta ganancia, no es la suficiente como para considerarse una solución total al

problema original. Debido a estos inconvenientes, si bien se implementa este punto en la herramienta modificada, esta opción no puede significar la solución completa al problema.

La segunda y tercer opción, si bien suponen un entorno cómodo de trabajo para el desarrollador (un escritorio completo) y veloz en el acceso a los recursos críticos (locales a la misma red remota); también significa una alta carga de transmisión de datos entre las redes distantes de los diferentes centros de desarrollo, al requerir una conexión de escritorio remoto por cada desarrollador de software. Esto puede tornarse en altos gastos de infraestructura de red para soportar dicho tráfico, y lentitud en el acceso cuando el ancho de banda consumido se acerque a los límites disponibles. A su vez, no escala como solución cuando se incorporen más miembros al equipo de desarrollo, y el acceso esté limitado. Otra desventaja es la duplicación de todo el entorno de desarrollo en los escritorios locales y virtuales de cada desarrollador, generando una carga inicial de instalación y configuración bastante grande. Por ende, estas soluciones también son descartadas.

Se decide así, optar por una solución que combina los mejores aspectos de la tercer y cuarta opción. Así se despliega en la red con los recursos lentos, un equipo por desarrollador con el mismo sistema operativo productivo, pero con un conjunto mínimo de software instalado, que permite la ejecución del aplicativo web de modo ágil y símil a como sucedería para usuarios finales, pero dejando toda la carga de edición (y de aplicaciones CASE) permaneciendo en la estación de trabajo actual y local del desarrollador. Así la carga de transferencia entre ambas redes puede minimizarse mientras que no se afecta negativamente la operatoria de la aplicación (al contrario, se logra un entorno más fidedigno al productivo). Al restringir el acceso a las máquinas virtuales usando tan sólo SSH, se gana en cuanto a seguridad, ya que como se vió en el capítulo 3.2.1, el administrador puede otorgar acceso sólo a los usuarios indicados a la red de máquinas virtuales (o incluso a cada máquina virtual en particular), revocar el acceso cuando ya no sea necesario, y más importante, mantener abiertos tan sólo unos pocos puertos a estas instancias virtuales, para que otro software no autorizado no pueda realizar tareas de intrusión.
Al contar con tan sólo el software mínimo de aplicación en la máquina virtual remota, se debe suministrar un mecanismo ágil y conveniente para la transferencia y actualización de datos entre ambos entornos (especialmente el código fuente aplicativo). Más adelante en el capítulo 5.3 se abarca el análisis y modelo logrado para la sincronización de software entre estos dos entornos; y en el capítulo 5.3.2 se presenta la herramienta de software explícita implementada para este cometido.
Además de poder ejecutar la aplicación web remota y de poder mantenerla actualizada en cuanto a cambios en el software; llegado cierto punto del ciclo de vida del desarrollo de software, siempre será necesario realizar cierta interacción con el software remoto, como ser pruebas. Si bien el acceso SSH podría ser suficiente para interactuar con las herramientas (mediante instrucciones de línea de comandos), puede tornarse engorroso para desarrolladores con pocos conocimientos sobre la materia, además de propenso a errores. Para minimizar este problema y lograr un entorno de desarrollo agradable sobre esta solución, se implementan las acciones más frecuentes sobre la herramienta CASE del

desarrollador a modo de plug-in. Los casos de uso implementados son los que se definieron en el capítulo 4.2, mientras que la implementación en sí de los mismos en el CASE, con la tecnología apropiada para el caso, se desarrollará luego en el capítulo 5.4.2.

Así queda una arquitectura de hardware y software como la presentada en el capítulo 4.1, que sirve a los fines expuestos, minimizando los inconvenientes, la carga de configuración y transferencia de datos, y cuestiones de seguridad en el acceso; mientras que se mantiene un entorno de trabajo cómodo para el desarrollador, y propenso a menos fallas que el anterior al ser más parecido al entorno productivo.
El entorno de trabajo local del desarrollador varía mínimamente, pudiendo convivir los dos esquemas de trabajo y pudiendo seleccionar el más conveniente en cualquier momento (si por ejemplo, hubiera un corte en el suministro de red y no pudiera accederse a las máquinas virtuales remotas).

5.1 - Plataforma de virtualización

La explosión que viven las grandes compañías de la tecnología de la información en los últimos años, comienza a generar algunos problemas a la infraestructura de hardware. Es por ello que para mejorar el uso y la disponibilidad de los servidores en los centros de datos, en este trabajo se tomó la decisión de sumar una serie de software de virtualización para resolver esta situación (particularmente el paquete de soluciones de Oracle, por los motivos contados en el capítulo 3.2).

Particularmente, los desafíos que conllevan en la actualidad a crear conciencia de la necesidad de virtualizar servidores son:

- Consolidar y validar la infraestructura tecnológica, altamente demandante de servidores, para reducir el tamaño de los centros de datos y acompañar el crecimiento de la compañías.
- Alcanzar alta disponibilidad y escalabilidad en los servidores y aplicaciones requeridas para el desarrollo de software (y posiblemente también para el servicio a usuarios finales), para simplificar la administración de la infraestructura tecnológica, reducir costos operacionales y soportar el constante aumento de desarrolladores de software (y llegado al caso, también clientes).
- Reducir el consumo de electricidad en el centro de datos, para ahorrar y, al mismo tiempo, cumplir con una política de sustentabilidad del medio ambiente.

En concreto, en la implementación de este trabajo, incorporar Oracle VM permitió reducir sustancialmente el hardware requerido para el desarrollo. A su vez, se alcanzó una implementación acelerada de las aplicaciones de desarrollo en el nuevo ambiente, y una mejor escalabilidad y confiabilidad necesarias en los sistemas desplegados; al contar con un proceso de soporte simplificado, con menor costo y con un único punto de contacto para toda

la aplicación de software y hardware durante la fase del desarrollo de software.

En la implementación realizada, también se hace uso de Oracle VM Templates, que son máquinas virtuales preinstaladas y preconfiguradas de software empresarial, lo que permite acelerar la implementación del software en cuestión. En conjunto, este software junto con Oracle VM y Oracle Unbreakable Linux (sistema operativo usado para la administración de las máquinas virtuales, como se vió en el capítulo 3.2), permiten reducir los costos de servidores, a la vez que se proporciona una infraestructura de tecnología escalable y confiable, para admitir una base de desarrolladores de software en rápido crecimiento.

Como se mencionó al comienzo de este capítulo, el crecimiento dinámico de las empresas de tecnología comienza a ser una carga para la infraestructura de hardware. Para estar a la altura de las demandas de los propios empleados de las compañías, constantemente se necesita de más hardware y, a su vez, más espacio físico, sistemas de refrigeración y energía. Las compañías también se enfrentan a problemas de gestión cada vez mayores relacionados con la utilización y disponibilidad de los servidores en todos sus centros de datos.
Al buscar una solución de virtualización, se necesita encontrar una solución con soporte que pudiera ajustarse sin problemas dentro de la mayoría de las infraestructuras de hardware existentes en el mercado. Durante el análisis de las opciones disponibles, se concluyó que Oracle VM estaba mejor equipado para satisfacer las necesidades de una amplia cantidad de empresas en situaciones de crecimiento rápido de recursos humanos (específicamente, desarrolladores de software).

La implementación desplegada con Oracle VM, ha podido consolidar cientos de servidores físicos en menos de diez servidores centrales, cada uno ejecutando múltiples máquinas virtuales. Esta consolidación permitió escalar gabinetes de servidores, mejorando a la vez la utilización de cada servidor, minimizando y centralizando el espacio de disco, mejorando la disponibilidad del equipamiento clave de hardware, simplificando la administración y reduciendo sustancialmente el espacio y el uso de energía totales.
El uso de Oracle VM Templates permitió lograr un tiempo de llegada más rápido a la solución objetivo, junto con un menor costo de operaciones, a través de la creación de plantillas de software estándar que incluyen Linux y el software de aplicación requerido (detallado en el capítulo 4), que se crean una vez y luego son replicadas en otras máquinas virtuales [OraPress09].

Resumiendo, a modo de conclusión de la implementación hecha de virtualización, estos son algunos de los hitos logrados (más adelante, en el capítulo 6.1, donde se exponen las conclusiones de todo el trabajo, se detallan números concretos alcanzados en la implementación):

- Se implementó el software de virtualización de servidores Oracle VM, para soportar el gran volumen de transferencia de datos entre distintos centros de desarrollo de software.

- Se consolidó y escaló la infraestructura del centro de datos, para soportar el crecimiento en las operaciones de desarrollo.
- Se implementaron máquinas virtuales, reduciendo el consumo eléctrico y el espacio físico del centro de datos.
- Se alcanzó una reducción en los costos de infraestructura, servidores y electricidad por la virtualización de servidores.
- Se simplificó el manejo de la infraestructura con alta disponibilidad, con la capacidad de migrar fácilmente un servidor a otro sin requerir de un corte en el servicio.
- Se replicaron las configuraciones por defecto de software con Oracle VM Templates, reduciendo costos operativos y acelerando el tiempo de llegada a desarrolladores existentes y nuevos en el equipo de desarrollo.

5.2 - Comunicación mediante conexiones seguras

Como se presentó durante todo el capítulo 3.2, la seguridad en el acceso a las máquinas virtuales no es un tema que deba tomarse a la ligera. Accesos desautorizados podrían causar pérdidas significativas a cualquier compañía. Por el otro lado, implementar multitud de reglas de seguridad en el acceso torna la usabilidad del sistema en algo ineficaz para los usuarios finales (desarrolladores de software), especialmente para aquellos profesionales de IT que no poseen tantos conocimientos técnicos, pero que igual requieren acceso al ambiente virtualizado (como profesionales de experiencia de usuario). Entonces, hay que balancear el hecho de, contar con un nivel de seguridad adecuado, que asegure que sólo los usuarios necesarios dentro de la red adecuada puedan acceder al entorno virtual, sin comprometer la seguridad del sistema en su conjunto, y sin generar una mala experiencia en el acceso de los usuarios en sí; como podría resultar el caso de tener que ingresar manualmente múltiples nombres de usuario o contraseñas, direcciones IP, requerir el ingreso a través de múltiples redes físicas o locales, etc.

Una forma ampliamente adoptada de automatizar el acceso remoto, manteniendo la seguridad y sin entorpecer la experiencia de los usuarios, es mediante la aplicación de un **esquema de clave pública/privada**, también llamado "**criptografía asimétrica**".
La criptografía asimétrica es el método criptográfico que usa un par de claves para el envío de mensajes. Las dos claves pertenecen a la misma persona; una clave es pública y se puede entregar a cualquier otra persona sin comprometer la seguridad, mientras que la otra clave es privada y el propietario debe resguardarla de modo que nadie tenga acceso a ella (o al menos nadie que no esté autorizado a leer los mensajes que el propietario reciba). Si bien el contenido de las dos claves es distinto, ambas están "enlazadas" mediante uno o más métodos matemáticos complejos (como suelen ser la factorización de números enteros, logaritmos discretos y relaciones de curva elíptica, entre otros). Además, los métodos criptográficos garantizan que esa pareja de claves sólo se puede generar una vez, de modo que se puede asumir que no es posible que dos personas hayan obtenido casualmente la

misma pareja de claves. La fortaleza del método radica en el hecho de que es imposible (o bien, no es factible computacionalmente hablando), que dada una clave privada correctamente generada, se pueda determinar su correspondiente clave pública.

Este método se diferencia de la criptografía simétrica, en el hecho básico de que este último usa una misma y única clave para cifrar y descifrar los mensajes (las dos partes que se comunican deben ponerse de acuerdo de antemano sobre la clave a usar).
Si el remitente usa la clave pública del destinatario para cifrar el mensaje, una vez cifrado, sólo la clave privada del destinatario podrá descifrar este mensaje, ya que es el único que la conoce. Por lo tanto se logra la confidencialidad del envío del mensaje, nadie salvo el destinatario puede descifrarlo.
Si el propietario del par de claves usa su clave privada para cifrar el mensaje, cualquiera puede descifrarlo utilizando su clave pública. Por lo tanto, en este caso se consigue la identificación y autentificación del remitente, ya que se sabe que sólo pudo haber sido él quien empleó su clave privada (salvo que alguien se la hubiese podido robar). Esta idea es el fundamento de la firma electrónica.

Los sistemas de cifrado asimétricos se inventaron con el fin de evitar por completo el problema del intercambio de claves de los sistemas de cifrado simétricos. Con las claves públicas no es necesario que el remitente y el destinatario se pongan de acuerdo en la clave a emplear. Todo lo que se requiere es que, antes de iniciar la comunicación secreta, el remitente consiga una copia de la clave pública del destinatario. Es más, esa misma clave pública puede ser usada por cualquiera que desee comunicarse con su propietario. Por lo tanto, se necesitarán sólo N pares de claves, por cada N personas que deseen comunicarse entre sí.
Una analogía con este método de cifrado es la de un buzón con una ranura para el correo postal. La ranura del buzón está expuesta y accesible al público; su ubicación (la dirección y número de la calle) es, en esencia, la clave pública. Alguien que conozca la dirección de la calle puede ir a la puerta y colocar un mensaje escrito a través de la ranura; sin embargo, sólo la persona que posee la llave (clave privada) puede abrir el buzón de correo y leer el mensaje.

La mayor ventaja de la criptografía asimétrica es que la distribución de claves es más fácil y segura, ya que la clave que se distribuye es la pública manteniéndose la privada para el uso exclusivo del propietario. Aún así, este sistema tiene algunas desventajas:

- Se requiere de **mayor tiempo de proceso de los mensajes**, para cualquier longitud de clave y de mensaje propiamente dicho. Esto se debe al hecho de requerir de la intervención de un mecanismo de codificación y decodificación de los mensajes. A mayor longitud de claves, mayor tiempo de procesamiento.
- Para asegurar un nivel de seguridad adecuado, **las claves deben ser de mayor tamaño que las simétricas** (generalmente son cinco o más veces de mayor tamaño).
- El **mensaje cifrado ocupa más espacio** que el original. Esto se debe no sólo al reemplazo del mismo por otro codificado, sino al hecho de que se requiere adjuntar la

firma digital calculada sobre cierta parte del mensaje (digesto del mensaje, o *fingerprint* en inglés).

Herramientas como PGP, SSH o la capa de seguridad SSL que utiliza el protocolo TCP/IP, utilizan un híbrido formado por la criptografía asimétrica para intercambiar claves de criptografía simétrica, y la criptografía simétrica para la transmisión de la información.

Afortunadamente, el software implementado para la transmisión de mensajes entre los entornos local y virtual (Plink, de la herramienta PuTTY, ver capítulo 4), ya posee su propio método de autenticación basado en el intercambio de claves de criptografía asimétrica. Esto no es casualidad, y obviamente se tornó en una característica decisiva en la elección de este software, durante la fase de diseño de la arquitectura a desplegar. Dicha herramienta se llama “PuTTYgen” (acrónimo de PuTTY Key Generator) y es una herramienta libre que se distribuyen junto con el mismo paquete de instalación de PuTTY, y que sirve para generar claves para utilizar con el cliente de SSH de PuTTY, con el agente de autenticación de PuTTY (Pageant), y otros programas en la línea de producción de PuTTY.

Configurar un esquema de autenticación de clave pública con PuTTY, para acceder a un equipo remoto en particular, es un procedimiento que se configura una sola vez, y consta de los siguientes tres pasos:

1. **Generación del par de claves pública/privada**: Como se mencionó recién, se hace uso de la herramienta PuTTYgen para llevar a cabo esta tarea. La misma posee una interfaz de usuario bastante intuitiva para el caso. Lo realmente interesante de esta aplicación, es que además de contar con la posibilidad de encriptar las claves generadas mediante el ingreso opcional de una contraseña; para generar la clave abrirá una ventana en donde se solicita al usuario que realice movimientos con el puntero del mouse. Esto es a fin de generar cierta aleatoriedad adicional, para ser usada en la generación misma de la clave, dando otro mecanismo de seguridad adicional, casi imposible de reproducir por un actor malintencionado.
   Una vez que la herramienta posee todos los datos necesarios, se puede finalmente crear el par de claves, copiarlas al portapapeles y/o exportarlas a archivos.
   Además, desde PuTTYgen es posible recuperar la clave pública a partir de la clave privada, por si la misma hubiese sido extraviada; no pudiéndose realizar la recuperación en el sentido contrario, por razones obvias de seguridad.
2. **Instalar la clave pública en el equipo remoto**: Esto se realiza copiando el contenido de la clave generada en el archivo de claves autorizadas para el acceso remoto, del sistema operativo del equipo en cuestión (en entornos Unix, este archivo suele ser ~/.ssh/authorized_keys). Este es un archivo de texto plano y puede editarse con cualquier editor de textos. Dado que para este caso de aplicación, se configura el acceso para la cuenta del mismo usuario, no se requieren permisos adicionales para editar el archivo.
3. **Verificar que la autenticación de clave pública funciona**: PuTTY ofrece también una herramienta visual dentro de su aplicativo para esto (aunque bien podría probarse

directamente con cualquier aplicativo de PuTTY, como Plink). Pero usando esta herramienta, además de probar la conectividad deseada, podemos dejar salvado el perfil de conexión, con la clave privada generada (dejando adicional el ingreso de contraseña, en caso de que se hubiera seleccionado una); y proveer un acceso simple y permanente, a la consola de comandos del equipo remoto. De hecho, justamente esto fue realizado en la implementación de este trabajo, de modo que el desarrollador pueda disponer desde su escritorio local, de un acceso directo mediante línea de comandos (y con autenticación asimétrica incorporada) a su máquina virtual remota.

