UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

Estudiantes

Nicolás Doroskevich

Sebastián Moreira

Tutores

Federico Piedrabuena

Raúl Ruggia

Proyecto de grado 2004

Facultad de Ingeniería – Universidad de la República

Resumen – El presente informe detalla un framework de deployment genérico, basado en dos premisas fundamentales: proveer mecanismos de extensión mediante el uso de plugins y simplificar al máximo su personalización. Definida una plataforma de aplicación, el framework provee la infraestructura sobre la cual elaborar la solución de de deployment para la misma.

Tomando como caso de estudio el desarrollo del componente de deployment para la plataforma LinkAll, se describe cómo el framework puede ser utilizado para la construcción del mismo, considerando que la plataforma se sustenta en las funcionalidades de un servidor J2EE.

Mediante este trabajo, se pretende introducir al lector a la problemática vinculada con el deployment de aplicaciones haciendo hincapié en la elaboración de un framework genérico sobre el cual construir una solución para un sistema basado en servicios de la plataforma J2EE.

Palabras clave – Sistemas de Información, Bases de Datos, Deployment, Servidores de aplicación, J2EE.

Índice general

1. Introducción

El presente proyecto se encuentra enmarcado dentro del desarrollo de una plataforma de servicios federados a través del Web denominada LinkAll. Dentro de los distintos componentes que conforman la plataforma LinkAll, el componente de deployment es de especial interés puesto que se requiere contar con un mecanismo que permita la inclusión de nuevas aplicaciones, las cuales se van a ejecutar dentro de un servidor de aplicaciones J2EE.

Este contexto ha servido de motivación para la construcción de un framework de deployment genérico, basado en dos premisas fundamentales: proveer mecanismos de extensión mediante el uso de plugins y simplificar al máximo su personalización. Definida una plataforma de aplicación, el framework provee la infraestructura sobre la cual elaborar la solución de de deployment para la misma.

Simplificar la extensibilidad y facilitar su personalización, son características deseadas debido a la propia naturaleza de la solución. Como framework, éste debe brindar los servicios más básicos en materia de deployment y proveer diversos mecanismos que permitan a los desarrolladores aplicarlo a un cierto problema objetivo.

Tomando como caso de estudio el desarrollo del componente de deployment para la plataforma LinkAll, en este informe se describe cómo el framework puede ser utilizado para la construcción del mismo.

Como proyecto embebido dentro del proyecto LinkAll, la solución presentada debe respetar las restricciones que su propia arquitectura determine. Dicha arquitectura se basa en servidores que almacenan datos y exportan servicios.

Otro elemento importante a considerar es la naturaleza de los usuarios de LinkAll. Se exige facilidad de uso, acceso simple a los datos, y un sistema robusto en recuperación de errores por fallas de sistema y de conectividad.

2. Objetivos generales

En este capítulo se describen los objetivos específicos del proyecto, enmarcados dentro del contexto de los objetivos definidos para la plataforma LinkAll.

El objetivo principal es el desarrollo de funcionalidades de:

- **Deployment** de nuevos componentes por parte de usuarios no técnicos.
- **Configuración** y registro del modelo de datos y metadatos a ser usado por los componentes a ser instalados (deployados).
- **Re-deployment** y **des-deployment** (eliminación) de aplicaciones, manteniendo la base de datos y su configuración. Para esto se debe analizar, diseñar e implementar las estructuras de datos auxiliares que almacenen los metadatos utilizada en estas funcionalidades.
- **Registro del estado** de cada instalación de forma de poder recomendar la instalación de nuevas liberaciones.

Como objetivo secundario, se deben desarrollar utilitarios que permitan monitorear el estado de los deployments de forma de poder retomar la misma en caso de fallas o deshacer los pasos realizados dejando el sistema en su estado original.

Como se mencionó anteriormente, se definieron además otros objetivos relacionados con algunos factores de calidad del producto a construir:

- **Facilidad de uso** debido a la naturaleza de los usuarios, los cuales no son expertos.
- Proveer mecanismos de **extensibilidad** que permitan una fácil personalización del producto.

3. Marco conceptual

Como punto de partida, es necesario definir ciertos conceptos de uso frecuente en las restantes líneas de éste informe. También es importante comentar el estado del arte en materia de deployment de software[1] para así poder comprender los pilares en los cuales se basa el presente trabajo. En este capítulo se cubren estos aspectos de manera breve y concisa.

Existen diversos trabajos realizados por distintos autores en los cuales se pretende definir deployment, así también como explicar sus implicancias. Algunos de estos trabajos han hecho aportes que permiten formalizar el proceso de deployment de software. Dentro de esta línea, se define al proceso de deployment de software como la **transferencia**, **ensamblado** y **mantenimiento** de una **versión** particular de un software en un cierto **sitio** [4]. Un sitio es la plataforma, o ambiente, de destino en donde se pretende instalar una cierta versión de un software. La transferencia implica el envío del software hacia el mencionado ambiente destino; el ensamblado se relaciona con la incorporación de cada componente del mismo a la plataforma. Posteriormente al ensamblado, el software ingresa en una etapa de mantenimiento por parte de los interesados.

Así como se ha definido el proceso, también se define el ciclo de vida de deployment de software como una serie de actividades y subprocesos por los que un cierto software atraviesa dentro de un sistema de deployment. Dicho ciclo de vida, brinda una perspectiva de todas las funcionalidades que este tipo de sistemas debería tener y por las cuales el software debe pasar desde que es creado hasta que queda obsoleto. El ciclo completo se describe gráficamente en el siguiente diagrama:

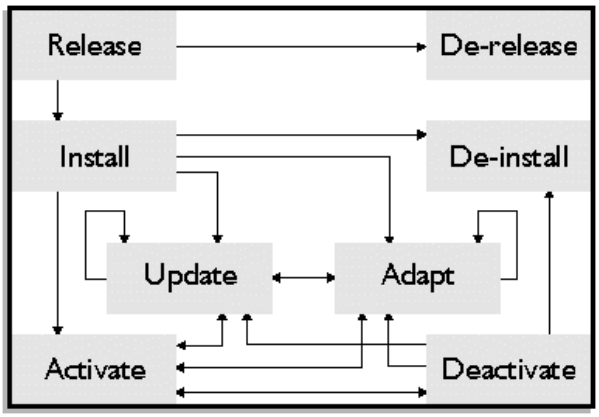


Fig. 3-1 - Ciclo de vida de deployment de software

Release (liberación)

El proceso de liberación es la interfaz entre el proceso de desarrollo y el proceso de deployment. Se compone de dos actividades, la preparación y la publicación. La **preparación** consiste en empaquetar el software conteniendo sus componentes, dependencias, restricciones y toda información requerida en otras etapas del ciclo de vida. La **publicación** es la actividad mediante la cual es posible exponer los paquetes de tal forma de que estos estén disponibles para ser obtenidos desde cualquier sitio o plataforma.

[1] En este informe se utilizará “deployment de software” o simplemente “deployment” para referirse al mismo concepto.