Hasta aquí la generación de la clave y el uso dentro de la misma en la misma herramienta de PuTTY; pero de poco vale el método si no está integrado dentro de la herramienta CASE desarrollada, de modo que las acciones invocadas desde el IDE Eclipse puedan beneficiarse de la autenticación remota automática con el entorno.
Al usar Plink como software de ejecución de comandos remotos (ver capítulo 4) en el plugin broker desarrollado, la integración con las claves criptográficas generadas es inmediata, dado que se trata de otro software más, dentro del mismo paquete de PuTTY. Plink acepta un atributo adicional en su forma de invocación para poder especificarle la ruta de un archivo de clave privada de PuTTY, para usar en la autenticación remota (atributo -i). Por ende en la implementación de la herramienta CASE desarrollada, basta con agregar dicha clave al paquete exportado del plugin, y suministrarla del modo debido a los comandos ejecutados por medio de Plink. Para ver un ejemplo de aplicación directa, de la implementación del mecanismo de autenticación mediante Plink, en el software CASE desarrollado, ver el capítulo 5.3.2.

5.3 - Modelo de sincronización

Una parte importante de este trabajo, notablemente reiterada a lo largo de todo este informe, es el uso del término de “sincronización” de archivos, en lugar de meramente copia o traslado. Esto no es un hecho menor, ya que en efecto, el principal objetivo de la arquitectura desplegada y de la herramienta CASE desarrollada, es que, en su conjunto, mejoren sustancialmente la velocidad de desarrollo y pruebas del software, bajo un ambiente similar al descripto con anterioridad.
Si por cada cambio en el software (incluso ediciones que podrían significar el cambio de unas pocas líneas de código fuente en unos pocos archivos) fuera necesario copiar todo el árbol de directorios del repositorio de código fuente, desde el entorno local del desarrollador a su entorno virtual remoto; la ganancia del método probablemente sería muy escasa. La velocidad adquirida en la ejecución del software en la misma red que aloja los recursos lentos, se vería opacada en contrapartida por la lentitud del proceso de copia; sin siquiera mencionar los problemas que ocasionaría sobre la performance de la red, y el consumo de recursos de la misma (peor aún durante copias concurrentes de múltiples desarrolladores, y

a la larga generaría un serio problema de escalabilidad sobre el equipo de desarrollo y sus herramientas de trabajo).
Copiar tan sólo los archivos modificados en lugar de todo el repositorio, ante cada instante en que el desarrollador decide que quiere ver puestos en práctica sus cambios sobre el código fuente; mejora sustancialmente la velocidad y carga de recursos del método (así como el tráfico de datos sobre la red). Aún así, esta opción tiene sus falencias en varios casos particulares:

- Si se cambia tan sólo un carácter (o un byte) en un archivo, sigue siendo necesario copiarlo completo desde la ubicación local a la otra remota a través de la red. Si el archivo es particularmente grande, el método pierde total eficacia y se desperdicia por completo la transferencia, al reemplazar una copia existente del archivo, por otra casi idéntica.
- Si se realizan cambios similares sobre muchos archivos, como puede ser la refactorización de software (técnica de la ingeniería de software para reestructurar código fuente, alterando su estructura interna sin cambiar su comportamiento externo); en el caso en que por ejemplo se reemplace el nombre de una variable o método por otro más idóneo, en todas las ocurrencias del mismo en todo el repositorio; nuevamente se deben copiar completos todos los archivos en donde ocurrió dicho reemplazo. Esto sucedería sin ningún proceso ni análisis lógico sobre el cambio realizado, que en esencia, es el mismo sobre todos los archivos, incluso sobre todas las ediciones sufridas en dichos archivos. En términos menos técnicos, se realizaría una copia bruta de múltiples archivos, sin considerar qué ha cambiado en ellos y cómo puede optimizarse la transferencia para ello.
- Cambios en los metadatos de los archivos, como podrían ser los permisos de lectura, de escritura o de ejecución de los mismos; o incluso, cambios sobre la fecha de modificación del archivo (en esencia, cambios sobre los datos de descripción del archivo que no implican cambios en su contenido); aún así significarían la copia de todo el archivo de un extremo a otro de la arquitectura. Nuevamente, esto se sucedería sin mucho sentido y con un desperdicio total del ancho de banda y tiempos consumidos.
- Un concepto similar al anterior aplica al renombre de archivos o cambio de ubicación de los mismos, en el cual un simple proceso de copia volvería a trasladar otra copia del mismo archivo, sin considerar el cambio de ruta del mismo. Peor aún, sin ningún chequeo de consistencia del árbol de directorio destino de la copia, hasta podrían quedar recursos duplicados en este caso; debido a archivos que cambiaron de ubicación, se realizó la copia de los mismos según la nueva ubicación, pero nunca fueron purgados de la ubicación antigua (ya que una simple copia los entiende como nuevos archivos, en lugar de archivos existentes en una nueva ubicación, o con un nuevo nombre).

Como se ve, para lograr una arquitectura eficiente, es necesario que la manutención del software (en especial del código fuente), sea realizada a través de mecanismos y procesos lógicos que entiendan el sucesivo procesamiento de los archivos, en pos de que las

transferencias de datos a través de la arquitectura desplegada, sean las menores posibles, sin desperdiciar tiempo ni recursos.
Para ello, se opta por estudiar los posibles procesos de sincronización de archivos, y en particular, qué herramientas de software se encuentran disponibles para este fin, y que puedan incorporarse fácilmente a la arquitectura. La sincronización de archivos es utilizada para mantener la misma versión de archivos en múltiples dispositivos (también se habla de sincronización de datos cuando dos dispositivos se actualizan de forma que contengan los mismos datos). Por ejemplo, sincronizar la libreta de direcciones de un teléfono con la libreta de direcciones de una computadora [Alegsa14]. Algo importante en todo proceso de sincronización, es el establecimiento de consistencia entre datos en fuentes remotas y de la continua armonización de los datos en el tiempo. Si se agrega, modifica o elimina un archivo de una ubicación, el proceso de sincronización agregará, modificará o eliminará el mismo archivo en las otras ubicaciones.

Básicamente, existen dos tipos de procesos de sincronización de archivos [Alegsa14]:

- Sincronización de una sola vía (one-way): también llamada mirroring o espejado, los archivos son copiados sólo desde una ubicación fuente hacia una (o más) ubicación destino, pero ningún archivo es copiado en el sentido inverso. Por lo tanto las modificaciones hechas en el destino no afectan a la fuente original.
- Sincronización de dos vías (two-way): los archivos son copiados en ambos sentidos, manteniendo ambas ubicaciones sincronizadas una con la otra.

Para el caso de aplicación de este trabajo, basta con optar por un software de sincronización de una sola vía, ya que la edición de archivos sólo se realiza en la fuente (computadora del desarrollador de software), mientras que los archivos de la ubicación remota destino tan sólo son ejecutados, no modificados. Al reducir el proceso de sincronización a una sola vía, se ahorra tiempo de procesamiento y consumo de recursos; mientras que un proceso de dos vías requeriría controles de consistencia en el sentido inverso que no son necesarios en este caso.

Además, se espera que el software de sincronización aplicado, cuente con algunas otras posibles características avanzadas de este tipo de programas [Alegsa14]:

- Para lograr más eficiencia, lo ideal es que se sincronicen sólo aquellos datos que han cambiado, en lugar de copiar todo un archivo completo (punto fundamental expuesto con anterioridad).
- Posibilidad de compresión de datos, si es que la sincronización se hace a través de una red como es el caso de aplicación de este trabajo.
- Capacidad de detección de conflictos, por ejemplo, cuando hay algún archivo que no está sincronizado correctamente (diferentes versiones de ambos lados).
- Posibilidad de previsualizar cualquier cambio antes de que se realice.
- Posibilidad de ver las diferencias entre archivos individuales.

5.3.1 - Alternativas estudiadas

Buscando posibles programas de sincronización de archivos que se encontraran disponibles, y que fueran libres y pudieran acomodarse en nuestra arquitectura de software, se llegó en primer instancia a **Unison**.

Unison es una herramienta para sincronización de archivos para Unix y Windows. Permite que dos réplicas de un conjunto de archivos y directorios sean almacenadas en diferentes equipos (o diferentes discos en el mismo equipo), modificadas de forma separada, y luego puestas al día propagando los cambios en cada réplica a la otra. En otras palabras, cuando dos dispositivos son sincronizados, el usuario puede estar seguro de que la versión más actualizada de cada archivo estará disponible en ambos dispositivos; sin importar en cuál de ellos hubo modificaciones.
Unison comparte cierto número de características con herramientas como programas de control de versiones (Git, CVS, Subversion, Mercurial, etc.), sistemas de archivos distribuidos (Coda, Lustre, etc.), utilidades de mirroring (ver próximo capítulo) unidirectional (como Rsync), y otros sincronizadores. Sin embargo, hay varios puntos en los que difiere de la mayoría de estas herramientas:

- Unison corre tanto en Windows, como en muchas variedades de sistemas Unix (Solaris, Linux, OS-X, etc.) y Android. De hecho, Unison trabaja por sobre las plataformas, permitiendo sincronizar por ejemplo, una notebook Windows con un servidor Unix.
- A diferencia de simples utilidades de mirroring o backup de archivos, Unison puede procesar actualizaciones en ambas réplicas de una estructura de directorios distribuida, sin el overhead adicional de los sistemas de control de versiones. Las modificaciones que no generan conflictos son propagadas automáticamente; mientras que aquellas que sí generan conflictos, son detectadas y mostradas al usuario para que seleccione en qué forma proceder.
- Pese a trabajar sobre sistemas de archivos distribuidos, Unison es un programa de nivel de usuario; por lo que no hay necesidad de poseer privilegios de superusuario en ninguno de los equipos involucrados en la sincronización.
- Unison funciona entre cualquier par de máquinas conectadas a internet (se comunica mediante el protocolo TCP/IP), comunicándose sobre un enlace de socket directo, o bien a través de un túnel de conexión SSH encriptado.
- Es cuidadoso con el ancho de banda de red; las transferencias de pequeñas modificaciones a grandes archivos son optimizadas haciendo uso de un protocolo de compresión similar al de Rsync (ver capítulo 3.3.4). Implementa un versión modificada del algoritmo de Rsync desarrollado originalmente por Andrew Tridgell; que al igual que el original, transfiere sólo las partes de cada archivo que han sido modificadas, de modo que es más rápido que transferir los archivos completos.
- Unison es flexible ante errores, y fue diseñado de modo robusto en caso de fallas de comunicación en el programa o en el sistema. Deja las réplicas y sus propias estructuras internas en un estado sensible en todo momento, incluso en caso en que la ejecución del programa termine de forma anormal (problemas en la comunicación

de red por ejemplo); de modo que pueda restaurarse fácilmente la sincronización en el punto interrumpido.

Como se ve, a primer golpe de vista Unison se torna en una opción interesante para implementar la sincronización de código fuente en la arquitectura planteada. Tal el es caso que resultó en la primer opción puesta en práctica para dicho objetivo. Si bien al comienzo y con modificaciones simples en el código fuente, la herramienta mostraba ser útil al fin requerido; luego de algunas pruebas más complejas comenzaron a notarse los problemas de aplicar esta herramienta como parte de la solución:

- Como se mencionó en este capítulo, el tipo de sincronización usado por Unison es de dos vías (ver capítulo 5.3). Lo que para el caso de uso de este trabajo genera overhead redundante y conexiones entrantes a la máquina local del desarrollador de software para chequeo de conflictos, totalmente innecesarias.
- Dado que la sincronización aplicada de dos vías es maestro/maestro con resolución de conflictos manual; si hubieran cambios en la máquina virtual, estos se propagarían al origen (máquina local del desarrollador de software en este caso), situación que no se desea. Mucho menos resolver conflictos manualmente, en caso de modificación del mismo archivo en ambos lados (el servidor web remoto podría modificar durante la ejecución algún archivo del repositorio remoto como logs, configuraciones, cachés en disco, etc; en cualquier caso, no se desea que dichas modificaciones sean persistentes ni que se transfieran al origen en ningún caso).
- El programa en sí incluye interfaz gráfica y demás herramientas junto con su instalación, que no son necesarias para la arquitectura de este trabajo. Esto hace que su instalación sea más grande y complicada de lo deseado.
- Dada su interfaz gráfica y gran cantidad de características, se torna difícil para utilizar como herramienta pura de línea de comandos (forma en que desea usarse transparentemente a través del plugin broker de la herramienta CASE).

Luego de la experiencia en el uso de Unison para la sincronización de archivos, se concluye que si bien tiene características interesantes y útiles al fin, se requiere una alternativa que principalmente cumpla algunas nuevas premisas:

- Para el caso de estudio aplicado, la sincronización debe ser de una sola vía; de modo de evitar el overhead innecesario generado al realizar comprobaciones de dos vías y su correspondiente tráfico de red.
- Se requiere una herramienta que sea más fácil de instalar (en lo posible que sea tan sólo un archivo ejecutable) y de implementar en la herramienta CASE (que posea unas pocas opciones de línea de comandos para el fin requerido).

Buscando programas de sincronización de una sola vía para aplicar en la arquitectura, se llega a indagar en la opción de un programa de control de versiones de archivos, específicamente **Mercurial**; que si bien no es una herramienta de sincronización pura, merece la pena investigar si encaja para el cometido.

Mercurial es un sistema de control de versiones multiplataforma y distribuido para desarrolladores de software. Está implementado principalmente haciendo uso del lenguaje de programación Python, pero incluye una implementación binaria de diff escrita en C. Mercurial fue desarrollado originalmente para funcionar sobre Linux, pero ha sido adaptado para Windows, Mac OS-X y la mayoría de los otros sistemas de tipo Unix. Mercurial es, principalmente, un programa para la línea de comandos (aunque están disponibles interfaces gráficas de usuario en forma de extensiones). Todas las operaciones de Mercurial se invocan como opciones dadas a su programa motor, hg (cuyo nombre hace referencia al símbolo químico del mercurio).

Las principales metas de desarrollo de Mercurial incluyen un gran rendimiento y escalabilidad; desarrollo completamente distribuido y colaborativo, sin la necesidad de un servidor; gestión robusta de archivos tanto de texto plano como binarios; y capacidades avanzadas de ramificación (branching) e integración de archivos (merging), todo ello manteniendo sencillez conceptual [Mackall06]. La distribución estándar incluye además una interfaz web integrada, útil para ser instalada en servidores Mercurial, para poder navegar fácilmente el árbol de revisiones del software.

Al igual que otras herramientas de versionado, como Git y Monotone, Mercurial usa digestos SHA-1 para identificar revisiones. Para acceder a repositorios a través de la red, Mercurial usa un protocolo eficiente, basado en HTTP, que persigue reducir el tamaño de los datos a transferir, así como la proliferación de peticiones (requests) y conexiones nuevas. Mercurial puede funcionar también sobre SSH, siendo el protocolo aplicado muy similar al basado en HTTP. Por defecto, usa un algoritmo de mergeo de tres vías, antes de invocar a otras herramientas de merge externas (un merge de tres vías realiza un análisis automatizado de diferencias entre dos archivos, considerando a su vez al origen de ambos, o el ancestro común).

Si bien el enfoque de aplicar una herramienta de control de versiones pareció interesante en un primer instante, no hizo falta indagar mucho en el tipo de solución que habría que implementar para llegar a la conclusión de que su complejidad excedería en grande la implementación deseada en la arquitectura de software. En [Omar11] se encuentra un tutorial para sincronizar copias locales y remotas de un web blog (WordPress) haciendo uso de un sistema de control de versiones (Mercurial). Si bien el proceso requiere de varias adaptaciones propias de implementar un workflow aplicable a una aplicación web basada en Java y Resin, en lugar de WordPress y MySQL; es realmente obvio que este enfoque no era el deseado por varias razones:

- Si bien la interacción con Mercurial se realiza mediante un sólo programa ejecutable de línea de comandos como es requerido, hay que realizar muchas operaciones sucesivas para los fines requeridos. Si bien esto es automatizable mediante scripting, se suma el problema de que ante fallas en la red, queden operaciones inconclusas y estados de repositorios inconsistentes, haciendo que se torne difícil de recuperar mediante la herramienta. Además, integrar todas estas instrucciones en el plugin

broker de la herramienta CASE, infiere una gran complejidad también.

- En caso de requerir operaciones manuales por parte del usuario (el desarrollador de software), se complejiza mucho su uso y hay grandes posibilidades de dejar todo el espacio de trabajo en estados altamente inconsistentes por procedimientos propios de la herramienta (sin siquiera mencionar los requerimientos técnicos necesarios para el caso, que todos los desarrolladores tendrían que incorporar).
- Ante conflictos en la sincronización de software, la herramienta aplicaría, al igual que Unison, merge remoto; y en caso de conflictos en ambos lados sobre los mismos archivos, se requiere acción del usuario nuevamente. Esta es una situación que se desea evitar; y si bien es posible configurar el proceso de sincronización para que ignore los cambios remotos, y siempre aplique los cambios locales en la copia remota del repositorio; esto complejiza aún más los scripts que deben desarrollarse e integrarse en la herramienta CASE.

Así es que con un poco de análisis y sin necesidad de implementar algo sobre esta opción, se concluye de forma temprana en que tampoco es la más adecuada. Entonces, volviendo un poco sobre Unison y lo investigado sobre dicha rama del software; se había concluído que se requiere algo similar en cuanto a su proceso de sincronización, pero debe ser de una sola vía y sin el overhead adicional del resto de sus herramientas. Si Unison basa su algoritmo de transferencia en **Rsync**… ¿por qué no probar directamente el uso de Rsync y así evitar todo el software y overhead periférico de aquellas soluciones que a su vez, lo aplican? Así fue que finalmente se dió con el software apropiado para el propósito, el cual se detalla a continuación.

5.3.2 - Rsync

Rsync es una versátil y pequeña utilidad increíblemente fácil de configurar en la arquitectura desarrollada (y presumiblemente en cualquier otra). En lugar de requerir de una sesión de FTP (File Transfer Protocol / Protocolo de transferencia de archivos), o cualquier otra forma de transferencia de datos mediante scripting, Rsync copia sólo los diffs (ver capítulo 3.3.1) de los archivos que sufrieron cambios en lugar de los archivos completos (esto acelera las actualizaciones y especialmente sobre enlaces lentos; FTP realizaría la transferencia del archivo completo en lugar del diff, aún aunque hubiera cambiado tan sólo un byte de todo el contenido). Más aún, comprime los diffs durante el proceso (reduciendo aún más la cantidad de datos transmitidos y la carga en la red) y los transfiere a través de SSH (Secure SHell, intérprete de línea de comandos seguro, comunicación mediante un protocolo de red criptografiado que encripta la sesión del usuario) si deseamos seguridad [Holve99]. Otras características útiles de Rsync incluyen el soporte para la copia (además de archivos) de enlaces, dispositivos lógicos y configuraciones de pertenencia, grupos y permisos en el sistema de archivos; listas de exclusión mediante expresiones regulares; pipelining de transferencia de archivos para minimizar la latencia y soporte para servidores Rsync anónimos o autenticados, ambos sin requerir privilegios de root (super-usuario en entorno

Unix).

Así, Rsync se torna una herramienta útil para realizar procesos de backup, sincronización y mirroring de código fuente (un mirror o espejo, es un sitio web o un conjunto de archivos en un servidor que han sido copiados a otro servidor distinto, de modo que el sitio o archivos originales queden disponibles desde más de una única ubicación). En particular, en este trabajo se usa para sincronizar código fuente, desde la computadora del desarrollador de software, hacia su máquina virtual. No sólo se sincroniza el árbol de archivos de la aplicación web principal, sino a su vez, otras áreas clave del software como ser recursos estáticos de la aplicación (por ejemplo imágenes, scripts, configuraciones, cachés en disco, etc). El proceso de sincronización se realiza de modo automatizado mediante una implementación de scripting, la cual se detalla a continuación. Esta implementación de la herramienta, resulta en la solución que mantiene en sincronía el software del entorno local de desarrollo con su contrapartida remota; sobre la que ejecuta el servidor de pruebas del ambiente de desarrollo.

Lo que se requiere en este trabajo es realizar una copia eficiente de una selección de archivos entre computadoras heterogéneas y remotas (para el caso de uso implementado, la computadora del desarrollador posee una instalación de Windows, mientras que la contrapartida remota posee una distribución de Linux, similar al entorno productivo; esto no necesariamente debe ser así, ya que la herramienta de scripting implementada para el proceso en cuestión, puede ser fácilmente adaptada para ser ejecutada entre dos arquitecturas cualesquiera, que tuvieran la facultad de poder ejecutar el software desplegado, o en su defecto, versiones portadas del mismo). No se precisa de mayores características disponibles en costosas soluciones de backup; simplemente actualizar las copias de los archivos entre cada par de estas computadoras, bajo demanda del desarrollador de software. Debido a la velocidad de transferencia incremental de Rsync y su facilidad de configuración, se torna una elección obvia para este propósito. Además, como se muestra más adelante en esta misma sección, si el desarrollador de software requiriera adaptar esta configuración a sus fines particulares, puede hacerlo muy fácilmente también, editando tan sólo 2 archivos de texto plano (uno en cada una de sus máquinas en la arquitectura desplegada), que poseen una sintaxis muy simple y comprensible.
Para el caso de uso mostrado, se tiene una máquina virtual Linux (por cada desarrollador de software) que posee una estructura de directorios bajo el directorio raíz de ejemplo “/datos”, para alojar los archivos de la aplicación web. Rsync posee una arquitectura cliente/servidor, en donde el cliente de Rsync entabla una comunicación con un demonio del mismo programa ejecutando en el servidor. El cliente Rsync puede conectarse al servidor Rsync directamente o mediante otros programas o protocolos de transporte de datos como rsh, ssh, etc [Garg05]. Se decide aplicar el uso de ssh a la transferencia de datos por cuestiones de seguridad y simpleza.

A partir de aquí, se realiza la muestra de una configuración del software que ejemplifica la solución obtenida a los fines de este trabajo, pero con la salvedad de que simplemente se muestran ejemplos y modelos acotados para la correcta comprensión de la herramienta y de

la implementación realizada de la misma. Para obtener un detalle más específico, puede recurrirse al anexo 3, en donde está la implementación completa de la herramienta con la configuración precisa utilizada en este trabajo, y las implementaciones correspondientes a los sistemas operativos instalados en cada extremo de la arquitectura de software desplegada.

El demonio de Rsync requiere un archivo de configuración rsyncd.conf. Para el caso de uso, realizamos una configuración similar al ejemplo mostrado a continuación:

```
[root@remoteVirtualMachine ~]# cat rsyncd.conf
use chroot = no
[lib]
        path = /datos/webserver/java
        read only = no
        comment = librerías de terceras partes
[classes]
        path = /datos/webserver/java
        read only = no
        comment = resin classes
[queries]
        path = /datos/application-config
        read only = no
        comment = sql queries
```

Esto significa:

- No cambiar a root (**chroot**, comando para convertirse en superusuario). Esto es necesario ya que el proceso será ejecutado como un usuario que no necesariamente es el root.
- **[classes]** especifica un módulo llamado “classes”, mediante este nombre es posible referenciarlo como se verá luego. Aquí particularmente, se configura la sincronización de los archivos de clases java al directorio necesario del servidor de aplicación Resin. Se muestran nomás los módulos [lib], [classes] y [queries] a modo de ejemplo de la nomenclatura de la configuración, pero la realidad es que para el proceso implementado, se definen varios módulos más, para especificar también archivos estáticos, archivos de configuración, cachés en disco, etc. (ver anexo 3).
- **path**, es el directorio local (en la máquina virtual) del sistema de archivos, en donde sincronizar cada módulo correspondiente.

Eso es todo lo que se necesita en el lado del servidor, además de obviamente, el demonio de Rsync ejecutando en segundo plano, para atender las solicitudes de sincronización. Una opción menos oportuna sería iniciar el demonio desde el cliente, usando ssh antes de comenzar con la sincronización, y terminar el proceso al finalizar la misma. Por cuestiones de

eficacia del método, en este trabajo se hacen ambas cosas: se mantiene en ejecución el demonio de Rsync en el servidor, y además se lo inicia antes de la sincronización desde el cliente (chequeo del proceso), por si por alguna razón estuviera caído el servicio al momento de requerirlo (o hubiera sido deliberadamente detenido). Así se logra una mayor efectividad en el método, logrando cierto nivel de recuperación automática ante fallas del software.

Para especificar el entorno de ejecución requerido del lado del cliente, a continuación se hace mención a las piezas de software y utilidades introducidas al inicio del capítulo 4 (Diseño de la herramienta CASE). El lector puede referir a dicha sección para rememorar la aplicación de cada pieza de software al objetivo del trabajo, o simplemente para obtener una breve descripción de la misma.

Del lado de Windows (el workstation del desarrollador de software en este trabajo), se requiere de Rsync y algún cliente ssh; afortunadamente existe software portado para el propósito. Una copia portada (o simplemente, port), es una adaptación del software de modo que un programa ejecutable pueda ser creado para un ambiente de computación o ejecución que es distinto de aquel para el cual fue diseñado originalmente. Rsync está disponible para Windows a través de la copia portada de Cygwin (durante la instalación de Cygwin, hay que seleccionar agregar Rsync como utilidad; para el caso de aplicación de este trabajo, se distribuye directamente el paquete de instalación de Cygwin con todas las utilidades requeridas ya descargadas). Como cliente ssh, se puede utilizar tanto ssh que viene en la distribución de Cygwin, como la herramienta de línea de comandos plink que se incluye en la distribución de Putty. Se selecciona la opción de comunicación mediante plink, ya que se torna más simple de configurar un método de autenticación automática que no requiera el ingreso de contraseña, mediante un par de claves pública y privada (ver capítulo 5.2); aunque de todos modos es posible usar cualquier cliente de ssh. Además, para mantener los comandos cortos y los scripts de comunicación más sencillos y fáciles de editar o replicar, se agregan los directorios que contienen los ejecutables de rsync y plink a la variable de ambiente “PATH” del sistema operativo del cliente. Esta variable contiene una lista de rutas en el sistema de archivos en donde el sistema operativo busca programas ejecutables, para los cuales no se especificara su directorio de ubicación al ser ejecutados. Particularmente en el sistema operativo Windows (el que ejecuta para el caso de uso implementado en el cliente), esta variable se llama %PATH% (el nombre de la misma bien pudiera cambiar entre distintas implementaciones o incluso entre distintas versiones del mismo sistema operativo, pero a fines prácticos siempre existe una variable de ambiente para este propósito y el paquete de instalación del software debe configurar las rutas al software utilizado, en esta variable).

Primero, se requiere iniciar el demonio de Rsync en el servidor (en la máquina virtual). Puede iniciarse desde el cliente (máquina local) con el siguiente comando:

```
plink -i <login_key.ppk> -v -t -l <username> <virtual_machine_IP>
rsync --daemon --port=<transfer_port> --config=/root/rsyncd.conf
```

en donde <login_key.ppk> es el archivo de clave privada de identificación generado en Putty (ver capítulo 5.2), <username> el nombre de usuario con el cual loguearse en la máquina virtual, y <virtual_machine_IP> la dirección IP de dicha máquina virtual. Estos parámetros son suministrados dinámicamente por el plugin del broker al script en cuestión, al momento de ejecutar la sincronización, de modo que las claves, nombres de usuario y máquinas virtuales puedan personalizarse para cada desarrollador en particular. Entonces el comando loguea al usuario indicado en el sistema de archivos remoto y ejecuta el demonio de rsync en el puerto suministrado <transfer_port> (para el caso de uso implementado, se hace uso del puerto 1873, que es el puerto por defecto de Rsync, pero podría usarse cualquier puerto que estuviera disponible en la arquitectura desplegada). Rsync queda ejecutando en segundo plano (como todo demonio) y plink retorna inmediatamente. Si se ejecutara rsync directamente a través del protocolo ssh, entonces se requeriría también de configurar un túnel de transporte mediante plink, desde un puerto local al puerto remoto que se haya especificado para la transferencia (lo que se conoce formalmente como forward de puertos), para que la copia local de rsync pueda conectar ambos puertos para la transferencia ejecutando la sincronización contra el puerto local definido. Una opción mejor y más fácil de implementar, es hacer uso del protocolo de transferencia propio de rsync (rsync://), que se encarga de configurar y entablar la comunicación automáticamente al puerto remoto seleccionado, sin necesidad de abrir el túnel ssh manualmente; por lo que se usa este último método. Así que sólo se necesita ejecutar rsync en la máquina cliente con el siguiente comando:

```
rsync -avz --del <local_path>
rsync://<username>@<virtual_machine_IP>:<transfer_port>/<module>
```

Este comando sincroniza incrementalmente el archivo o directorio <local_path> al directorio remoto especificado para el módulo <module> en la configuración del demonio ubicada en la máquina virtual remota (atributo “path” bajo el módulo en cuestión del archivo rsyncd.conf explicado anteriormente). Esto debe hacerse por cada módulo definido (cada colección de artefactos de software o de código fuente que se quiere sincronizar en distintos subdirectorios del sistema de archivos remoto). Dado que la copia de Rsync de la máquina cliente del caso de uso implementado (Windows) es la que viene en la distribución de Cygwin (ver capítulo siguiente), esta sólo comprende paths locales especificados según el formato de Cygwin; esto es por ejemplo, C:\datos\webserver\java\classes se debe traducir a /cygdrive/c/datos/webserver/java/classes en términos de Cygwin; y la herramienta broker implementada debe ser la encargada de realizar la traducción durante la sincronización (en el caso implementado, se realiza a través de un archivo de configuración que se detalla más adelante en esta misma sección).

Ahora bien, esta operatoria no es del todo útil si no se encuentra dispuesta en un script que se encargue automáticamente de realizar el proceso de inicialización de rsync y de sincronización para cada módulo de software necesario. Para automatizar el proceso, se

diseña un script de línea de comandos de Windows (runrsync.bat), para que sea invocado por el plugin del broker del IDE CASE ante la demanda del desarrollador de software. Aquí se presenta el diagrama de actividades con las tareas que ejecuta este proceso (si se desea ver una instanciación concreta de este diagrama, implementada en el caso de uso del trabajo realizado, puede referirse al anexo 3):

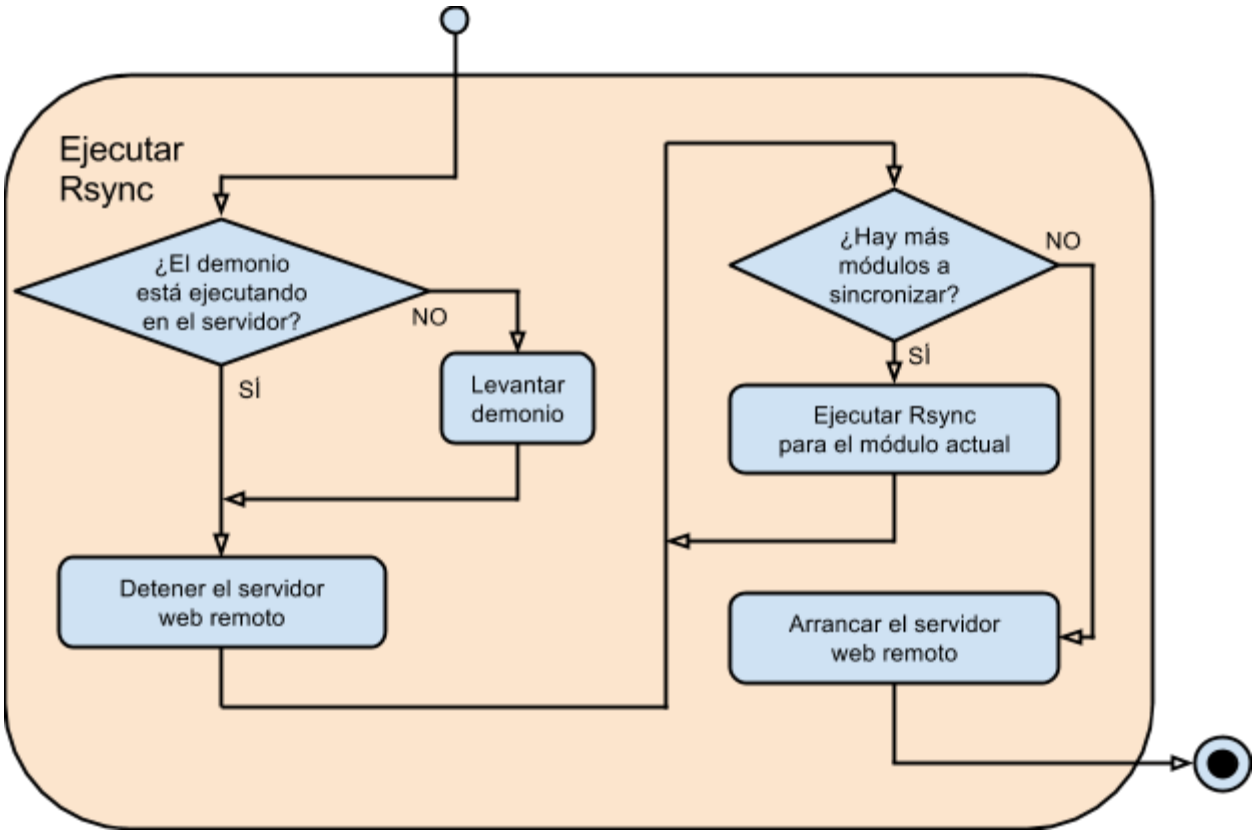

Figura 5.1: Diagrama de actividades del proceso diseñado para la sincronización de archivos.