Install (instalación)

Este proceso cubre la inserción inicial de un sistema en un sitio. Usualmente, éste es el más complejo de los procesos de deployment debido a que todos los recursos necesarios deben ser encontrados y ensamblados. En el proceso de instalación, el paquete creado en el proceso de liberación es abierto para posteriormente **transferir** cada componente a la plataforma e interpretar sus metadatos de tal manera de **configurar** el sitio objetivo adecuadamente para asegurar una correcta inserción del nuevo aplicativo.

Activate (activación)

La activación se refiere a la inicialización de aquellos componentes de un sistema que deban ejecutarse para que el mismo pueda ser utilizado. En el caso de grandes sistemas, la activación puede ser especialmente compleja porque podría requerir la re-inicialización de servidores, sistemas cliente, sistemas de base de datos, etc.

Deactivate (desactivación)

La desactivación es la inversa de la activación, y se relaciona con la finalización de la ejecución de cualquiera de los componentes de un sistema. La desactivación es a menudo requerida para la realización de otras tareas de deployment, por ejemplo, un software podría requerir ser desactivado previo a su actualización.

Update (actualización)

La actualización implica la existencia de un cierto software instalado con anterioridad. Una actualización es una nueva versión de dicho software que puede solucionar ciertas fallas (bugs) o agregar nuevas funcionalidades. Las actualizaciones pueden ser menos complejas que las instalaciones puesto que muchos de los recursos requeridos ya han sido obtenidos durante el proceso de instalación. Aunque un sistema es usualmente desactivado antes de una actualización, eso no es una regla. Algunos sistemas podrían requerir una operativa continua durante el proceso de actualización, en tales casos, este proceso podría tornarse bastante más complejo.

Adapt (adaptación)

El proceso de adaptación implica la modificación de un sistema de software que ha sido previamente instalado en un sitio. La adaptación difiere de la actualización puesto que la segunda responde a eventos remotos, mientras que la primera responde a eventos locales. Por ejemplo, si la configuración de un sitio cambia afectando a un sistema instalado, puede ser necesario aplicar, sobre el mismo, diversas medidas correctivas.

De-install (desinstalación)

En algún momento, un sistema puede dejar de ser requerido en un sitio y puede ser desinstalado. La desinstalación no es necesariamente un proceso trivial. Se debe prestar especial atención y sumo cuidado con respecto al manejo de recursos compartidos tales como archivos de datos y librerías.

De-release (sistema obsoleto)

Un sistema puede ser marcado como obsoleto, y su responsable puede dejar de brindar soporte. Pasar a un estado obsoleto difiere de la desinstalación en el sentido de que el sistema pasaría a quedar no disponible para cualquier instalación en cualquier sitio, aunque éste no sea removido de aquellos sitios que aún lo tengan en producción.

El ciclo de vida presentado permitirá más adelante, al momento de presentar conclusiones, contrastar la solución propuesta con la solución ideal. Es importante aclarar que no existen sistemas que abarquen en su totalidad todos los procesos y actividades mencionados, y son pocos aquellos que cubren más del 50% de los mismos.

4. Marco tecnológico

El presente proyecto, se enmarca dentro del contexto de una plataforma basada en los servicios de la plataforma J2EE. Una implementación de dicha plataforma, es la brindada por el servidor de aplicaciones Jboss. Durante el desarrollo, Jboss fue utilizado como sitio (o plataforma) de prueba, y es el ambiente sobre el cual se desarrolla la propia plataforma LinkAll. Es importante ilustrar algunos conceptos vinculados con la especificación de la plataforma J2EE, para posteriormente explicar muy brevemente las principales características del servidor Jboss.

4.1 Plataforma J2EE

La plataforma **Java 2 Platform Enterprise Edition (J2EE)** [2] de Sun, es un conjunto de especificaciones y prácticas coordinadas que juntas permiten brindar soluciones para el desarrollo, deployment, y gestión de aplicaciones multicapa, típicas en sistemas de porte empresarial. Provee valor mediante la reducción significativa de costos y complejidad en el desarrollo y deployment, resultando en servicios que pueden ser rápidamente desarrollados y desplegados, además de ser fácilmente gestionados.

La plataforma J2EE utiliza un modelo de aplicación distribuida multicapa para las aplicaciones empresariales. La lógica de la aplicación se divide en componentes de acuerdo a su función, y éstos a su vez son instalados en diferentes máquinas dependiendo la capa a la cual pertenezcan. Una aplicación J2EE está dividida en al menos las siguiente capas:

- **Capa cliente**: componentes ejecutándose en la máquina del cliente.
- **Capa Web**: componentes ejecutándose en el servidor J2EE.
- **Capa de negocios**: componentes ejecutándose en el servidor J2EE.
- **Capa de sistema de información empresarial (EIS)**: software ejecutándose en el servidor EIS.

Desde este punto de vista, la capa cliente podría contener desde una aplicación de escritorio a una serie de páginas HTML dinámicas, las cuales se procesan localmente en la máquina del cliente. Por otro lado, en la capa Web se cuenta con elementos basados en tecnologías tales como JavaServer Pages (JSP) y Java Servlet, que son componentes en el servidor capaces de generar contenido HTML. La capa de negocios contiene la lógica principal de la aplicación y la tecnología fundamental aplicada a este nivel son los Enterprise JavaBeans (EJB). Finalmente, la capa EIS es típicamente un sistema de base de datos ejecutándose en un servidor dedicado para tales fines.

A pesar de que las aplicaciones empresariales pueden consistir en tres o cuatro capas, las aplicaciones J2EE son generalmente consideradas de tres capas debido a que ellas son típicamente distribuidas entre tres nodos: la máquina cliente, la maquina del servidor J2EE y la máquina del sistema de base de datos. Este modelo extiende el ya bastante conocido y difundido modelo de dos capas o “cliente-servidor”, interponiendo un servidor de aplicaciones multihilado entre la aplicación cliente y el almacenamiento de los datos en el backend.

Ya se han mencionado algunas tecnologías aportadas por la plataforma J2EE, sin embargo el abanico es amplio e imposible de cubrir en el presente informe. Simplemente, y a modo de pequeña referencia, listaremos aquellas más importantes: Enterprise JavaBeans (EJB), Java Servlet, JavaServer Pages (JSP), Java Message Service API (JMS), Java Transaction API (JTA), Java API for XML Processing (JAXP), Java API for XML-Based RPC (JAX-RPC), SOAP with Attachments API for Java (SAAJ), Java API for XML Registries (JAXR), J2EE Connector Architecture (JCA), JDBC API, Java Naming and Directory Interface (JNDI), Java Authentication and Authorization Service (JAAS), Java Management Extensions (JMX).

Cualquiera de estas tecnologías puede ser nombrada más adelante en este informe por lo que se encomienda al lector, en caso de ser necesario, buscar más información al respecto.