Una vez ejecutado el proceso, este comienza chequeando que el demonio de Rsync se encuentre ejecutando en el servidor, y levantándolo en el caso contrario como fallback ante fallas. Luego se detiene el servicio de la aplicación web para evitar inconsistencias en su ejecución durante la sincronización de archivos. Paso seguido se ejecuta Rsync sobre la lista de archivos y directorios (módulos configurados), de modo que sean sincronizados incrementalmente contra cada uno de los módulos definidos en la configuración del demonio remoto. Para esto puede consultarse una lista local de configuraciones a fin de saber qué módulos sincronizar. En el caso implementado, esta lista está codificada en el archivo synclist.txt cuya especificación se muestra a continuación. Una vez que la sincronización termina, se vuelve a levantar el servidor web remoto, así puede ejecutar nuevamente con los cambios realizados en el código fuente (y demás archivos y librerías sincronizadas); y se dá por terminado el proceso.

El archivo synclist.txt como se mencionó, contiene la lista de directorios a sincronizar, junto con el nombre del módulo correspondiente de la siguiente manera (se muestra nuevamente a modo de ejemplo, la nomenclatura para los tres módulos anteriores, pero en realidad son más los módulos implementados):

```
"/cygdrive/c/datos/webserver/java/lib";lib
"/cygdrive/c/datos/webserver/java/classes";classes
"/cygdrive/c/datos/applications-config/queries";queries
```

Esta lista, puede editarla el desarrollador de software de acuerdo a sus necesidades, ya que pueden haber módulos que una vez desplegados, ya no necesite sincronizar en el futuro (módulos que sepa que no va a editar); o por el contrario, agregar nuevos módulos correspondientes a nuevos desarrollos realizados en la aplicación web (cada módulo nuevo debe tener también su contrapartida ingresada en el archivo remoto rsyncd.conf remoto).

5.3.3 - Cygwin como máquina virtual

El software de sincronización de archivos Rsync, presentando en la sección anterior, es un programa Unix. Esto significa que está diseñado para ejecutarse nativamente en sistemas operativos que implementan la API POSIX (ver capítulo 3.4), como Linux y Mac OS-X, por ejemplo. Ahora bien, el problema que se presenta particularmente en el caso de aplicación de este trabajo, es que los equipos (físicos, locales) de los desarrolladores de software, se basan en Windows, y este sistema operativo no implementa la API POSIX.
Afortunadamente, este no es un impedimento en la actualidad para ejecutar programas diseñados para Unix en Windows, y no es necesario descartar de la arquitectura de software implementada, la herramienta Rsync, que luego del análisis de posibles herramientas de sincronización hecho en 5.3.1, mostró ser la opción más oportuna.

Para casos como el que se presenta, existen dos opciones sencillas de solución:

1. Ejecutar alguna herramienta de virtualización (como VirtualBox o VMware) en el sistema operativo anfitrión, de modo de levantar una instancia de otro sistema operativo que sí implemente la API POSIX, y ejecutar Rsync sobre este sistema operativo virtual local.
2. Instalar las librerías necesarias en el sistema operativo, que brinden la implementación de la API POSIX, a fin de poder ejecutar los programas Unix en dicho sistema operativo.

Si bien ambas opciones son funcionales, la segunda es mucho más simple de implementar (una única instalación una sola vez, y luego los programas Unix ejecutan transparentemente de forma nativa). Además corre con la ventaja de ser más eficiente en cuanto a consumo de recursos de hardware, ya que levantar una máquina virtual con otro sistema operativo completo, siempre insume mucha más memoria y procesamiento de CPU, incluso en distribuciones de Linux pequeñas. Así, además de servir al fin buscado, al poseer las librerías

que transparentemente implementen toda la API POSIX, es posible usar cualquier herramienta Unix que el desarrollador desee, incluso aquellas que ya se usen productivamente en la aplicación web o en la arquitectura de la misma; achicando aún más la brecha entre las características del entorno de desarrollo o testeo usado, y el entorno de producción. Estas librerías por sí mismas, pueden considerarse una completa máquina virtual sobre la que ejecutar programas Unix; en el caso de este trabajo, se selecciona el software Cygwin para tal fin.

Cygwin es una colección de herramientas desarrollada originalmente por Cygnus Solutions (en la actualidad, el paquete de software está mantenido principalmente por trabajadores de Red Hat), para proporcionar un comportamiento similar a los sistemas Unix, pero bajo sistemas operativos Microsoft Windows. Su objetivo es portar software que ejecuta en sistemas POSIX a Windows, con una recompilación a partir de sus fuentes. Así, se torna en un entorno e interfaz de línea de comandos de tipo Unix, pero para Windows; proveyendo integración nativa de aplicaciones, datos y otros recursos del sistema basados en Windows, con aplicaciones, herramientas de software y datos de entornos Unix. Mientras que, sigue siendo posible lanzar aplicaciones Windows desde el entorno de Cygwin, así también como usar las aplicaciones y herramientas de Cygwin dentro del contexto operativo de Windows.

El sistema Cygwin consta de varias partes diferenciadas:

- Una biblioteca de enlace dinámico (formato de librerías de Windows) que implementa la interfaz de programación de aplicaciones POSIX, usando para ello llamadas a la API nativa de Windows.
- Una cadena de desarrollo GNU (que incluye el compilador GCC y el depurador GDB, entre otras utilidades), para facilitar las tareas básicas de desarrollo sobre Windows.
- Una colección extensiva de aplicaciones equivalentes a los programas más comunes de los sistemas UNIX, **entre los que se distribuye Rsync** nativamente en el paquete de instalación, así como otras utilidades de uso frecuente como SSH. Estos programas se recompilan desde sus fuentes originales, pero con las utilidades de desarrollo de software portadas que incluye Cygwin, de modo de poder comenzar a utilizarlas tan pronto como el paquete de Cygwin es instalado en el sistema Windows.

Así, para completar el stack de herramientas de software combinadas durante la fase de implementación de la arquitectura de desarrollo remoto mostrada, se instala Cygwin en el terminal de cada desarrollador de software. Esto no es una tarea tediosa, rutinaria ni que deba realizar cada desarrollador. El paquete de instalación puede ser completamente descargado por un administrador una única vez, y realizar una instalación desatendida a través de la red, sin necesitar de intervención alguna de los desarrolladores, ni de acceso físico a sus equipos.

5.4 - Desarrollo de la herramienta CASE

En el capítulo 3.4.2 de este informe, se introdujo la metodología para la expansión del IDE CASE Eclipse, a través de su arquitectura de plug-ins; y se explicaron los conceptos básicos de utilización de dicho framework, como ser las extensiones y los puntos de extensión. Aquí se explica ahora, cómo se implementa dicho framework para el desarrollo de la herramienta CASE de este trabajo (el broker, plug-in para la sincronización de recursos y otras tareas útiles dentro del IDE, para la interacción sencilla con la arquitectura de software desplegada). Antes de abocarse en qué puntos de extensión fueron implementados, cómo y para qué; se detalla el workflow que se decidió seguir, para la utilización de las herramientas de Eclipse PDE, durante el desarrollo del software en cuestión.

5.4.1 - Workflow de desarrollo de plug-ins de Eclipse

Para facilitar la comprensión del desarrollo del plug-in realizado (y del proceso de desarrollo de plug-ins para Eclipse en general), se explica y se sigue el workflow descripto en la siguiente figura [Aniszczyk08]. El mismo torna mucho más fácil el desarrollo de plug-ins, y más fácil aún el mantenimiento de los mismos, mediante la aplicación de buenas prácticas que se detallan a continuación.

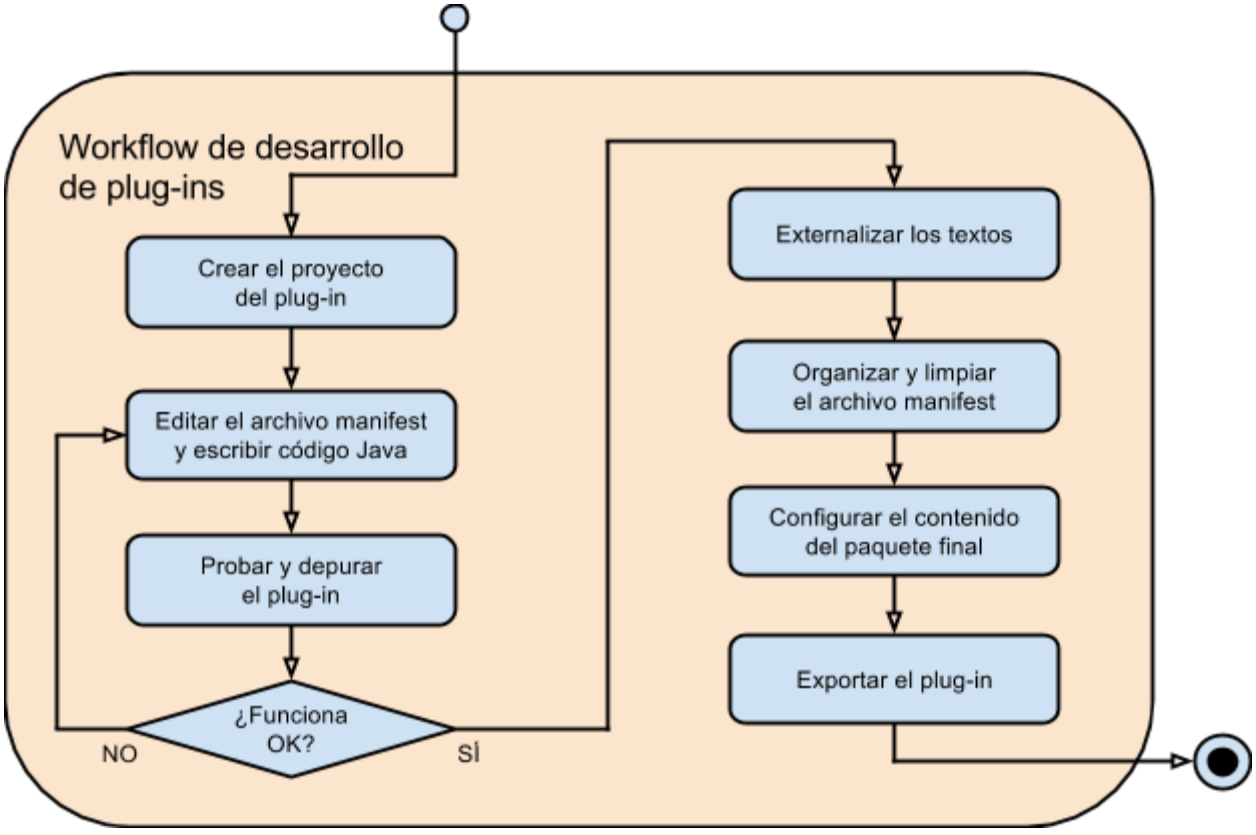

Figura 5.2: Workflow llevado a cabo para el desarrollo del plug-in broker en Eclipse PDE

La primera parte del workflow de desarrollo de plug-ins involucra la **creación misma del proyecto** en el IDE. Eclipse PDE posee un simple asistente visual para llevar a cabo esta tarea, el cual permite crear la estructura inicial de archivos y directorios, sin necesidad de escribirlos manualmente sino a través de las pantallas con controles gráficos de dicha herramienta. No se ahondará aquí en mayores detalles de dicho asistente, pero básicamente permite especificar el nombre y versión del plug-in, para qué plataforma se diseñará (en qué versión de la Máquina Virtual de Java ejecutará, por ejemplo entre otros atributos), qué dependencias posee y qué piezas se desean exportar; y generar automáticamente algunas plantillas iniciales para algunos de los artefactos que tendrá el plug-in a desarrollar.

Paso seguido, comienza la **edición del plug-in** en sí; esto es, editar las clases Java y demás archivos que componen el plug-in en cuestión. Puede realizarse todo el trabajo manualmente, o hacer uso de los asistentes que provee Eclipse PDE para cada tarea en particular. Algo importante en este punto es el archivo llamado “manifest” (manifiesto). Este archivo particular del proyecto incluye todos los metadatos que el framework requiere para entender qué hace el plug-in, y qué extensiones contribuye o a qué puntos de extensión se conecta (este archivo en particular es el que hace al plug-in *self-describing*, como se explicó en 3.4.2). Eclipse PDE posee también una interfaz visual para la edición de este archivo desde la cual se pueden lanzar directamente asistentes que guían al desarrollador en la tarea de extender puntos de extensión (a su vez de buscar cuál es el punto más idóneo para cada tarea), y generar plantillas listas para comenzar a escribir el código funcional del software, sin tener que lidiar con aspectos de implementación propios del framework de PDE (de todos modos se ofrece también un editor de texto plano para este archivo, con autocompletado inteligente de código). Más adelante en este capítulo se hará mención de qué puntos de extensión se implementaron desde esta herramienta y con qué propósitos.

Una vez que el plug-in comienza a tener la funcionalidad que se desea, el próximo paso del workflow son las **pruebas y depuración** del mismo. Para testear el desarrollo, Eclipse PDE implementa la noción de self-hosting (ver capítulo 3.4.2); esto significa que es posible lanzar una nueva ventana de Eclipse, la cual ya incluye instalado el plug-in en desarrollo dentro de su interfaz, sin la necesidad de exportar o desplegar nada aún. Además es posible lanzar dicha instancia en modo de depuración, y hacer uso de todas las herramientas de debugging usuales de Eclipse (en su ventana original) para terminar de revisar el código del plug-in.

Luego de sucesivos tests, prueba y error, y modificación o fixing del código del plug-in para lograr el cometido que se desee; comienza la fase de preparación del software para ser exportado en un paquete cómodamente distribuible.
Lo recomendable en este punto, es **externalizar todas las cadenas de texto** que se incluyen en la interfaz gráfica que pudiera ofrecer el plug-in, como así también en sus metadatos. Esto es deseable ya que un punto común e importante en el desarrollo de plug-ins es la internacionalización de los mismos (internacionalización, o más comúnmente su abreviación i18n, refiere al proceso de diseñar software de manera tal que pueda

adaptarse a diferentes idiomas y regiones, sin la necesidad de realizar cambios de ingeniería ni en el código fuente [Arevalillo04]). Cuando se llega al punto en que un plug-in es útil y será testeado y usado por múltiples actores, usualmente le llegan pedidos al desarrollador original para que lo traduzca a algún otro idioma del que se provee por defecto, o al menos asegurarse de que dicho plug-in es capaz de ejecutarse en algún idioma diferente. Por suerte, la cantidad de trabajo necesario para externalizar las cadenas de texto del plug-in que se haya desarrollado, realmente es mínima en el framework de PDE. El mismo posee un asistente que muestra automáticamente todas las cadenas identificadas que son relevantes a la externalización (infiere cuáles cadenas de texto son mostradas en la interfaz de usuario), y permite exportarlas a un archivo te texto plano que puede replicarse para otros idiomas y ser distribuido dentro del mismo plug-in. De hecho, contando con las cadenas externalizadas, cualquier persona podría extender el plug-in escribiendo el fragmento de software que sólo adapte dicho plug-in a un nuevo idioma, sin necesidad de conocer detalles de su implementación, ni de programación en absoluto.

Antes de exportar el plug-in, es recomendable **organizar el contenido del archivo manifest** que se comentó antes en esta sección, ya que tras sucesivos cambios que pudieran haber surgido durante la edición y pruebas del plug-in, es común que quede contenido irrelevante u optimizable en este descriptor del plug-in. PDE provee un conveniente asistente para esto, que presenta una variedad de opciones que pueden ser optimizadas automáticamente, como la organización y depuración de los paquetes que serán exportados, agregar las dependencias necesarias, remover contenido o entradas que actualmente no estuvieran en uso en el plug-in, etc.

Configurar el contenido del plug-in distribuible (build), es una paso importante en la aventura del desarrollo del mismo. Para esto se hace uso de otro asistente de PDE, que provee una simple interfaz de usuario para personalizar esta tarea y volcar el resultado en un archivo de texto de configuración, que contendrá el detalle de los archivos, librerías y demás artefactos que componen el build. Particularmente, se deben incluir el archivo manifest, demás archivos de configuración del plug-in generados en los pasos anteriores, los archivos de internacionalización que se hayan agregado y las librerías necesarias en el orden que sean requeridas para la correcta ejecución del software en otras instalaciones de Eclipse. En este punto es importante distinguir si se desea generar un build binario o de código fuente. El primero consta (además de los artefactos ya nombrados), de las clases Java compiladas (archivos con extensión .class) que componen el desarrollo del plug-in, de modo que pueda ser instalado y ejecutado directamente en el framework de Eclipse. Si se opta por un build de fuentes, se agregan las clases Java de texto plano originales (archivos con extensión .java), sin compilar. De este último modo, se distribuye el plug-in como una pieza de software libre, en el sentido de que quien lo obtenga puede inmiscuirse en el código fuente original, verlo, estudiarlo, modificarlo, y proponer mejoras si así lo decidiera. Obviamente, el mismo Eclipse compilará dichas clases desde los fuentes al instalar el plug-in distribuido de esta forma.
Para los fines del trabajo desarrollado aquí, se decidió optar por un build mixto, que incluya tanto los binarios como los fuentes, además de los scripts de línea de comandos necesarios

para la interacción del broker con la arquitectura de software local y remota desplegadas. El sentido de esto es acelerar los tiempos y tareas necesarias para la instalación en cada terminal (versiones binarias de las clases), a su vez que se reparta conocimiento en el equipo de desarrollo sobre la herramienta y se fomente a la expansión de la misma, o que se propongan mejoras o correcciones de ser necesario (versiones fuente de las clases).