4.2 Servidor de aplicaciones Jboss

JBoss [3] es una implementación de código abierto bajo licencia LGPL de la plataforma J2EE. Su arquitectura es altamente modular basada en un mecanismo de plugins. Utiliza el estándar Java Management eXtentions (JMX) para la administración de los distintos componentes y servicios de la plataforma. La siguiente figura muestra los principales componentes de JBoss interactuando por medio de JMX:

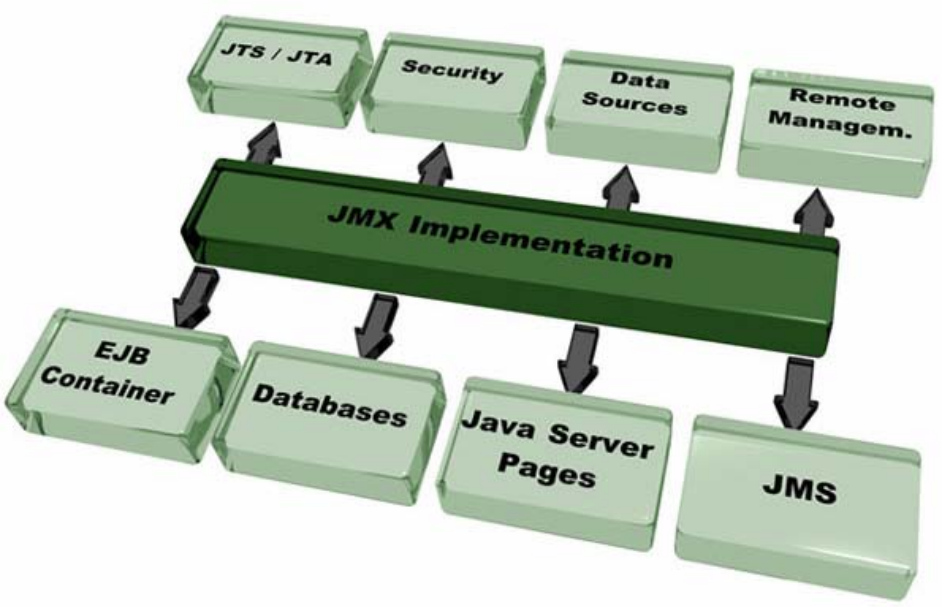


Fig. 4-1 – Arquitectura Jboss basada en JMX

Actualmente, el servidor de aplicaciones Jboss se encuentra certificado por Sun como un servidor 100% J2EE 1.4-compliant, es decir, que cumple en su totalidad con dicha versión de la especificación J2EE. Debido a la amplia variedad de especificaciones que conforman dicha espcififcación, la inclusión de una explicación detallada de la implementación que Jboss realiza de la misma sería demasiado extensa como para ser cubierta en este informe, sin embargo, en [3] es posible encontrar abundante información al respecto.

Jboss brinda la infraestructura sobre la cual se basa la plataforma LinkAll. Debido a ello, el framework de deployment que en este informe se presenta, fue desarrollado teniendo en cuenta dicho aspecto pero sin perder la generalidad en su solución.

5. Framework de deployment

En este capítulo se presenta la solución propuesta. En primera instancia, se enumeran las principales funcionalidades del framework. Posteriormente se describe la arquitectura del mismo, brindado una visión de alto nivel de los diversos componentes, y de cómo éstos interactúan para la correcta realización de las tareas de deployment. Por último, se incluye una breve explicación acerca de cómo utilizar el framework en base a los mecanismos de extensión y/o personalización que él provee.

5.1 Principales funcionalidades

5.1.1 Ciclo de vida de deployment

En el capítulo 3, se detallaron las distintas actividades que un sistema de deployment completo debería soportar para cubrir la totalidad del ciclo de vida que atraviesa una cierta versión de una aplicación, brindando un primer paso de acercamiento a las distintas funcionalidades que el framework de deployment provee, aspirando a alcanzar la completitud en tal sentido.

5.1.1.1 Liberación

Con respecto a las funcionalidades de liberación, el framework brinda un mecanismo de para describir el contenido de una cierta versión de una aplicación. Por otro lado, y considerando actividades posteriores como aquellas de instalación/desinstalación, también brinda un mecanismo para la especificación de acciones (tareas) a realizar al momento de deployment y undeployment. Todos estos elementos, junto con los componentes que forman parte de una aplicación, son empaquetados dentro de un único archivo, quedando disponibles para continuar con las subsiguientes etapas del ciclo de vida de la misma. Este paquete se corresponde con el concepto de **Paquete Deployable**.

Para la descripción de una aplicación, se introduce el concepto de **Descriptor de Aplicación**, archivo desde el cual es posible obtener información referente al nombre de la misma, su versión, su autor, entre otros datos. Las dependencias y restricciones pueden ser incluidas dentro este descriptor, ambas bajo el concepto de **Dependencia**. Por defecto, el framework brinda una dependencia básica mediante la cual una aplicación puede exigir la existencia de otra previamente instalada en la plataforma destino. No obstante, es posible incluir nuevos tipos de dependencias utilizando los mecanismos de extensión existentes, los cuales se verán en detalle más adelante en este capítulo.

Con respecto a la especificación de acciones a realizar al momento de una instalación, y posteriormente al momento de la desinstalación, se introduce el concepto de **Especificación de Deployment**. Este archivo de especificación consiste en una lista de **acciones** parametrizadas divididas en dos categorías, acciones de deployment y acciones de undeployment. Cada acción puede incluir una serie de parámetros que las personaliza. A modo de ejemplo, si una acción ejecuta un script SQL sobre una base de datos, dicha acción podría ser parametrizada incluyendo el string de conexión, el nombre de usuario y su respectiva contraseña, y una referencia al script.

Finalmente, el framework incluye un mecanismo de publicación de paquetes deployables los cuales pueden ser obtenidos desde un servidor de paquetes, o nodo soporte. En la sección 5.2 aparece una descripción más detallada de la arquitectura propuesta y los distintos nodos participantes en la topología de la solución.

5.1.1.2 Instalación

La instalación, consta de dos pasos: transferencia y configuración.

La transferencia se logra mediante la ubicación en el sitio objetivo, o nodo objetivo según la terminología del framework, de una aplicación encapsulada en un paquete deployable. Este paquete puede ser obtenido mediante algún mecanismo de descarga desde el servidor de paquetes, o por copia directa en la plataforma objetivo. En caso de descarga la funcionalidad es responsabilidad del framework, mientras que en caso de copia directa la responsabilidad corre por cuenta del usuario administrador de la plataforma objetivo.

La configuración se realiza procesando el Descriptor de Aplicación y la Especificación de Deployment. El elemento crucial en esta etapa es sin duda el segundo mencionado. Se toman las acciones de deployment incluidas en la Especificación de Deployment y se procede a la ejecución de cada una de las mismas en el orden especificado y en base a los parámetros establecidos para cada una de ellas. Como resultado, una aplicación debería quedar correctamente incorporada al ambiente objetivo luego de haberse ejecutado la totalidad de las acciones indicadas.

Del mismo modo que las dependencias en el Descriptor de Aplicación, el framework puede ser extendido con nuevos tipos de acciones además de las que él provee por defecto. Dentro de estas acciones básicas incorporadas por defecto, se destacan aquellas que permiten operar sobre la plataforma J2EE puesto que son de interés para la elaboración del componente de deployment de la plataforma LinkAll. En posteriores secciones se abundará más en lo referente a las acciones que son de utilidad para LinkAll.