El paso final en un workflow típico de desarrollo de plug-ins como el implementado, consiste en la **exportación del plug-in creado**. Eclipse PDE nuevamente facilita esta tarea con un asistente de exportación, integrado en el cuadro de diálogo usual de exportación de artefactos en general de Eclipse. Se puede seleccionar, además de la exportación del plug-in desarrollado, cualquier otro instalado que quisiera añadirse para ser distribuido junto con el actual, como así también las opciones de destino del plug-in (dónde guardarlo y con qué formato de destino), así como realizar un firmado digital del mismo (una opción interesante en términos de seguridad, pero que excede el alcance de este trabajo). Esto es todo lo que se requiere para obtener un plug-in desde el espacio de trabajo de Eclipse, a una forma consumible en el disco duro, que pueda ser distribuida al resto del equipo de desarrollo (o cualquier individuo en general).

5.4.2 - Puntos de extensión implementados

Ahora sí pasan a detallarse los puntos de extensión específicos implementados en la herramienta desarrollada. Para una comprensión sencilla, se muestran las extensiones en forma de diagrama de componentes. A su vez, se decide extender la especificación de UML, de forma de mostrar gráficamente qué componentes corresponden a los puntos de extensión provistos por Eclipse PDE (aquellos con la iconografía de tomacorriente o enchufe), y cuáles componentes corresponden a las extensiones desarrolladas (aquellos con el ícono de conector). Así puede apreciarse no sólo los componentes que forman parte del software dentro de la arquitectura de Eclipse, sino también cómo se conectan y cuáles componentes dan origen a cuáles otros.

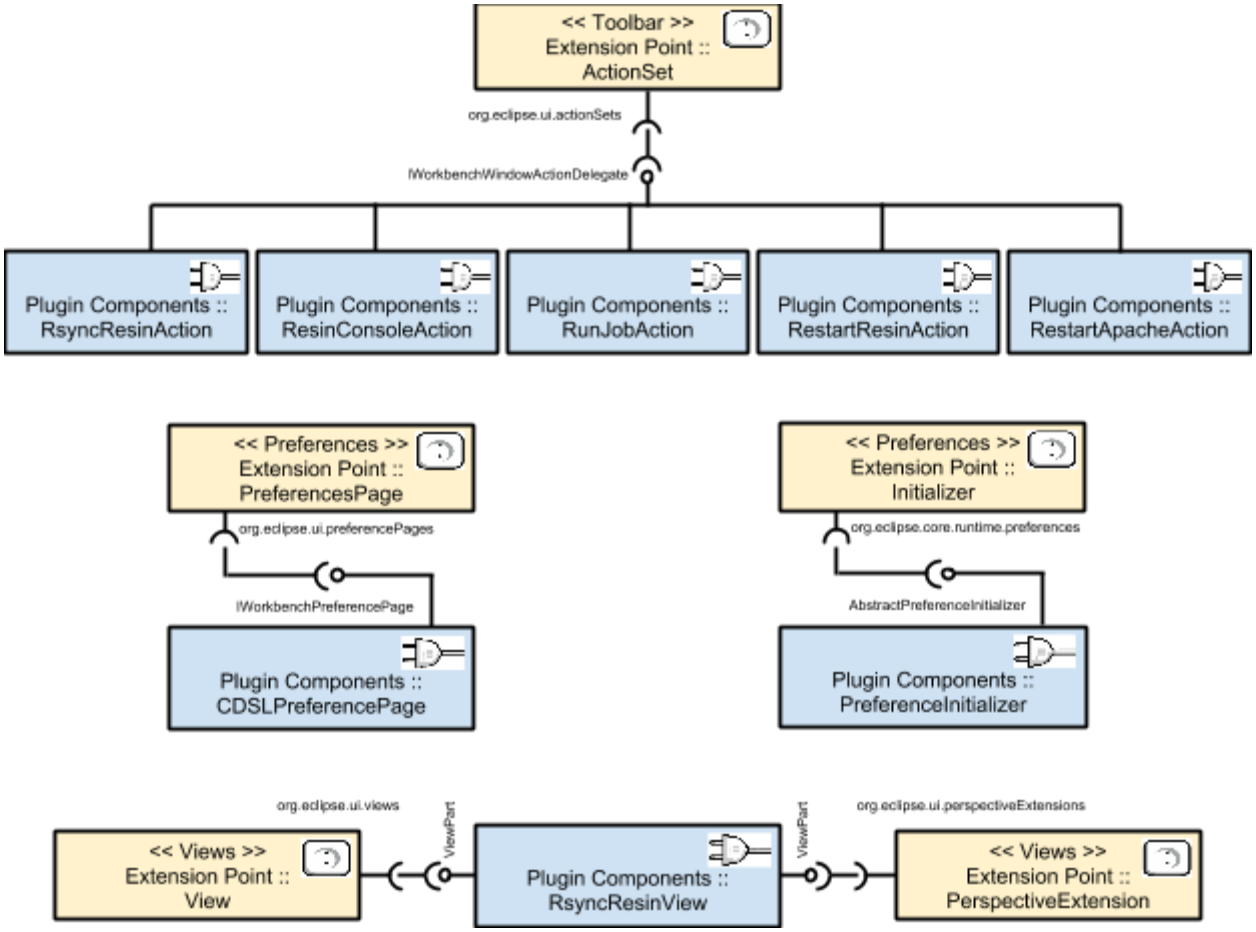

Figura 5.3: Diagrama de componentes de los puntos de extensión implementados y sus correspondientes extensiones.

Como se ve en el diagrama, se extienden cinco puntos de extensión del framework de Eclipse PDE (puede verse una implementación concreta con la sintaxis de PDE en el anexo 4). Cada una de estas extensiones se logran implementando las interfaces mostradas en el diagrama, propias del framework. Se detalla a qué sirve cada una en el contexto del broker desarrollado y el objetivo de la herramienta hecha:

- **org.eclipse.ui.actionSets**: usando este punto de extensión, un plug-in puede agregar menús, ítems de menús, y botones de la barra de herramientas a áreas comunes del espacio de trabajo de Eclipse. Estas contribuciones se conocen colectivamente como un “action set” y aparecen en la ventana de Eclipse según la configuración de perspectiva usada por el usuario [EDocExtPoints]. La herramienta desarrollada contribuye cinco botones a la barra de herramientas de Eclipse, para la ejecución de las acciones correspondientes a las funcionalidades provistas por la misma. Así, es posible lanzar simplemente presionando el botón adecuado de la barra de herramientas, la sincronización de archivos contra la máquina virtual remota, obtener una ventana local del log del servidor web remoto, reiniciar servicios o lanzar jobs en

modo de debug. Una vez lanzada la acción, el broker ejecutará el proceso adecuado que recolecte los parámetros correspondientes, muestre las opciones o información adecuadas, y desemboque en la ejecución del script de línea de comandos preciso para la tarea en cuestión.

- **org.eclipse.ui.preferencePages**: la plataforma de Eclipse PDE provee un cuadro de diálogo común para las preferencias de usuario. El propósito de este punto de extensión, es permitir a los plug-ins agregar páginas a dicho cuadro de diálogo cuando el mismo sea abierto [EDocExtPoints]. A los fines de la herramienta desarrollada, se usa para presentar al desarrollador en una interfaz común, las opciones que requiere el plug-in para su correcta configuración; a saber: la dirección IP de la máquina virtual otorgada y el nombre de usuario con el que loguearse a la misma. Además, para evitar diálogos reiterativos y posiblemente molestos, se presentan aquí, opciones para no mostrar confirmaciones de las acciones expuestas en la extensión anterior. Por último se agrega un switch para permitir seleccionar si se desea aplicar el uso de la arquitectura desplegada, o espontáneamente se desea volver a interactuar con el entorno de desarrollo completamente local (útil si se produjera un corte en la conexión de red que imposibilite trabajar con la instancia remota).
- **org.eclipse.core.runtime.preferences**: el punto de extensión de preferencias permite a los plug-ins agregar nuevos ámbitos de configuraciones al mecanismo de opciones de Eclipse; como así también, especificar una clase a ejecutar para inicializar los valores por defecto de estas configuraciones en tiempo de ejecución [EDocExtPoints]. Se implementa en el plug-in desarrollado para inicializar los valores de las opciones presentadas en la extensión del punto de extensión anterior (panel de preferencias del plug-in); de modo de especificar por defecto que el login sea realizado con el usuario root (pero que desde dicho panel puede editarse), o que en un principio se mostrarán al desarrollador todos los cuadros de diálogo de alerta antes de accionar contra la máquina virtual remota.
- **org.eclipse.ui.views**: este punto de extensión se usa para definir nuevos componentes visuales (vistas de usuario) para el espacio de trabajo de Eclipse. Usualmente se utilizan para navegar una jerarquía de información, abrir un editor de archivos, o mostrar propiedades para el editor activo [EDocExtPoints]. El broker desarrollado implementa extensiones en la forma de vistas para mostrar mensajes (cuadros de diálogo) al desarrollador con feedback acerca del resultado de las operaciones solicitadas. A su vez, se implementa una nueva vista dentro de la interfaz de Eclipse, donde puede verse el detalle de cada módulo y archivo sincronizado a la máquina virtual remota, luego de que el desarrollador solicite ejecutar el proceso en cuestión. Para la ejecución de jobs remotos, se implementa otra vista que recolecta algunas opciones adicionales para la ejecución del job, como parámetros opcionales y si se desea ejecutar en modo de debugging.
- **org.eclipse.ui.perspectiveExtensions**: una perspectiva define los contenidos iniciales de las barras de acción de la ventana (menú y barra de herramientas), y el conjunto inicial de vistas y su correspondiente disposición dentro del espacio de

trabajo de Eclipse. Este punto de extensión es usado entonces, para extender perspectivas registradas por otros plug-ins, que pueden contribuir acciones o vistas a la perspectiva que aparece cuando el desarrollador selecciona una en cuestión [EDocExtPoints]. El broker extiende este punto de extensión, para disponer una de las vistas implementadas mediante el punto de extensión anterior (la que informa el detalle de archivos sincronizados), dentro de la perspectiva de desarrollo para la plataforma Java, y próxima al panel de lista de tareas de dicha perspectiva.

6 - Conclusiones y trabajo futuro

La herramienta lograda durante el desarrollo de este trabajo, no sólo supo cumplir con la meta principal de lograr un entorno de desarrollo de software más veloz bajo el panorama descripto (se aportan las mediciones de la siguiente sección junto con las de los anexos 1 y 2, además de testimonios de usuarios en el capítulo 6.2); sino que a su vez, mostró cómo combinando creativamente varias piezas de software distintas (incluso de sistemas operativos distintos o arquitecturas distintas), se puede lograr no sólo la optimización o características provistas por cada una de estas piezas, sino un sistema completo mucho más novedoso, actualizado y ameno para sus usuarios.
Durante el proceso, se investigaron, implementaron y adquirieron conocimientos adicionales sobre tópicos tan interesantes del mundo de la computación, como virtualización de software, sincronización de archivos, criptografía, arquitecturas de hardware y más; lo que hizo que este trabajo permita no sólo dar por cerrada una etapa académica personal, sino también lograr un crecimiento profesional enorme en cuanto a conocimientos y aptitudes adquiridas.
Para dar por finalizado entonces el trabajo, es que en las siguientes secciones de este capítulo se exponen comparativas de tiempos de desarrollo, en un ambiente que implementa lo desarrollado en este trabajo; junto con comparativas de optimización de hardware o de recursos en general. Se exponen testimonios de primera mano de aquellos que pudieron tener la oportunidad de experimentar un entorno de desarrollo de software del mundo real, de una aplicación altamente demandante; y cómo la aplicación de las tecnologías de este trabajo, y fundamentalmente la herramienta desarrollada, logró mejorar su experiencia de desarrollo de software en varios aspectos, y por ende, su productividad y satisfacción personal.
Por último, se detallan cuáles son las líneas de investigación que pueden continuar lo desarrollado en este trabajo, qué pueden aportarle al desarrollo dichas líneas de investigación; y hacia dónde se pretende llegar mediante la aplicación u optimización de la herramienta desarrollada.

6.1 - Mediciones

Al utilizar el ambiente de desarrollo de software mejorado con las adaptaciones en la arquitectura mostradas en este trabajo, para el caso de aplicación específico de MercadoLibre, es bastante obvia la mejora en la velocidad de ejecución. No sólo la primer impresión, a medida que se le dá más uso y se navegan más secciones del sitio web en desarrollo, mayor es la ganancia y los tiempos de respuesta.

Aún así, es posible realizar un análisis un poco más detallado sin caer en gran cantidad de números que sí se aportan en los anexos antes mencionados. Partiendo de la suposición de

que se puede hacer una comparativa tomando como caso base o caso de estudio, un acceso común a una sección del sitio que requiera de un usuario autenticado en el sistema (se suponen 5 accesos a la sesión del usuario durante dicha navegación), y que se realicen una cantidad prudente de consultas a la base de datos (para el ejemplo ahora medido, 10). Entonces se deben comparar los tiempos promedios de cada acción en cada ambiente para la sumatoria total, considerando las mediciones de los anexos 1 y 2 respectivamente, para el caso de la configuración de threads más coherente (aquella mostrada en detalle al final del anexo 1, con 10 threads ejecutando concurrentemente). No se debe olvidar que para el nuevo ambiente, es necesario considerar además el tiempo que demora realizar el proceso extra de sincronización de archivos.

Entonces, viendo en comparativa para el caso especificado:

Entorno local original, con lecturas remotas		**Nuevo entorno remoto, con lecturas locales**	
Acción	**Tiempo (en millis)**	**Acción**	**Tiempo (en millis)**
-	-	Sincronización de archivos	5000 (aprox.)
Consulta a BD remota (10 lecturas)	30000	Consulta a BD local (10 lecturas)	10100
Acceso a sesión remota (5 lecturas)	8105	Acceso a sesión local (5 lecturas)	1195
TOTAL	**38105**	**TOTAL**	**16295**

Tabla 6.1: Comparación de tiempos de acceso al aplicativo web, antes y después de las mejoras realizadas, para un caso de uso típico.

Como se ve en estas mediciones, se logra un speed-up de 2,3x para un acceso típico al sistema (speed-up refiere a la métrica de mejora de performance relativa entre las dos tareas). Navegando mayores contenidos, haciendo consultas más costosas a la base de datos y realizando debugging del software; la realidad muestra que la ganancia en la velocidad de acceso es mucho mayor aún.

Abocándose ahora en las mejoras en la infraestructura de hardware de desarrollo, logradas en el caso de aplicación de la arquitectura desplegada en MercadoLibre; se logró ahorrar un 75% en consumo energético con un datacenter de desarrollo consolidado, virtualizado y escalable.

Resumiendo los desafíos a los que se expone la mejora descripta en la virtualización de la

arquitectura de hardware:

- Consolidar y validar la infraestructura tecnológica, altamente demandante de servidores; para reducir el tamaño del datacenter de desarrollo y acompañar el crecimiento en el negocio.
- Alcanzar alta disponibilidad y escalabilidad en los servidores y aplicaciones requeridas para el desarrollo de software; para simplificar la administración de la infraestructura tecnológica, reducir costos operacionales y soportar el constante aumento de desarrolladores.
- Reducir el consumo de electricidad en el datacenter de desarrollo, para ahorrar y, al mismo tiempo, cumplir con la política de sustentabilidad del medio ambiente de la empresa.

Sobre la base de estos desafíos, es que se aplicaron las siguientes soluciones con los siguientes ahorros o beneficios [OraCust11]:

- Se implementó el software de virtualización de servidores Oracle VM, Oracle Management Pack para Linux y Oracle Clusterware para Oracle Linux; para soportar el gran volumen de crecimiento de la plataforma de desarrollo de software.
- Se consolidó y escaló la infraestructura del datacenter de desarrollo de la compañía, para soportar el crecimiento en el negocio y por consiguiente, en las necesidades de desarrollo de software.
- Se implementaron máquinas virtuales, reduciendo 75% el consumo eléctrico en el datacenter de desarrollo y reduciendo la huella de carbono.
- Se alcanzó una reducción de costos para infraestructura, servidores y electricidad por la virtualización de servidores con Oracle VM dentro de la estrategia de sustentabilidad de la compañía, ahorrando un 75% en la inversión en equipamiento sin interrumpir las tareas de desarrollo.
- Se disminuyó un 80% el espacio que ocupa la infraestructura en el datacenter de desarrollo, pasando de 50 servidores a tan sólo 10 con alta eficiencia.
- Se simplificó el manejo de la infraestructura con alta disponibilidad con la capacidad de Oracle VM, que le permite migrar fácilmente de un servidor a otro sin enfriamiento previo, con cero interrupción en las tareas de desarrollo.
- Se replicaron las configuraciones estándar de software con Oracle VM Templates, reduciendo costos operativos y acelerando el tiempo de llegada con un ambiente de desarrollo totalmente funcional, para los nuevos desarrolladores que se incorporan al equipo.