5.1.1.3 Activación y desactivación

La activación consiste en la inicialización de aquellos componentes de una aplicación que deban ejecutarse para que la misma pueda ser utilizada. Por otro lado, la desactivación es el proceso inverso, es decir, la finalización de los componentes evitando que la aplicación pueda ser utilizada.

La activación y desactivación de aplicaciones incorporadas a una plataforma objetivo corre por cuenta del administrador. Una acción de deployment podría incorporar un componente a la plataforma sin necesidad de auto-activar el mismo y posponer la activación hasta el momento en que un administrador la realice mediante algún software de gestión del ambiente objetivo. Otro enfoque diferente podría realizar la activación luego de la instalación. En todos los casos, el framework no determina un mecanismo específico de activación, por lo que esta funcionalidad deberá ser provista por la propia plataforma objetivo.

Del mismo modo, la funcionalidad de desactivación deberá formar parte de la plataforma objetivo por las mismas razones expuestas anteriormente.

5.1.1.4 Adaptación

La adaptación implica contar con mecanismos de modificación en la configuración de las aplicaciones instaladas en la plataforma objetivo, para poder aplicar medidas correctivas en caso de ser necesario. La actual versión del producto no provee funcionalidades que permitan editar la configuración de cada una de las aplicaciones instaladas, sin embargo, es viable implementar una solución de adaptación de aplicaciones mediante la modificación de la información persistida ya sea manualmente o haciendo uso de las utilidades del framework.

5.1.1.5 Actualización

La actualización de una aplicación, al igual que la instalación, requiere la transferencia de la nueva versión al sitio objetivo y su posterior re-configuración. La solución propuesta no considera aún el manejo de este tipo de circunstancias debido a la complejidad que supone la re-configuración del ambiente objetivo. No obstante, aún es posible la realización de una desinstalación-instalación, o redeployment.

5.1.1.6 Desinstalación

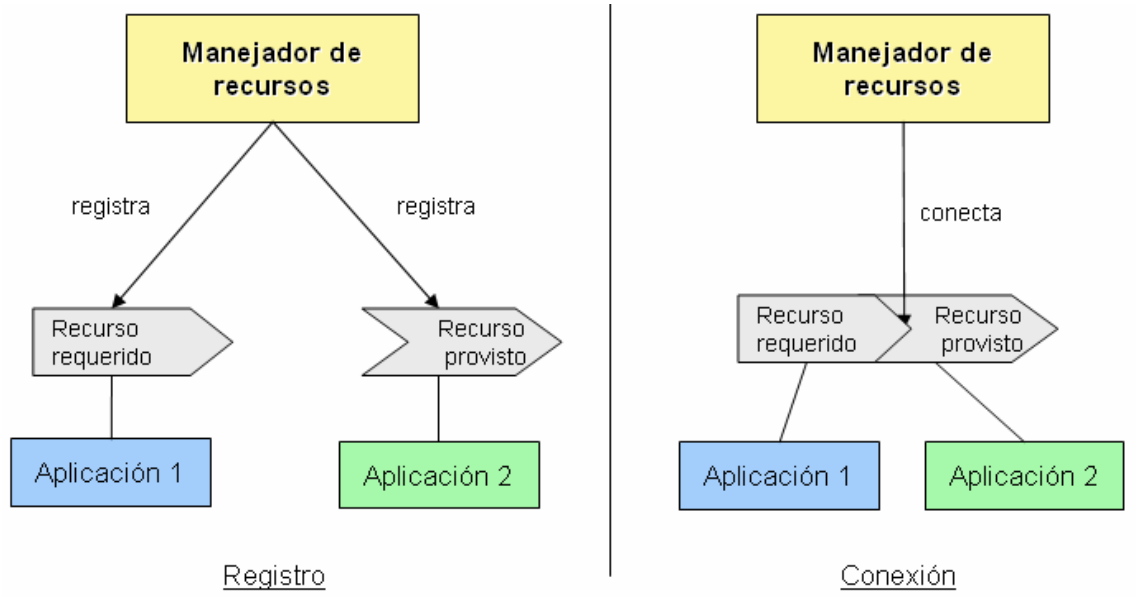
La desinstalación de una aplicación se realiza de manera similar a la instalación, salvo que las acciones consideradas son aquellas de undeployment. Luego de la desinstalación, la aplicación queda eliminada del sitio objetivo y ya no queda disponible para su uso.

5.1.1.7 Sistema obsoleto

Una aplicación no puede ser directamente marcada como obsoleta, sin embargo, eliminándola del servidor de paquetes se obtiene un mismo efecto, puesto que ésta no estará más disponible y así ningún sitio objetivo podrá acceder e instalar la misma.

5.1.2 Gestión de recursos

El framework incluye un componente para la resolución de la conectividad entre los distintos componentes que conforman las aplicaciones instaladas en una plataforma objetivo, y la localización de otros tipos de recursos. Dicho componente, denominado **Manejador de recursos**, registra recursos requeridos y provistos por una aplicación.



La centralización de dicha información en este componente permite:

- Realizar modificaciones en la configuración de los recursos de cada aplicación de tal manera de poder re-configurar el acceso a recursos en caso de que estos cambien su localización en la plataforma objetivo, ya sea por modificaciones a la misma o por la aplicación de cambios estructurales por parte de un usuario administrador (adaptación).
- Resolver de manera automática la localización y conexión de aquellos recursos requeridos por una aplicación. Una aplicación requiriendo un recurso delega al componente de gestión de recursos la responsabilidad de encontrarlo y de realizar el enlace con el mismo, o bien hacerle llegar la información necesaria para que la misma pueda efectivizar la conexión.

El manejador de recursos recibe como entrada una **Especificación de Recursos** en donde se listan los recursos provistos y requeridos por una aplicación. La especificación de recursos es registrada al momento de realizarse una instalación, y su procesamiento es indicado mediante una acción de deployment.

En la sección 5.5 se incluye una descripción de la utilización de la Especificación de Recursos para registrar recursos provistos y requeridos, y de cómo se utiliza manejador de recursos para resolver el acceso a los recursos provistos.

5.1.3 Extensión y personalización

Gran parte de las funcionalidades que un sistema de deployment debe incluir son implementadas en el propio framework. Sin embargo, es posible extenderlo agregando nuevas funcionalidades, o bien personalizarlo para que ciertos componentes sean sustituidos por nuevas implementaciones de los mismos, de tal forma de cumplir con los requerimientos de cada plataforma objetivo.

Los puntos de extensión considerados son:

- Tipos de dependencias.
- Tipos de acciones.
- Tipos de recursos.

Los componentes personalizables son:

- Componente de conexión entre la plataforma objetivo y el servidor de paquetes.
- Componentes de lectura y escritura de archivos de configuración (Descriptor de Aplicación, Especificación de Deployment, Especificación de Recursos).
- Componente de abstracción de la plataforma objetivo.

Todas las extensiones pueden ser incorporadas del mismo modo que una aplicación es instalada en la plataforma objetivo, es decir, como una aplicación en sí misma. En este caso, se deben incluir la ejecución de acciones de deployment específicas para tales fines en la Especificación de Deployment (ver 5.6).

Para el caso de los componentes personalizables, todos incluyen una implementación por defecto y pueden ser sustituidos por otras nuevas implementaciones mediante re-configuración del framework.

5.2 Arquitectura de la solución

El ciclo de deployment sugiere la existencia de dos nodos, el servidor de paquetes y el sitio objetivo en donde se instalan los mismos. En la figura presentada a continuación (Fig. 5-1), la cual consiste en la arquitectura general de la solución, aparecen dichos nodos pero bajo la denominación de **Nodo soporte (SN)** y de **Nodo objetivo (TN)**. Por otro lado, ambos nodos permiten el control remoto desde un nodo cliente ya sea por medio de una interfaz web utilizando un browser, como cualquier otra alternativa para gestionar remotamente cada uno de ellos.

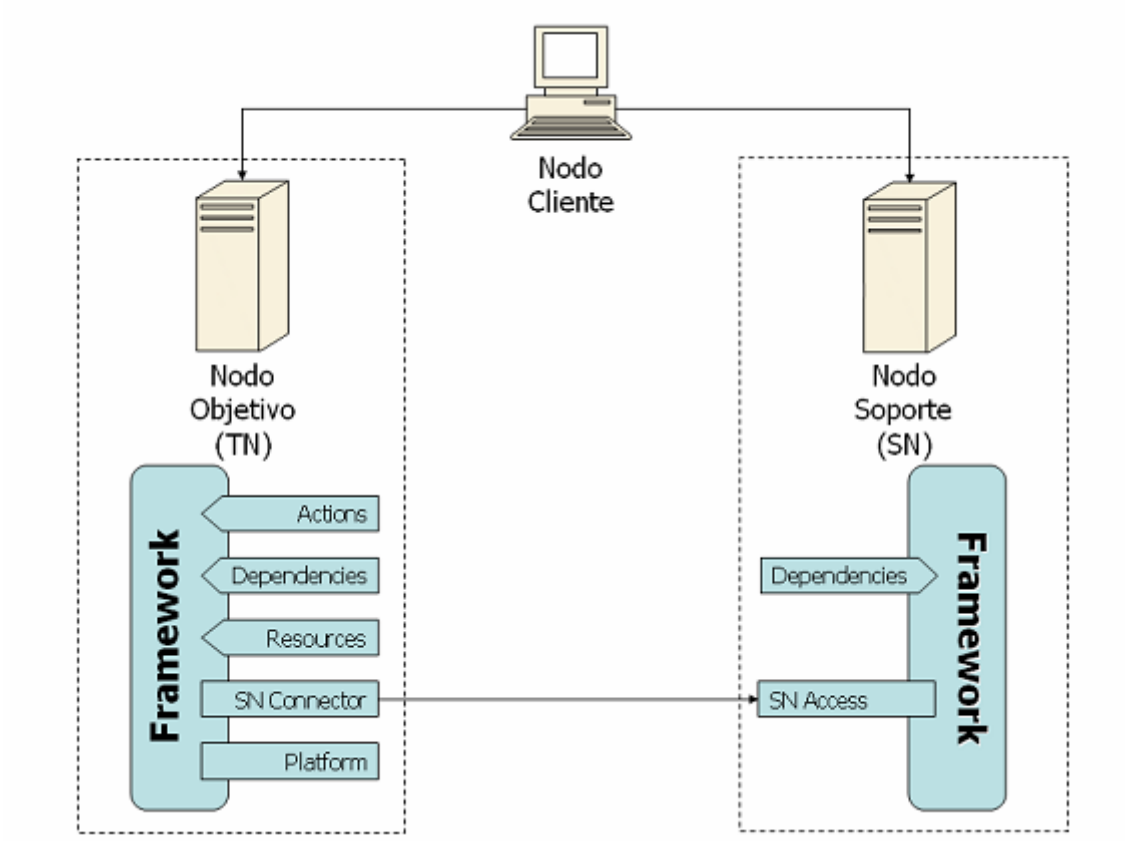


Fig. 5-2 – Arquitectura de la solución

Cada nodo se basa en las funcionalidades de un framework común el cual puede ser extendido y personalizado para cumplir con los requerimientos del sistema de deployment donde se aplique el mismo.

Como se mencionó en la sección 5.1, se consideran ciertos puntos de extensión y otros de personalización. Dentro de los primeros, nuevas acciones (actions), nuevas dependencias y nuevos recursos (resources) pueden ser incorporados en la forma de plugins. Dentro de los componentes personalizables, se tiene el componente de conexión entre la plataforma objetivo y el servidor de paquetes (node connector), los componentes de lectura y escritura de archivos de configuración (readers y writers), y el coomponente de abstracción de la plataforma objetivo (platform).

5.2.1 El nodo objetivo

El nodo objetivo tiene como principales responsabilidades obtener paquetes desde el nodo soporte (o desde algún medio físico de almacenamiento a nivel local), e instalar y configurar aplicaciones. En tal sentido, el nodo objetivo debe ser extendido y personalizado adecuadamente para satisfacer las necesidades de la plataforma objetivo.

Con motivos de obtener paquetes deployables desde el nodo soporte o servidor de paquetes, una opción de obtención podría basarse en una descarga desde un servidor FTP o desde un sitio web. Otro enfoque podría requerir la serialización/deserialización de los paquetes y ser obtenidos vía web services desde un servidor HTTP dedicado a tales fines. Sea cual sea la opción elegida, más allá de la complejidad que la misma implique, el framework abstrae la comunicación hacia el nodo soporte en el denominado **Node connector**, componente que debe ser implementado adecuadamente según los requerimientos del sistema de deployment en cuestión. Por otro lado, el mencionado componente tiene la responsabilidad de resolver el mecanismo de consulta desde el nodo objetivo al nodo soporte; tal es el caso de consultar cuáles paquetes están disponibles para descargar, y la ubicación de los mismos.

Con respecto a la instalación de aplicaciones, definida la plataforma de base, es posible personalizar el comportamiento del framework realizando una implementación propia del componente de abstracción de la plataforma objetivo, denominado simplemente **Platform** (plataforma). El componente Platform es el punto específico de interacción entre las funcionalidades del framework y la plataforma de base. Típicamente, se espera que en este lugar se coloquen las funcionalidades que permitan indicar el deployment de los componentes de una aplicación en el ambiente objetivo.

Siguiendo con la instalación, la configuración radica en el procesamiento de los archivos de configuración. Utilizando el Descriptor de aplicación se registra la información más básica y descriptiva de la aplicación a instalar, y se verifican las dependencias que dicha aplicación posea. Posteriormente, en caso de una verificación positiva, se procesa la Especificación de Deployment ejecutando las acciones especificadas; de esta manera, es posible indicar la realización de diversas tareas en función de los requerimientos de la plataforma de base. Luego de la ejecución de las acciones, se espera que el sitio objetivo resulte correctamente configurado, permitiendo que la nueva aplicación pueda ser utilizada. Finalmente, el restante archivo de configuración, la Especificación de Recursos, puede ser procesado mediante la inclusión de una acción responsable de realizar dicha tarea (ver 5.5). Lo descrito anteriormente es, ni más ni menos, una explicación en alto nivel del algoritmo de instalación de aplicaciones.