Según Rodrigo Benzaquén, Director de Infraestructura de MercadoLibre: “MercadoLibre es una compañía que quiere ser sustentable en el largo plazo en todos los frentes, incluido el medio ambiente. Con Oracle VM, nosotros virtualizamos nuestra infraestructura de desarrollo en un 100%. Tenemos menos equipos para enfriar y mejoramos 75% la eficiencia energética. Además, redujimos en un 75% el espacio necesario en el datacenter logrando la meta de hacer negocios y cuidar el ambiente.”

6.2 - Experiencias de uso

Para relevar la experiencia en el uso de la arquitectura, y tomando específicamente el caso de aplicación de MercadoLibre de la arquitectura de desarrollo de software remoto desarrollada e implementada; se entrevistó a algunos usuarios para recibir algo de feedback de la herramienta. Todos estos usuarios son ingenieros de software que tuvieron experiencia concreta en el uso del ambiente de desarrollo de software anterior a las mejoras aplicadas (el panorama descripto en el capítulo 1.1), y posterior experiencia en el uso del ambiente mejorado (con la aplicación de la arquitectura de software de este trabajo, metodología de sincronización de recursos y uso del plugin desarrollado para la herramienta CASE).
Es así que se les consultó específicamente sobre cómo les resultó el uso de la herramienta, en qué mejoró su labor diaria, qué les hubiera gustado agregar o modificar al ambiente, y cualquier otro comentario o crítica en general al respecto (en fin, cómo fue su experiencia con el uso de la herramienta).

En general, el feedback recolectado muestra que la principal ventaja que percibieron los usuarios sobre el uso de la herramienta, es el incremento en la velocidad de desarrollo. Esto es algo muy bueno, ya que era uno de los objetivos principales de la herramienta y de todo el caso de estudio; además de que permite a los desarrolladores y al resto del equipo de IT, adoptar más fácilmente las metodologías de desarrollo ágiles de software.
También se destaca entre los testimonios el hecho de poseer un ambiente de desarrollo más símil al de producción. Esto favoreció a los desarrolladores en lograr desarrollos que fueran efectivos desde etapas tempranas del ciclo de desarrollo de software, y que la mayoría de las fallas introducidas en los nuevos desarrollos, relativas a configuraciones de ambiente, no se pusieran en evidencia en los entornos productivos, sino antes en sus propias máquinas.
Por último, vale destacar los comentarios acerca de que la mayoría experimentó una fácil transición desde el entorno de desarrollo de software anterior hacia el nuevo, y que el plugin desarrollado para la herramienta CASE les permitió administrar dicho ambiente de una forma fácil y cómoda.
Para obtener los comentarios explícitos y detallados de estas entrevistas realizadas, se puede referir al anexo 6.

6.3 - Reconocimientos

La implementación realizada de este trabajo, en el caso de uso puntual de MercadoLibre (la mayor plataforma de compras y ventas por internet de Latinoamérica), fue concebida dentro de una iniciativa aún mayor de virtualización de infraestructura de hardware. Básicamente no sólo se aplicó la metodología diseñada en este trabajo de investigación, para atacar el segmento de desarrollo de software de la compañía, sino que también se aplicaron conceptos similares e implementación de estas tecnologías de virtualización en otras áreas,

como ser infraestructura destinada a usuarios finales del sitio web (usuarios compradores y vendedores de MercadoLibre).
Esta implementación en su conjunto, tanto la parte abocada a desarrollo expuesta en todo este trabajo, de la cual participó activamente quien suscribe; como así también el segmento destinado al resto de la virtualización del hardware de infraestructura, permitieron a la compañía ser reconocida con el premio Oracle a la Innovación 2009. Este premio reconoce a aquellas empresas que han sabido implementar de forma innovadora los productos de la multinacional Oracle.

Durante la conferencia anual “Oracle Open World 2009” se dieron a conocer las empresas a nivel mundial ganadoras del premio “Oracle Innovation Awards”. En el caso de MercadoLibre, el logro obtenido corresponde al segmento “Management e Infraestructura”, una de las cuatro categorías que conforman el listado, por el exitoso proceso de virtualización de su plataforma tecnológica.
En dicha edición del Oracle Open World, que se llevó a cabo en la ciudad de San Francisco en Estados Unidos, MercadoLibre fue la única compañía argentina seleccionada para este galardón. Fue reconocida por haber implementado los productos Oracle VM y Oracle Unbreakable Linux (ver capítulos 3.2 y 5.1), permitiendo a la compañía reducir costos, espacio y consumo de energía, acelerando al mismo tiempo la implementación de software; y manteniendo así, una plataforma robusta orientada a seguir acompañando el crecimiento del negocio.

Según Ramiro Cormenzana, Director Corporativo de Infraestructura y Bases de Datos de MercadoLibre: “este reconocimiento es un orgullo para MercadoLibre y en especial para el equipo de Infraestructura, que pudo transformar esta iniciativa en una respuesta concreta y efectiva a las necesidades que teníamos en términos de infraestructura tecnológica”.

6.4 - Trabajo y visión de futuro

Si bien la arquitectura de software junto con la herramienta CASE desarrolladas mostraron ser efectivas a los fines buscados, aún puede seguir mejorándose para alcanzar un nivel de productividad cada vez mayor, y que pueda ser utilizada sin complicaciones por un público cada vez más grande y diverso también. Hacia donde se desea llegar, es a una herramienta en la que no haga falta bajo ningún caso, realizar acciones mediante línea de comandos, o editar configuraciones en archivos de texto plano. En definitiva, que la ejecución de la máquina virtual remota sea aún más transparente para el desarrollador, logrando una experiencia de usuario aún más enriquecida. Esto puede alcanzarse desarrollando más piezas de software de tipo middleware (software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, redes, hardware y/o sistemas operativos; a su vez que simplifica el trabajo de los desarrolladores de software en la compleja tarea de generar las conexiones y sincronizaciones que son necesarias en los sistemas distribuidos [Karne03]).

Tomando como nuevo punto de partida la premisa anterior, se puede continuar mejorando la herramienta desarrollada, agregando algunas características adicionales, bien en el broker CASE, o bien en la máquina virtual remota:

- Por lo general, los archivos de mayor tamaño en los repositorios de código fuente se corresponden con librerías (en el caso de Java, archivos con extensión .jar). Además, estos archivos suelen sufrir muy pocas modificaciones en el tiempo. Algunas librerías son añadidas cuando se requiere de alguna característica extra; o bien se actualizan cuando el desarrollador original pone a disposición una nueva versión, con características nuevas requeridas por el proyecto que la usa.
  Entendiendo este caso particular de artefactos de software, y posiblemente encontrando nuevos casos similares, pueden alojarse las librerías necesarias para la aplicación en un directorio compartido a todos los desarrolladores y máquinas virtuales, de forma que esta unidad se monte automáticamente en la máquina virtual usada, y la aplicación web pueda leer las librerías desde allí, sin la necesidad de que sean sincronizadas por los desarrolladores de software desde sus equipos.
- En cuanto a la configuración manual de la herramienta, un aspecto mejorable es la lista de archivos/directorios a sincronizar con la máquina virtual. Esta configuración, como se vió en el capítulo 5.3.2, es mantenida en dos archivos de texto plano, uno local y otro remoto. Cada vez que se requiere agregar o quitar colecciones de archivos a sincronizar (o directorios), es necesario editar estos archivos de texto.
  Una opción más cómoda sería desarrollar un módulo adicional sobre el broker CASE, que presente una interfaz gráfica de usuario para tal fin, a modo de poder incluir o remover los directorios de la configuración de forma amena, editando ambos archivos automáticamente y reiniciando el demonio remoto de Rsync para que tome los cambios realizados.
  Si bien desde un primer punto de vista, esta parece ser es una característica interesante, el uso de la herramienta mostró que en la gran mayoría de los casos, los desarrolladores ajustan esta configuración una sola vez cuando comienzan a utilizar el nuevo entorno de desarrollo; y luego ya no hacen cambios en la misma. Por lo que el desarrollo de esta característica en el plugin, si bien es un agregado útil, finalmente no fue realizado durante el desarrollo de este trabajo.

Además, no se deben ignorar los deseos de los usuarios, por lo que de entre los testimonios obtenidos (ver capítulo 6.2), se pueden recolectar ideas útiles para continuar mejorando la herramienta:

- Un manual de uso del entorno, y particularmente de la herramienta CASE, es un agregado de valor que puede entregarse junto con el software. Este manual debe brindar soporte sobre los problemas más comunes que haya presentado el entorno para con los desarrolladores, y guiarlos en la solución a aplicar en cada uno de los mismos.
- Probado el objetivo de lograr un ambiente de desarrollo transparente, por sobre arquitecturas heterogéneas, el próximo paso obvio y ambicioso es el de soportar la

mayor cantidad posible de arquitecturas de hardware y software (para así dar soporte a la mayor cantidad posible de usuarios). Para el caso de aplicación de este trabajo, se desarrolló el plugin de Eclipse para ser ejecutado bajo máquinas clientes basadas en Windows, y máquinas virtuales remotas basadas en Linux, siguiendo la investigación realizada. Si bien el plugin de Eclipse está escrito en Java y es portable entre arquitecturas distintas (así como el mismo Eclipse), los scripts de línea de comandos desarrollados para la interacción con el ambiente remoto, son dependientes de la plataforma local (usan instrucciones de la API del sistema operativo local; para el caso de aplicación, Windows). Para continuar mejorando el ambiente y ampliando su adopción, pueden desarrollarse los scripts equivalentes para que el plugin pueda ser ejecutado también en máquinas locales basadas en Linux y Mac OS-X (adaptar los scripts para que usen la API de POSIX/Unix bajo estos sistemas operativos).

En cuanto a las líneas de investigación actuales que pretenden continuar modernizando, automatizando y haciendo que el desarrollo remoto de software llegue a cada vez más individuos, se puede hacer referencia a la evolución natural de las máquinas virtuales preconfiguradas. Como se detalló en el capítulo 5.1, estas máquinas proveen un enfoque innovador para el despliegue de configuraciones completas de software, mediante la aplicación de imágenes de disco preinstaladas (para el caso de aplicación de este trabajo se utilizó el producto de esta categoría “Oracle VM Templates”, como se mostró en el mismo capítulo). Aún así, luego de la adquisición de la imagen de disco de máquina virtual correcta para las necesidades que se posean, sigue siendo necesario instalar la misma en la arquitectura de hardware y software usada en cada entorno de desarrollo. Es decir, se automatiza y desliga de la instalación del software, pero no de la administración de los equipos en donde estas imágenes ejecutan.

Todo desarrollador de software necesita de infraestructura para diseñar, desarrollar, probar y entregar aplicaciones personalizadas. El siguiente paso entonces, y hacia donde se dirigen todos los esfuerzos de investigación de la actualidad, es llevar esta metodología de creación de ambientes de desarrollo y testeo, completamente a la nube. Uno de los actuales y principales jugadores en este segmento, es Microsoft con su tecnología “Azure”, que le ofrece todo lo necesario al desarrollador de software para provisionarse y administrar todo un entorno de desarrollo y testeo, completamente en la nube. Este producto permite construir dichos ambientes de desarrollo de aplicaciones bajo demanda, para Windows, Linux o cualquier otro stack de tecnología, en tan sólo minutos.
Así se logra aún más agilidad, obteniendo el equipamiento de cómputo, almacenamiento y red exactos que se requieren para desarrollar en minutos, no horas ni días. Se mejora la calidad, a la vez que se realizan entregas de software desarrollado y testeado en ambientes iguales a los productivos. Se ahorra tiempo y dinero, al automatizar y desplegar la arquitectura necesaria, que será la misma tanto para desarrollo como para pruebas. [Azure14]

A1 - Acceso a archivos de sesión en disco

Un proceso que se realiza continuamente al cambiar de sección en el sitio web de la aplicación tomada como marco de referencia de este trabajo, o al ingresar o seleccionar un administrador (herramientas internas de backoffice), es la lectura del archivo de sesión del usuario. En el entorno de desarrollo especificado, este archivo se almacena en una unidad de red ubicada en Buenos Aires (unidad N: a partir de aquí), causando lentitud en su acceso reiterado desde la oficina de San Luis.
Una posible mejora planteada sería almacenar los datos de sesión del usuario en el disco C: (unidad local a la estación de trabajo de cada desarrollador), o quizás en una unidad de red local a San Luis (unidad I: en este caso, ya instalada), para la oficina en cuestión.

Se escribió un programa Java para realizar un benchmark (técnica utilizada para medir el rendimiento de un sistema o componente del mismo) de los tiempos de acceso a los archivos de sesión de usuario, en N: y en C: (no se tomaron los tiempos de acceso a I:).
El mismo lanza X threads concurrentes, los cuales realizan Y accesos a la sesión del usuario de forma repetitiva. Cada acceso consta de la lectura del archivo y del parseo del mismo (exactamente el mismo que realiza la aplicación para obtener los datos que requiere), a una sesión de usuario determinada predefinida (hardcodeada).
Se lanzó el test con distintas configuraciones en cantidad de threads y en cantidad de accesos por cada thread al archivo de sesión. Se obtienen los siguientes listados con el tiempo promedio de acceso por thread, y el tiempo promedio general.

A1.1 - Resumen de mediciones

Medición en el entorno de desarrollo original, lectura al disco N: sin alteraciones en la arquitectura ni conectividad.

Unidad de disco	Cantidad de threads	Cantidad de accesos por thread	Tiempo (promedio en milisegundos)
N	10	100	**1470**
N	10	10	**1621 (*)**
N	20	10	**3092**
N	30	10	**4221**
N	40	10	**5673**

N	50	10	**6804**

*: más adelante en este anexo, se ofrece un análisis detallado de esta corrida del test, y comparativo con el test análogo accediendo al disco C:

Tabla A1.1: Tiempos de acceso a la sesión del usuario en el entorno de desarrollo original.

Medición desde el switch principal

Unidad de disco	Cantidad de threads	Cantidad de accesos por thread	Tiempo (promedio en milisegundos)
N	10	100	**1317**
N	10	10	**1979**
N	20	10	**3482**
N	30	10	**5392**
N	40	10	**6470**
N	50	10	**7685**

Tabla A1.2: Tiempos de acceso a la sesión del usuario desde el switch principal de la conexión de red.

En mucho accesos (100) no se aprecia una ganancia significativa, y para colmo al aumentar la concurrencia, los tiempos se tornan levemente peores que con la conexión original.

Medición desde una conexión punto a punto a Buenos Aires

Unidad de disco	Cantidad de threads	Cantidad de accesos por thread	Tiempo (promedio en milisegundos)
N	10	100	**1317**
N	10	10	**1520**
N	20	10	**2796**
N	30	10	**3941**
N	40	10	**5033**

N	50	10	**6256**

Tabla A1.3: Tiempos de acceso a la sesión del usuario desde una conexión punto a punto alternativa.

Los tiempos aquí sí son levemente mejores de forma casi uniforme, sin embargo la ganancia sigue resultando escasa y no justifica el cambio de migrar el tráfico para acceder a las sesiones de esta forma.

Medición desde una VPN* a Buenos Aires, provista por Speedy

Unidad de disco	Cantidad de threads	Cantidad de accesos por thread	Tiempo (promedio en milisegundos)
N	10	100	**4145**
N	10	10	**3828**
N	20	10	**7620**
N	30	10	**10705**
N	40	10	**13182**

Tabla A1.4: Tiempos de acceso a la sesión del usuario desde una conexión VPN.

Los tiempos de acceso resultan mucho peores, sin mencionar que dependen además totalmente de la performance de la conexión provista por un ISP (Internet Service Provider, Proveedor de Servicios de Internet) en particular (Speedy). Si el centro de desarrollo de San Luis tuviera que cambiar de ISP por cualquier razón, los tiempos de acceso podrían fluctuar sin valores predecibles.

*: una VPN o Virtual Private Network (red privada virtual), es un protocolo de red que reemplaza el tendido u otro soporte físico en una red con una capa abstracta, permitiendo crear una red a través de internet.

Medición con lecturas al disco local C:

Unidad de disco	Cantidad de threads	Cantidad de accesos por thread	Tiempo (promedio en milisegundos)
C	10	100	**30**

C	10	10	**239 (*)**
C	20	10	**292**
C	30	10	**340**
C	40	10	**525**
C	50	10	**776**

*: más adelante en este anexo, se ofrece un análisis detallado de esta corrida del test, y comparativo con el test análogo accediendo al disco N:

Tabla A1.5: Tiempos de acceso a la sesión del usuario desde el disco rígido local.

En este caso la mejora de performance es muy grande respecto de la primera medición (entre 7x y 49x, según la configuración de threads y accesos). Esto se debe a que ya no se depende de la distancia a la cual se encuentra el archivo físico, ni de la calidad y congestión de la red a través de la cual se accede al mismo.

A1.2 - Detalle de mediciones

Aquí se muestra en detalle cada acceso de cada thread, en el mismo orden en que finalizan los accesos concurrentemente. Por el tamaño de los datos recolectados, se muestra el detalle para un caso significativo, testeado en cada una de las unidades (en el mejor y más representativo caso posible para la unidad N:).