Es importante destacar la capacidad de extender el framework con nuevos tipos de dependencias, acciones y recursos en forma de plugins. El enfoque aplicado para la extensión permite la incorporación de nuevos elementos de la misma forma que se incorporan aplicaciones, de esa manera es posible agregar valor a la plataforma en caliente, es decir, sin reiniciar la ejecución de la misma. Mediante la incorporación de nuevas dependencias se agregan nuevas funcionalidades en materia de restricciones entre aplicaciones, incorporando nuevas acciones se amplia el abanico de tareas ejecutables durante una instalación, y finalmente, agregando nuevos tipos de recursos se extiende el mecanismo de interconexión entre aplicaciones.

5.2.2 El nodo soporte

El nodo soporte tiene como principales funcionalidades el registro de nuevos paquetes y la publicación de los mismos para que sean obtenidos desde nodos objetivo. El primero oficia de servidor mientras que los segundos ofician de clientes. Aunque en este ambiente no se realizan instalaciones, el registro de paquetes implica la realización de un control de dependencias entre los mismos verificando una mutua satisfacción de las mismas.

Por otro lado, existe una total libertad en la forma en que se puede realizar la publicación puesto que no es directamente controlada por el framework. El framework simplemente es

responsable de conocer la URL en dónde se encuentra un paquete expuesto. En el nodo objetivo, el Node connector el responsable de implementar la funcionalidad de transferencia, tomando como entrada la URL correspondiente a cada paquete requerido. En el nodo soporte se denomina **Node access** a aquel componente responsable de atender los pedidos realizados desde el nodo objetivo por intermedio del Node connector.

5.3 Gestión de paquetes

Un paquete deployable es el elemento resultante del primer paso del ciclo de vida de una aplicación, la liberación. Este elemento concentra información, ya sea descriptiva de la propia aplicación, o específica para el uso del framework de deloyment. Por otra parte, también es en este punto donde se incluyen los distintos componentes que conforman la misma.

Definido su contenido, un paquete no es más que un archivo contenedor de otros diversos archivos, cada uno con la información detallada anteriormente. La estructura de un paquete puede verse en la siguiente figura:

Fig. 5-4 – Estructura de un paquete deployable

Como puede apreciarse, el paquete presentado a modo de ejemplo, contiene los componentes de una aplicación J2EE, siendo uno de un componente Web (WAR) y los restantes simplemente módulos Java (JAR).

Dentro de la carpeta **DEPLOY-INF** se ubica la información descriptiva y aquella específica del framework:

deploy.xml	Especificación de Deployment.
descr.xml	Descriptor de Aplicación.
resources.xml	Especificación de Recursos

La Especificación de Deployment contiene una enumeración de acciones a tomar al momento de instalación y posteriormente al momento de desinstalación, el Descriptor de Aplicaciones contiene información descriptiva de la aplicación empaquetada, y finalmente, la Especificación de Recursos incluye aquellos recursos provistos y requeridos por la misma. De todos estos elementos, solo son imprescindibles el Descriptor de Aplicación y la Especificación de Deployment puesto que el primero es necesario para conocer cuál es la aplicación en consideración, mientras que la segunda es requerida para la instalación y posterior desinstalación. El contenido del Descriptor de Aplicación y su procesamiento serán desarrollados más adelante en esta sección; la Especificación de Deployment y su ejecución, en la sección 5.4; y tras ella, en la sección 5.5 le tocará el turno a la Especificación de Recursos y el registro de recursos provistos y requeridos.

5.3.1 Descriptor de aplicación

El contenido del descriptor de aplicación y su utilidad ya ha sido mayormente explicado en páginas anteriores; en esta sección se presenta más en detalle su contenido.

Dentro del descriptor se destacan tres niveles de datos descriptivos:

- **Información general**: datos de interés general como un identificador para la aplicación, su nombre, su autor y modo de contacto (correo electrónico), la fecha de empaquetado y una descripción.

```
01. <name>MiApp</ name>
02. <version>1.0.0</version>
03. <author>juan</author>
04. <email>juan@email.com</email>
05. <date>2004-12-16-10:00</date>
06. <description>Una breve descripción</description>
```

Fig. 5-5 – Información general en el Descriptor de Aplicación

- **Parametrizaciones**: información adicional no utilizada por el framework directamente pero que puede ser consultada y utilizada en diversas tareas. Supongamos que para una cierta plataforma objetivo se desee incluir una interfaz gráfica en donde se desee presentar al usuario la empresa que elaboró la aplicación y cuál es el país de origen. Dicha información puede ser incluida en el Descriptor mediante una parametrización tal como se muestra en la siguiente figura:

```
01. <parametrization type="MiParamz">
02.   <parameter name="enterprise">
03.       <simple type="string">MiEmpresa</simple>
04.   </parameter>
05.   <parameter name="country">
06.       <simple type="string">Uruguay</simple>
      </parameter>
    </parametrization>
```

Fig. 5-6 – Parametrizaciones en el Descriptor de Aplicación

- **Dependencias**: cualquier requerimiento o restricción establecida en relación a una aplicación. El caso más simple, e incorporado al framework por defecto, consiste en la especificación del requerimiento de existencia de una cierta versión de otra aplicación. Es posible incorporar nuevos tipos de dependencias al framework mediante sus mecanismos de extensión. Una dependencia simple, como la mencionada, es de la forma que puede apreciarse en la siguiente figura:

```
01. <dependency type="ApplicationVersion" name="dep01">
02.   <parameter name="app">
03.     <simple type="string">OtraAplicación</simple>
04.   </parameter>
05.   <parameter name="version">
06.     <simple type="string">1.0.0</simple>
07.   </parameter>
08. </dependency>
```

Fig. 5-6 – Dependencias en el Descriptor de Aplicación

5.3.2 Manejador de paquetes

El manejador de paquetes (Deployable management), es responsable de la gestión de paquetes. Existen dos casos de uso en donde su rol es fundamental: el registro de paquetes tanto en nodo soporte como en el nodo objetivo, y la transferencia de paquetes entre ambos nodos, más precisamente del nodo soporte oficiando de servidor de paquetes hacia el nodo objetivo que los solicita. Más allá de los casos mencionados, el manejador de paquetes es utilizado a lo largo y ancho de todo el framework para la realización de otros tipos de tareas. En tal sentido, su presencia será frecuentemente destacada en otras actividades detalladas más adelante.

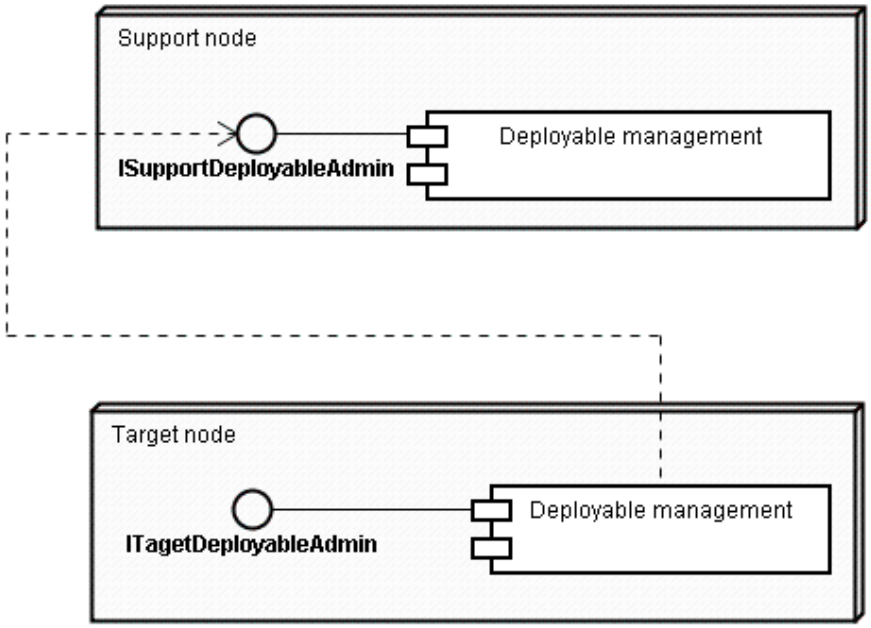


Fig. 5-7 – Arquitectura del manejador de paquetes deployables

La arquitectura interna del manejador de paquetes es distribuida entre los nodos. En el nodo soporte, es responsable del registro de nuevos paquetes para ser expuestos a todo nodo objetivo cliente, y de atender consultas desde dichos nodos. En el nodo objetivo, es responsable de la obtención y registro de paquetes, quedando prontos para ser instalados. Conviene destacar el hecho de que la transferencia se realiza en el nodo objetivo obteniendo los paquetes desde la ubicación provista por el nodo soporte (URL).

El registro de paquetes consiste simplemente en incorporar los paquetes a un repositorio local. Ésta tarea es realizada mediante un componente denominado **Repositorio de paquetes**, el cual adicionalmente permite la realización de consultas que permitan conocer su contenido, o la eliminación de paquetes.

Con respecto a la transferencia de paquetes, en la siguiente figura se presenta el protocolo de comunicación entre los nodos para resolver la misma:

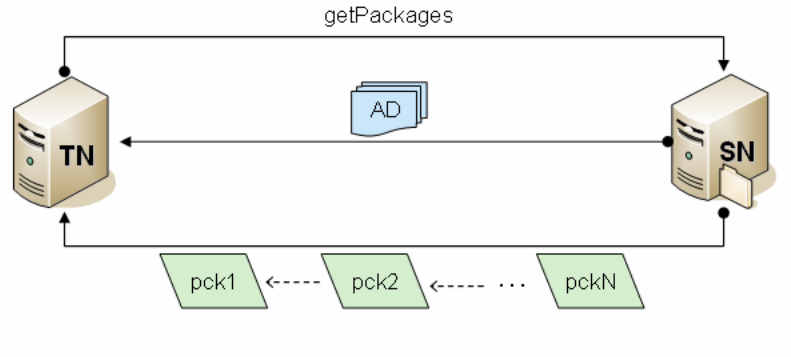


Fig. 5-8 – Transferencia de paquetes / Interacción inter-nodo

En primer lugar, el nodo objetivo (TN) envía una solicitud del listado de paquetes que el nodo soporte (SN) expone. Posteriormente, el nodo soporte responde retornando el listado solicitado. El listado es recibido por el nodo objetivo y a continuación es procesado obteniendo la URL para la realización de la descarga (transferencia). Determinados los paquetes a ser transferidos, el nodo objetivo inicia la descarga de los paquetes basándose en las respectivas URL suministradas.

El framework permite realizar una variante del esquema anterior. La misma consiste en la inclusión de otro tipo de nodo intermedio, o nodo mirror (espejo), el cual puede exponer e instalar paquetes[2]. Este tipo de nodo no será visto en profundidad en este capítulo, pero en el capítulo 6 se volverá tocar este punto.

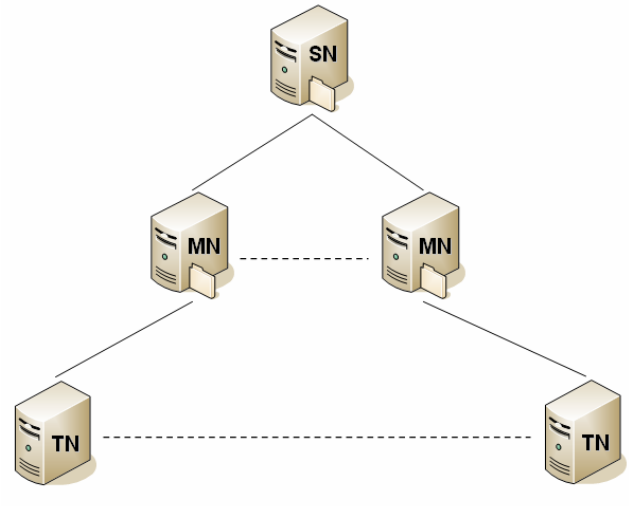


Fig 5-9 – Arquitectura con nodos mirror

Un nodo mirror (MN) obtiene paquetes del nodo soporte (supuestamente único), o desde otro nodo mirror. Luego expone los mismos para ser transferidos. Los paquetes expuestos pueden ser obtenidos desde otro nodo mirror, o bien, desde un nodo objetivo.

[2] Un nodo mirror expone paquetes por lo que se le considera un nodo soporte (variante de). Por otro lado, también es posible realizar instalaciones por lo que se le considera un nodo objetivo. Por tanto, toda característica de dichos nodos se asumen presentes también en los nodos mirror.

Con la incorporación de este nuevo elemento se pretende por un lado alivianar el trabajo del nodo soporte, y por el otro, aumentar la disponibilidad de servidores de paquetes debido a la mayor presencia de nodos con esa capacidad. Un nodo objetivo es configurado de acuerdo a la topología elegida, e incluso puede modificarse el nodo servidor de paquetes a utilizar en caso que el actualmente configurado no esté disponible y se conozca otro que sí se encuentra listo para atender nodos clientes.

5.4 Instalación

Instalar nuevos paquetes en un cierto nodo objetivo es una tarea compleja principalmente en lo vinculado con la actividad de configuración. En este punto se presentan dos opciones.

Por un lado es posible considerar una serie de restricciones sobre las aplicaciones a instalar, las cuales permitan que las mismas se incorporen a la plataforma objetivo en base a un mecanismo de configuración estático e invariante a lo largo de la vida de la plataforma mencionada. Éste criterio implica una importante pérdida en materia de flexibilidad debido a que una aplicación se configuraría siempre de la misma manera, mecanismo no apto en caso de que las aplicaciones sean de naturaleza distinta, o bien la plataforma requiera diversos mecanismos de configuración en función del tipo de aplicación a instalar.