Unidad N:, 10 threads, 10 accesos por thread

Número de thread	Número de acceso	Tiempo en milisegundos
T7	L0	3552
T5	L0	3742
T0	L0	3771
T6	L0	4031
T4	L0	4032
T8	L0	4095
T9	L0	4094
T2	L0	4126
T1	L0	4131
T3	L0	4452

T7	L1	1191
T5	L1	1268
T0	L1	1422
T4	L1	1189
T6	L1	1278
T1	L1	1235
T2	L1	1239
T9	L1	1325
T8	L1	1358
T3	L1	1202
T7	L2	1022
T5	L2	1269
T4	L2	1350
T0	L2	1407
T8	L2	1175
T6	L2	1319
T2	L2	1319
T1	L2	1376
T9	L2	1378
T3	L2	1143
T7	L3	1145
T0	L3	1017
T5	L3	1479
T4	L3	1190
T8	L3	1305
T6	L3	1389
T2	L3	1361
T1	L3	1305
T3	L3	1248
T9	L3	1313
T7	L4	1286
T0	L4	1341
T5	L4	1383
T4	L4	1494
T8	L4	1514
T6	L4	1574
T1	L4	1692

T2	L4	1693
T3	L4	1774
T9	L4	1709
T7	L5	1718
T0	L5	1205
T5	L5	1130
T4	L5	1149
T8	L5	1229
T6	L5	1259
T1	L5	1309
T2	L5	1310
T9	L5	1379
T3	L5	1379
T7	L6	1339
T0	L6	1289
T5	L6	1264
T4	L6	1313
T8	L6	1403
T6	L6	1344
T2	L6	1422
T1	L6	1507
T3	L6	1438
T9	L6	1439
T7	L7	1526
T0	L7	1358
T5	L7	1525
T4	L7	1399
T8	L7	1387
T6	L7	1333
T2	L7	1294
T1	L7	1388
T3	L7	1497
T9	L7	1582
T7	L8	1521
T0	L8	1520
T5	L8	1504
T4	L8	1505

T8	L8	1347
T6	L8	1552
T2	L8	1482
T1	L8	1427
T3	L8	1618
T9	L8	1563
T7	L9	1605
T0	L9	1604
T4	L9	1521
T5	L9	1605
T8	L9	1382
T6	L9	1205
T2	L9	1121
T3	L9	641
T1	L9	1026
T9	L9	639

Tabla A1.6: Análisis exhaustivo de los tiempos de acceso a la sesión del usuario en el entorno de desarrollo original; para un caso de uso representativo en cantidad de threads y lecturas por thread.

[T0] Promedio = 1593 milisegundos
[T1] Promedio = 1639 milisegundos
[T2] Promedio = 1636 milisegundos
[T3] Promedio = 1639 milisegundos
[T4] Promedio = 1614 milisegundos
[T5] Promedio = 1616 milisegundos
[T6] Promedio = 1628 milisegundos
[T7] Promedio = 1590 milisegundos
[T8] Promedio = 1619 milisegundos
[T9] Promedio = 1642 milisegundos

El promedio general del tiempo de acceso es de 1621 milisegundos. Un detalle que puede observarse al comparar la primer lectura de cada thread ([L0]) con respecto a las demás, es que bajo esta arquitectura, los primeros accesos de cada thread son considerablemente más costosos que los accesos subsiguientes.

Unidad C:, 10 threads, 10 accesos por thread

Número de thread	Número de acceso	Tiempo en milisegundos
T0	L0	2233
T0	L1	1
T0	L2	1
T0	L3	3
T0	L4	4
T0	L5	1
T0	L6	1
T0	L7	1
T0	L8	1
T0	L9	1
T1	L0	2287
T1	L1	1
T1	L2	3
T2	L0	2293
T2	L1	1
T2	L2	1
T2	L3	1
T2	L4	1
T2	L5	1
T2	L6	0
T2	L7	0
T2	L8	1
T2	L9	1
T1	L3	12
T1	L4	1
T1	L5	2
T1	L6	1
T1	L7	1
T1	L8	1
T1	L9	1
T9	L0	2346
T3	L0	2349
T3	L1	36
T3	L2	0
T3	L3	1
T3	L4	1

T3	L5	4
T3	L6	1
T3	L7	1
T3	L8	1
T3	L9	2
T9	L1	51
T9	L2	1
T9	L3	0
T9	L4	1
T9	L5	1
T8	L0	2406
T9	L6	7
T8	L1	1
T9	L7	0
T8	L2	1
T9	L8	1
T8	L3	1
T9	L9	1
T8	L4	0
T8	L5	0
T8	L6	4
T8	L7	3
T8	L8	1
T8	L9	1
T5	L0	2438
T5	L1	2
T5	L2	3
T5	L3	1
T5	L4	1
T6	L0	2445
T6	L1	5
T5	L5	6
T6	L2	1
T6	L3	2
T5	L6	2
T6	L4	5
T6	L5	1

T6	L6	0
T5	L7	9
T6	L7	2
T6	L8	1
T7	L0	2473
T6	L9	1
T5	L8	4
T7	L1	3
T5	L9	1
T7	L2	1
T7	L3	0
T7	L4	1
T7	L5	0
T7	L6	0
T7	L7	1
T7	L8	0
T7	L9	1
T4	L0	2522
T4	L1	1
T4	L2	2
T4	L3	4
T4	L4	1
T4	L5	1
T4	L6	0
T4	L7	3
T4	L8	1
T4	L9	1

Tabla A1.7: Análisis exhaustivo de los tiempos de acceso a la sesión del usuario desde el disco rígido local; para un caso de uso representativo en cantidad de threads y lecturas por thread.

[T0] Promedio = 224 milisegundos
[T1] Promedio = 231 milisegundos
[T2] Promedio = 230 milisegundos
[T3] Promedio = 239 milisegundos
[T4] Promedio = 253 milisegundos
[T5] Promedio = 246 milisegundos
[T6] Promedio = 246 milisegundos

[T7] Promedio = 248 milisegundos
[T8] Promedio = 241 milisegundos
[T9] Promedio = 240 milisegundos

El promedio general del tiempo de acceso en este caso es de 239 milisegundos. En esta corrida del benchmark, se pueden apreciar varios detalles importantes a recalcar de ejecutar bajo esta arquitectura modificada:

- El paralelismo en el acceso a los archivos de sesión, es menor. Se ve claramente como los accesos del mismo thread se encolan juntos en su mayoría en todos los casos.
- Los primeros accesos de cada thread ([L0]) son los más costosos, pero aún así son considerablemente más veloces que el mismo caso de lectura a la unidad N:.
- **Todos los accesos subsiguientes ([L1] .. [L9]), son instantáneos.**

A1.3 - Conclusión

En la nueva arquitectura del entorno de desarrollo, se realizan las modificaciones necesarias en el software para que los archivos de sesión del usuario puedan almacenarse y leerse en el mismo disco rígido local a la máquina que ejecuta el servidor web. Así se logra una ganancia sustancial en la velocidad de acceso a los mismos, la que se traduce directamente en mayor velocidad de acceso al sitio web y administradores (y todas las operaciones subsiguientes).

A2 - Acceso a base de datos

Otro cuello de botella en la arquitectura original del entorno de desarrollo expuesto en 1.1, es la base de datos. Particularmente, la base de datos utilizada es un Oracle RAC (Oracle Real Application Clusters) de Oracle9i (base de datos relacional, monolítica). La versión instalada en el ambiente de desarrollo es la misma que en el ambiente productivo, existiendo 2 instancias en desarrollo: una instancia ubicada en la oficina de Buenos Aires (a partir de aquí, DESA9iBA) y otra en la de San Luis (DESA9iSL).
La instancia principal resulta ser la que se encuentra en Buenos Aires, estando al día en cuanto a esquemas, configuraciones y datos de prueba respecto de la versión productiva. El acceso aplicativo a esta base de datos desde la oficina de San Luis es costoso (lento), sin mencionar la gran cantidad de operaciones de base de datos que puede llegar a ejecutar cada petición al servidor web, durante las pruebas cotidianas que debe realizar un desarrollador en la aplicación montada en su terminal, o directamente desde un DBMS (DataBase Management System, sistema gestor de base de datos). El acceso a la base de datos de desarrollo local de San Luis es sustancialmente mejor, pero en este caso se encuentra desactualizada en cuanto a esquemas y tablas, y sería necesario no sólo ponerla al día (con el trabajo comparativo que requiere para una enorme base de datos) sino también montar un sistema de constante sincronización entre ambas bases de datos para que se mantengan al día. Aplicar el esquema de sincronización podría resultar tedioso en cuanto a configuración, además del hecho de la gran cantidad de datos que tendrían que transmitirse constantemente a través de la red de ambas provincias, para mantener las copias de las bases de datos actualizadas.
A continuación se realiza otro benchmark desde la oficina de San Luis, para distintas clases de consultas a la base de datos de cada provincia con distintos números de threads, y así poder evidenciar la necesidad de un cambio en la arquitectura original.

A2.1 - Resumen de mediciones

Consulta a una de las principales tablas en el sistema

Se realiza la siguiente consulta a las bases de datos: **select * from customers where cust_id > 10000**. Esta consulta (query) busca todos los datos de los usuarios del sistema, cuyos identificadores son mayores al número proporcionado. Al tratarse de una de las principales tablas, el acceso a la misma suele ser bastante concurrente y la cantidad de resultados (registros) devueltos para el caso es grande también. Es decir, que este es un caso representativo de una query costosa.

Cantidad de threads que realizan la consulta	DESA9iSL		DESA9iBA	
	Tiempo total de ejecución	Tiempo promedio por thread	Tiempo total de ejecución	Tiempo promedio por thread
10 threads	1,1 segundos	1,01 segundos	3,5 segundos	3 segundos
20 threads	2 segundos	0,6 segundos	5 segundos	4 segundos
50 threads	5 segundos	0,2 segundos	10 segundos	5 segundos

Tabla A2.1: Comparación de tiempos de consulta entre la base de datos local y remota; para una query típica a una tabla consultada frecuentemente.

Como se puede apreciar, los tiempos totales de ejecución de la query son, en todos los casos, al menos el doble de rápidos al ser hechos a la base de datos de San Luis, respecto de la instancia ubicada en Buenos Aires. Además, a medida que se aumenta la concurrencia en cantidad de threads, el tiempo promedio de cada acceso mejora mucho más también.

Consulta que resuelve el motor de base de datos

Otro ejemplo a analizar es la siguiente query ejecutada en ambos entornos: **select trunc(sysdate + 1) + 2/24 from dual**. En este caso, la consulta no realiza accesos a disco, sino que se resuelve en el mismo motor de la base de datos, al no consultar esquemas o tablas particulares de la aplicación, sino funciones internas del mismo gestor. El resultset (conjunto de resultado) devuelto es a su vez, mucho menor que el anterior (un sólo resultado en lugar de miles).

Cantidad de threads que realizan la consulta	DESA9iSL		DESA9iBA	
	Tiempo total de ejecución	Tiempo promedio por thread	Tiempo total de ejecución	Tiempo promedio por thread
10 threads	1 segundo	0,9 segundos	2,5 segundos	2,2 segundos
20 threads	2 segundos	0,6 segundos	3,5 segundos	2,6 segundos

50 threads	5 segundos	0,2 segundos	5 segundos	1,8 segundos

Tabla A2.2: Comparación de tiempos de consulta entre la base de datos local y remota; para una query que no requiere acceso a datos.

Nuevamente, los tiempos de acceso a la instancia local son considerablemente menores, y a medida que aumenta la concurrencia, los tiempos promedio de acceso son mejores aún.

A2.2 - Conclusión

De las mediciones anteriores se pone en evidencia la ganancia que resulta el consultar una base de datos de desarrollo, ubicada en la misma red local que la aplicación web. Sin embargo, como se mencionó al principio de este anexo, realizar las modificaciones necesarias para acceder a la instancia de San Luis desde el entorno de desarrollo de la misma oficina, puede tornarse contraproducente en cuestiones de configuración inicial y posterior mantenimiento; al requerir la instalación de un sistema de réplica de datos entre ambas instancias para mantener la consistencia, y el elevado costo de la transferencia constante de estos datos.
La mejor solución entonces, parece ser el consultar una base de datos local a la aplicación, pero sin la necesidad de reubicar el acceso a la base de datos. Por ende, la solución que se desarrolla es, en su lugar, migrar de ubicación a la aplicación web en lugar de la base de datos. Llevando el aplicativo a una instancia virtual montada en Buenos Aires, ubicado en la misma red que la base de datos de desarrollo, se logra el acceso local al recurso, con la ganancia en velocidad que esto implica. La configuración de las terminales virtuales se realiza por única vez al requerir una nueva, sin necesidad de mantenimiento extra, pero con la consiguiente mejora en la velocidad de acceso. Esto se traduce directamente en mayor productividad, debido a la gran cantidad de consultas a la base de datos que se realizan durante la rutina del desarrollador.

A3 - Script de sincronización de recursos de software

En el capítulo 5.3.2 del informe, se habla de Rsync como herramienta de sincronización incremental de archivos entre ambos extremos de la arquitectura de software desplegada; y se presenta un diagrama de actividades que muestra los pasos que debe ejecutar el broker en un script automatizado que realice el proceso. A fines prácticos, aquí se detalla la implementación concreta de dicho diagrama, llevada a cabo para el caso de uso del trabajo realizado, en donde dicho proceso ejecuta en la línea de comandos de un sistema operativo Windows (el cliente), por lo que la implementación se realiza en el lenguaje Batch comprendido por este intérprete:

```
REM Start Rsync daemon if is down
START /MIN "PLINK_FOR_RSYNC" plink -i %3 -v -t -l %1 %2 rsync
--daemon --port=80 --config=/root/rsyncd.conf
REM Sleep for 2 seconds to give plink enough time to finish
START /MIN /WAIT sleep 2
REM Stop Resin
START /MIN "PLINK_FOR_RSYNC" plink -i %3 -v -t -l %1 %2 sh
/data1/resin/bin/httpd.sh stop
REM Iterate through filenames in synclist.txt and rsync them
for /F "tokens=1,2 delims=;" %%i in (%4) do rsync -avz --del %%i
rsync://%1@%2:1873/%%j
REM Sleep for 2 seconds to give rsync enough time to finish
START /MIN /WAIT sleep 2
REM Start Resin
START /MIN "PLINK_FOR_RSYNC" plink -i %3 -v -t -l %1 %2 sh
/data1/resin/bin/httpd.sh start
REM Sleep for 2 seconds to give plink enough time to finish
START /MIN /WAIT sleep 2
REM Kill opened windows
TASKKILL /T /F /FI "WINDOWTITLE eq PLINK_FOR_RSYNC*"
REM Done!
exit
```

El script es invocado desde el broker con los siguientes cuatro parámetros (que el intérprete de comandos de Windows mapea a las variables %1, %2, %3 y %4 respectivamente): archivo de clave privada de identificación de Putty para automatización de login, nombre de usuario, IP de la máquina virtual remota y por último un archivo local suministrado en el mismo paquete de distribución del plugin del broker, con una lista configurable de módulos a sincronizar (synclist.txt, referir a 5.3.2 para ver la especificación de este archivo).
Entonces el script, una vez ejecutado, comienza levantando el demonio remoto de rsync y

durmiendo unos segundos para asegurar que la conexión haya sido establecida. Luego se detiene el servicio de la aplicación web Resin para evitar inconsistencias en su ejecución durante la sincronización de archivos. Paso seguido se ejecuta Rsync sobre la lista de archivos y directorios en el archivo synclist.txt, de modo que sean sincronizados incrementalmente contra cada uno de los módulos definidos en la configuración del demonio remoto. Una vez que la sincronización termina, se vuelve a levantar el servicio del Resin remoto para que tome los cambios realizados en el código fuente (y demás archivos y librerías sincronizadas). Por último, se cierran las ventanas de consola de ejecución de comandos que fueron abiertas durante el proceso.

A4 - Puntos de extensión implementados de Eclipse PDE

En el capítulo 3.4.2 del informe, se introdujo la arquitectura expansible de plug-ins que ofrece el framework de Eclipse, para extender sus funcionalidades y prestaciones. Se explicaron los conceptos de extensión y puntos de extensión, claves en el entendimiento e implementación de este mecanismo de expansión. Un poco más adelante, en el capítulo 5.4.2, se mostraron en forma de diagrama de componentes, cuáles puntos de extensión explícitos fueron extendidos en el desarrollo de la herramienta CASE y con qué fin. Aquí se presenta ahora la implementación concreta, en el lenguaje de marcado XML definido por Eclipse PDE, de dichas extensiones, para comprender su funcionamiento y la forma en que están dispuestas en la implementación propia del plug-in desarrollado (contenido del archivo “plugin.xml”):

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
   <extension
         point="org.eclipse.ui.actionSets">
      <actionSet
            label="%actionSet.label.0"
            visible="true"
            id="brokerplugin.actionSet">
         <action
               class="brokerplugin.actions.RestartApacheAction"
               icon="icons/sync.png"
               id="brokerplugin.actions.RestartApacheAction"
               label="%action.label.3"
               toolbarPath="BROKER_ACTION_SET/BrokerGroup"
               tooltip="%action.tooltip.3">
         </action>
         <action
               class="brokerplugin.actions.RestartResinAction"
               icon="icons/sync.png"
               id="brokerplugin.actions.RestartResinAction"
               label="%action.label.4"
               toolbarPath="BROKER_ACTION_SET/BrokerGroup"
               tooltip="%action.tooltip.4">
         </action>
         <action
               class="brokerplugin.actions.RunJobAction"
               icon="icons/job.gif"
```

```
                id="brokerplugin.actions.RunJobAction"
                label="%action.label.1"
                toolbarPath="BROKER_ACTION_SET/BrokerGroup"
                tooltip="%action.tooltip.1">
          </action>
          <action
                class="brokerplugin.actions.ResinConsoleAction"
                icon="icons/resin.gif"
                id="brokerplugin.actions.ResinConsoleAction"
                label="%action.label.2"
                toolbarPath="BROKER_ACTION_SET/BrokerGroup"
                tooltip="%action.tooltip.2">
          </action>
          <action
                label="%action.label.0"
                icon="icons/sync.png"
                class="brokerplugin.actions.RsyncResinAction"
                tooltip="%action.tooltip.0"
                toolbarPath="BROKER_ACTION_SET/BrokerGroup"
                id="brokerplugin.actions.RsyncResinAction">
          </action>
       </actionSet>
    </extension>
    <extension
          point="org.eclipse.ui.preferencePages">
       <page
             name="%page.name.0"
             class="brokerplugin.preferences.CDSLPreferencePage"
             id="brokerplugin.preferences.CDSLPreferencePage">
       </page>
    </extension>
    <extension
          point="org.eclipse.core.runtime.preferences">
       <initializer
             class="brokerplugin.preferences.PreferenceInitializer">
       </initializer>
    </extension>
    <extension
          point="org.eclipse.ui.views">
       <category
             id="brokerplugin"
             name="%category.name.0">
       </category>
```

```
      <view
            allowMultiple="false"
            category="brokerplugin"
            class="brokerplugin.views.RsyncResinView"
            icon="icons/sync.png"
            id="brokerplugin.views.RsyncResinView"
            name="%view.name.0"
            restorable="true">
      </view>
   </extension>
   <extension
         point="org.eclipse.ui.perspectiveExtensions">
      <perspectiveExtension
            targetID="org.eclipse.jdt.ui.JavaPerspective">
         <view
               id="brokerplugin.views.RsyncResinView"
               ratio="0.5"
               relationship="right"
               relative="org.eclipse.ui.views.TaskList">
         </view>
      </perspectiveExtension>
   </extension>
</plugin>
```

Como se ve, se comienza definiendo el encabezado usual del lenguaje de marcado XML, con el descriptor del tipo de archivo, y para qué plataforma base de Eclipse se está desarrollado. A partir de allí, se especifican los puntos de extensión concretos que se extienden, qué clases implementan dichas extensiones, y algunas opciones o personalizaciones en particular. Además, todas las etiquetas gráficas (textos que se muestran en la interfaz de usuario del plug-in) se encuentran externalizadas (como se sugirió en el workflow de desarrollo de plug-ins presentado en el capítulo 5.4.1 del informe), de modo que sea más conveniente su exportación, a fin de que la herramienta CASE pudiera ser internacionalizada en el futuro, de un modo sencillo. Se detalla ahora brevemente, algunas cuestiones a destacar de las implementaciones realizadas:

- **org.eclipse.ui.actionSets**: se crea un conjunto de acciones, compuesto por cinco botones a ubicar en un mismo grupo de la barra de herramientas de Eclipse, que realizan cada uno de los casos de uso definidos en el capítulo 4.2 del informe. Se especifica qué clase implementa cada acción (las mismas que se definieron gráficamente mediante el diagrama de componentes del capítulo 5.4.2); y se otorga una iconografía representativa a cada botón, de modo que su ubicación en la barra de herramientas sea veloz, a la vez que intuitiva.
- **org.eclipse.ui.preferencePages**: esta extensión no posee muchas definiciones, ya que delega la mayoría de las cuestiones a la implementación concreta de la clase allí

especificada. De este modo, sólo se especifica aquí que el plug-in desarrollado agrega un página de preferencias a la interfaz común de opciones de Eclipse; pero es la clase suministrada la que define qué campos de preferencias mostrar, de qué tipos, y en qué orden (para conocer las opciones presentadas, referir al capítulo 5.4.2 del informe).

- **org.eclipse.core.runtime.preferences**: al igual que la extensión anterior, esta sólo define la clase concreta que implementa todo el comportamiento, sin definir más opciones en absoluto. Lo que se le dice aquí a Eclipse PDE, es que además de poseer una página de preferencias de usuario, el plug-in también debe tener una inicialización en particular de dichas preferencias, cada vez que el IDE inicia. Dichos valores se asignan desde la clase suministrada, que se encarga de traerlos desde el almacén persistente de opciones de Eclipse, o bien asignar valores por defecto en caso de no existir previamente (nuevamente referir al capítulo 5.4.2 para saber qué valores se asignan a cada opción).
- **org.eclipse.ui.views** y **org.eclipse.ui.perspectiveExtensions**: por último se especifican dos extensiones de interfaz de usuario para el plug-in del broker. Concretamente la primera le informa al framework de Eclipse PDE que se define una vista en particular para la acción de sincronización de recursos y mediante qué clase construirla; mientras que la segunda le informa que la disposición de dicha vista en el espacio de trabajo de Eclipse (workbench), debe ubicarse dentro de la perspectiva de desarrollo de Java, y próxima a la vista de la lista de tareas.

A5 - Implementación completa de la herramienta CASE

A fines de estudio y para poder profundizar en detalle la implementación de la herramienta CASE, para el acceso a las características remotas de la arquitectura desplegada; se agrega este anexo para dejar el enlace al desarrollo realizado en Java, como plug-in de Eclipse. El código fuente completo de la herramienta puede obtenerse de la cuenta de GitHub del autor de este trabajo: https://github.com/sebagun/ml-cdsl-eclipse_plugin
Allí se puede ver la implementación completa: el plug-in desarrollado para el IDE Eclipse, los scripts de sincronización de recursos y demás scripts para comunicación con el entorno virtual y para ejecutar diversas tareas o reiniciar servicios, archivos de configuraciones, etc. Esta implementación de la arquitectura presentada en este trabajo, refiere al caso de uso específico de la empresa MercadoLibre, por lo que se entrega con las configuraciones específicas que hacen a, por ejemplo, los recursos de software explícitos que utiliza la aplicación web de la misma, y que deben mantenerse en sincronía en los entornos de desarrollo local y virtual del desarrollador. Obviamente, y como se detalló en este trabajo, la arquitectura es lo suficientemente general y versátil como para ser aplicada en cualquier compañía y sobre cualquier plataforma, realizando la adaptación consecuente de los archivos de configuración.
Para una mayor comprensión de las tareas desarrolladas bajo el marco de este trabajo final de licenciatura, se dividió el código fuente subido en el repositorio en distintos commits, que reflejan el estado de la herramienta en su punto de concepción original, y aquellos agregados y mejoras que fueron realizados específicamente bajo el marco de este trabajo, conforme a lo detallado en el plan de trabajo final presentado oportunamente. Para ver el histórico de cambios sobre el código fuente, a modo de comparación del crecimiento de la herramienta sobre la implementación original, y las características implementadas en este trabajo, el lector puede dirigirse a la sección de commits de dicho repositorio: https://github.com/sebagun/ml-cdsl-eclipse_plugin/commits/master

A6 - Entrevistas a usuarios del entorno de desarrollo de software

En los capítulos 6 y 6.2, se explica la metodología de obtención de feedback aplicada, para relevar el nivel de satisfacción de los desarrolladores de software con la herramienta desarrollada. Esto se realiza a fines poder entender cómo benefició sus tareas de desarrollo y qué aspectos se deberían rever sobre la misma para futuras modificaciones o mejoras. Específicamente en 6.2, se resumen los comentarios más comunes recibidos, pero aquí pueden obtenerse los testimonios completos de cada uno de los usuarios:

- **Libertad Speranza**: “La herramienta mejoró muchísimo la agilidad del desarrollo, y fue fácil de adoptar; siguiendo pocos pasos podías tener todo funcionando rápido.
  Lo que por ahí me hubiera gustado tener, es algún tipo de wiki que tuviera documentación más técnica y profunda, ya que por lo general cuando tenías algún problema y se te rompía la máquina virtual, volver a configurarla para que funcionara bien era complicado, porque no sabías qué era lo que había que modificar.”
- **Sebastián Pérez**: “Recuerdo que era extremadamente lento trabajar en desarrollo, ya que la base de datos estaba en Buenos Aires; entre las mejoras que introdujo el sistema de virtualización, recuerdo como principal el incremento de la velocidad de desarrollo.”
- **Horacio Casatti**: “Principalmente, el paso del entorno de desarrollo original al esquema con SVN (Subversion) que se empezó a usar luego, simplificó bastante el proceso, principalmente con respecto a mantenimiento y mejoras.
  Como segundo punto importante, el tener un repositorio funcional sumó muchísimo también, ya que antes era muy difícil poder probar un desarrollo en modo local.
  El agregado a esto de las máquinas virtuales de desarrollo, creo que le sumó bastante en el sentido de replicar un ambiente más similar a producción, más controlado.
  Teniendo en cuenta los contras de este modelo (el tener que sincronizar con estas máquinas virtuales, configuraciones y demás), el plugin de Eclipse ayudó a manejar esto de manera simple (incluso gente de experiencia de usuario que no están tan acostumbrados a desarrollo lo han usado), y en poco tiempo se hizo de uso automático (nadie tenía que salir a buscar documentación para ver cómo se usaba).
  Lo único que creo que hubiera sumado mucho en ese momento es el soporte del plugin de Eclipse para Linux, ya que cuando se cambió el equipamiento nos vimos obligados a tener máquinas virtuales con Windows sólo para trabajar con este ambiente de desarrollo, y eso entorpeció un poco las tareas.”

Habiendo meditado sobre este feedback, es que en el capítulo 6.4 se seleccionan algunas de estas ideas o sugerencias para ser consideradas como parte del trabajo futuro sobre la herramienta.

Referencias

- [Abbott11] Abbott y Fisher, “Scalability Rules: 50 Principles for Scaling Web Sites”, Ed. Addison Wesley, 2011.
- [Alegsa14] Alegsa, “Diccionario de informática y tecnología”, 2014.
- [Alwis02] Alwis, Azad y Windatt; “PDE FAQ”; Eclipse Project; 2012.
- [Aniszczyk08] Aniszczyk, IBM Software Group, “Plug-in development 101: Learn the basics of Eclipse plug-in development and rich-client applications”, 2008.
- [Arevalillo04] Arevalillo, "La linterna del traductor", 2004.
- [Azure14] “Cloud development and test environments”, Microsoft Azure, 2014.
- [Bondi00] Bondi, “Characteristics of scalability and their impact on performance”, Second International Workshop on Software and Performance, 2000.
- [Denning76] Denning, "Fault tolerant operating system", ACM Computing Surveys, 1976.
- [Dorband10] Dorband, Palencia y Ranawake, “Commodity Computing Clusters at Goddard Space Flight Center”, Goddard Earth Sciences & Technology Center, 2010.
- [EDocExtPoints] Eclipse Documentation, http://help.eclipse.org/, “Platform Plug-in Developer Guide > Reference > Extension Points Reference”.
- [EDocRDT] Eclipse Documentation, http://help.eclipse.org/, “Remote Development Tools User Guide > Concepts > Overview”.
- [EDocRSE] Eclipse Documentation, http://help.eclipse.org/, “RSE User Guide > Getting Started”.
- [EMission] Eclipse Foundation, http://www.eclipse.org/, “Eclipse Platform Overview: Mission”.
- [EPTP] Eclipse Foundation, http://www.eclipse.org/ptp/, “Eclipse PTP”.
- [Garg05] Garg, “Backing up Windows machines using rsync and ssh”, A.P. Lawrence, 2005.
- [GNUEmacs13] Proyecto GNU, “GNU Emacs FAQ for MS Windows”, 2013.
- [Hadoop14] “Hadoop Overview: Project Description”, Apache Hadoop Wiki, 2014.
- [Hashimoto13] Hashimoto, “Vagrant: Up and Running”, Ed. O'Reilly, 2013.
- [Hill90] Hill, "What is scalability?". ACM SIGARCH Computer Architecture News 18, 1990.
- [Holve99] Holve, “A Tutorial on Using rsync”, Everything Linux, 1999.
- [IPaDPS07] Michael, Moreira, Shiloach y Wisniewski; "Scale-up x Scale-out: A Case Study using Nutch/Lucene", IEEE International Parallel and Distributed Processing Symposium, 2007.
- [Karne03] Karne, Bishop; “A Survey of middleware”; Towson University; 2003.
- [Kuhn89] Kuhn, "Selecting and effectively using a computer aided software engineering tool", Annual Westinghouse computer symposium, 1989; Pittsburgh, PA (EEUU); DOE Project.
- [Loogic08] “Virtualización: beneficios para una empresa”, loogic.com, 2008.

- [LouKa95] Loucopoulos y Karakostas, "System Requirement Engineering", Ed. McGraw-Hill, 1995.
- [Mackall06] Matt Mackall, “Towards a Better SCM: Revlog and Mercurial”, Ottawa Linux Symposium Proceedings, 2006.
- [Maxino06] Maxino, “Revisiting Fletcher and Adler Checksums”, Carnegie Mellon University, 2006.
- [Micro11] Chosnyk, Coulombe, Denny, Kaczmarek, Maher, Stewart y Stowe; "Infrastructure Planning and Design, Selecting the Right Virtualization Technology"; versión 2.3, Copyright © 2011 Microsoft Corporation (Creative Commons).
- [MicroTCV1] "Implementación de una infraestructura de nube privada mediante virtualización", TechCenter de Virtualización, Microsoft Corporation, 2014.
- [MicroTCV2] "Concretar tareas con virtualización", TechCenter de Virtualización, Microsoft Corporation, 2014.
- [Mogul00] Mogul, “What is HTTP Delta Encoding?”, Compaq Computer Corporation Western Research Laboratory, 2000.
- [Omar11] Abid Omar, “How to Sync A Local & Remote WordPress Blog Using Version Control”, tutsplus.com, 2011.
- [OraCust11] “MercadoLibre Inc. Ahorra 75% en Consumo Energético con Datacenter Consolidado, Virtualizado y Escalable”, Oracle Customer Snapshot, Febrero de 2011.
- [OraPress09] “MercadoLibre, Inc. Saves with Oracle® VM and Oracle Unbreakable Linux”, Oracle Press Release, Octubre de 2009.
- [OraTT] “Oracle TimesTen In-Memory Database”, Oracle Data Sheet, Oracle, 2014.
- [OraVM14] “Oracle VM Security Guide”, 3er edición, Oracle, 2014.
- [Palat12] Jay Palat, “Introducing Vagrant”, Linux Journal, 2012.
- [Press05] Pressman, “Ingeniería del software: un enfoque práctico”, 6ta edición, Ed. McGraw-Hill/Interamericana Editores, 2005.
- [Rehman02] Rehman y Paul, “The Linux Development Platform: Configuring, Using and Maintaining a Complete Programming Environment”, Ed. Prentice Hall, 2002.
- [RFC1950] Deutsch y Gailly, “ZLIB compressed data format specification”, versión 3.3, 1996.
- [RFC3229] The Internet Engineering Task Force (IETF), “RFC 3229: Delta encoding in HTTP”, 2002.
- [Saltzer75] Saltzer y Schroeder, "The Protection of Information in Computer Systems", 1975.
- [Somm07] Sommerville, "Software engineering", 8va edición, Ed. Addison-Wesley, 2007.
- [Tennen96] Tennenbaum, “Sistemas Operativos Distribuidos”, Ed. Prentice Hall, 1996.
- [TeXlipse14] TeXlipse, "TeXlipse homepage - LaTeX for Eclipse", 2014.
- [Turban08] Turban, King, Lee, y Viehland; “Electronic Commerce, A Managerial Perspective”, 5ta edición, Ed. Prentice-Hall, 2008.
- [Watson05] Watson y Rasmussen, "A Strategy for Addressing the Needs of Advanced Scientific Computing Using Eclipse as a Parallel Tools Platform", Los Alamos National Laboratory, 2005.

- [White12] White, “Hadoop: The Definitive Guide”, 3er Edición, Ed. O'Really, 2012.
- [Willey05] El-Rewini y Abd-El-Barr, "Advanced Computer Architecture and Parallel Processing", Ed. Wiley & Son, 2005.
- [Wiley96] Buschmann, Meunier, Rohnert, Sommerlad y Stal; “Pattern-Oriented Software Architecture, Volume 1: A System of Patterns”, Ed. Wiley, 1996.
- [Witten99] Ian Witten, Alistair Moffat y Timothy Bell; “Managing Gigabytes: Compressing and Indexing Documents and Images”; 2da edición; Ed. Academic Press; 1999.