La otra alternativa, más flexible y dinámica, implica la especificación de aquellas tareas que se desean ejecutar al momento de realizarse una instalación. De esta manera, definido un kit de tareas disponibles, un desarrollador puede indicar cómo se incorpora la aplicación a la plataforma objetivo en base a la inclusión, en una Especificación de Deployment, de aquellas tareas que considere apropiadas para resolver sus necesidades. Un motor de deployment procesa las tareas en el orden indicado, derivando en la correcta instalación de la aplicación.

La segunda opción es la elegida para el framework, y las tareas se corresponden con el concepto de **Acción (action)**. En la siguiente sección se presenta la Especificación de Deployment. Posteriormente, en la sección 5.4.2 se incluye una descripción del componente responsable de realizar la instalación de una aplicación, y de cómo se utiliza la mencionada Especificación para la instanciación y ejecución de las acciones especificadas.

5.4.1 Especificación de Deployment

La Especificación de Deployment consiste en una secuencia de acciones a tomar. Se dividen en dos grupos: acciones de deployment (**deployment-actions**) y acciones de undeployment (**undeployment-actions**). Las primeras conforman las acciones a ejecutar al realizarse la instalación, mientras que las segundas son aquellas a ejecutar al realizarse la desinstalación.

```
01. <deployment-actions>
02.   <action type="JARInstaller" name="act01">
03.     <parameter name="files">
04.       <complex type="collection">
05.         <item>
06.           <simple type="string">libs.jar</simple>
07.         </item>
08.         <item>
09.           <simple type="string">ejbs.jar</simple>
10.         </item>
11.       </complex>
12.     </parameter>
13.   </action>
14.   <action type="WARInstaller" name="act02">
15.     <parameter name="files">
16.       <complex type="collection">
17.         <item>
18.           <simple type="string">ui.war</simple>
19.         </item>
20.       </complex>
21.     </parameter>
22.   </action>
23. </deployment-actions>
```

Fig. 5-10 – Acciones de deployment en la Especificación de Deployment

En la figura anterior, se aprecia un fragmento de una Especificación de Deployment, conformando las acciones de instalación (deployment). En el ejemplo se consideran dos acciones denominadas **JARInstaller** y **WARInstaller**, las cuales reciben un conjunto de rutas correspondientes a archivos incluidos en un cierto paquete. De esta manera, se parametrizan dichas acciones para que éstas tomen las rutas, accedan a los archivos indicados y posteriormente procedan a instalarlos en la plataforma objetivo, en este caso un servidor basado en J2EE.

Existen tantas acciones como los desarrolladores que utilicen el framework deseen implementar. Más aún, el mismo permite incorporar nuevos tipos de acciones mediante un mecanismo de extensión basado en plugins. En la sección 5.6 se incluye una descripción más en detalle de este mecanismo así como el manejo en general de los diversos tipos de plugins.

5.4.2 Deployment

Este componente oficia de fachada para todas las tareas vinculadas a la instalación y desinstalación de aplicaciones, por lo tanto, es por medio del **Deployment** que se da inicio a la instalación de una aplicación.

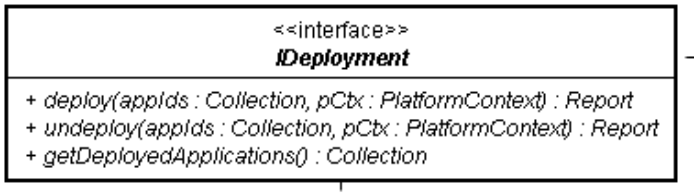


Fig. 5-11 – Interfaz del componente Deployment

El Deployment actúa sobre un conjunto de aplicaciones indicadas mediante un identificador único. En relación a la instalación (operación **deploy**), este componente se encarga de realizar las siguientes tareas:

1. Para cada aplicación:
   1.1. Cargar el Descriptor de Aplicación.
   1.2. Cargar la Especificación de Deployment.
2. En base a los Descriptores de Aplicación, controlar la mutua satisfacción de dependencias.
3. Para cada aplicación:
   3.1. Indicar le deployment por medio del **Deployer**.

Como puede apreciarse, en el anterior algoritmo se destaca el control de dependencias y la instalación delegada al componente denominado **Deployer**. El control de dependencias verifica que todas las aplicaciones a instalar puedan ser incorporadas a la plataforma objetivo en función de la mutua satisfacción de las mismas.

Fig. 5-12 – Interfaz del componente Deployer

Las acciones indicadas es la Especificación de Deployment son instanciadas y ejecutadas mediante el motor de deployment incluido en el framework. Dicho motor reside en el componente denominado **Deployer**. Este componente realiza la instanciación de las acciones de deployment especificadas, y luego precede a ejecutarlas en la secuencia en que fueron declaradas en la Especificación de deployment. Cada acción es inicializada con los parámetros establecidos, los cuales son verificados previa ejecución de la misma.

De esta manera, se cumple con los objetivos establecidos para la actividad de instalación.

5.5 Gestión de recursos

Un recurso es cualquier elemento provisto por una aplicación y que eventualmente puede ser requerido por otras. Una aplicación que desea exponer y/o utilizar un recurso, necesita notificar al sistema a través de un archivo de especificación de recursos. Este archivo debe incluirse en el paquete deployable de la aplicación como se vio en la sección 5.3.

Los recursos tienen una propiedad importante que es su tipo, esto los diferencia entre si y permite que se especifiquen recursos con capacidades distintas. Esto surge de la necesidad de las aplicaciones de utilizar diversos tipos de elementos provistos por otras aplicaciones para poder cumplir con sus cometidos. En una plataforma J2EE, un recurso puede ser de tipo Session Bean, Servlet, Web Service o cualquier otro tipo que se defina e implemente utilizando los mecanismos de extensión provistos por el sistema. De esta forma, los administradores tienen la posibilidad de desarrollar sus propios recursos y añadirlos al sistema sin mayor complejidad.

Para lograr que las aplicaciones utilicen recursos provistos por otras aplicaciones se cuenta con el componente **Manejador de Recursos**. Este es el componente responsable de la gestión de recursos. Sus principales funcionalidades son: administrar los recursos y proveer mecanismos de enlace entre las aplicaciones.

Administrar recursos implica gestionar los recursos registrados por las aplicaciones al momento de deployment. La funcionalidad de enlace entre aplicaciones permite a las mismas utilizar recursos expuestos por otras en tiempo de ejecución y sin necesidad de que éstas sepan dónde están ubicados, ni cómo se obtienen. Lo único que deben saber las aplicaciones es como utilizar el manejador de recursos y este les proveerá del elemento que solicitan.

5.5.1 Especificación de Recursos

El archivo de especificación de recursos es en el que se definen los recursos provistos y requeridos por una aplicación.

Para especificar un recurso provisto se debe declarar un **nombre** para el recurso, un **tipo** y luego una serie de **parámetros** que serán utilizados al momento de instanciar el recurso por el manejador correspondiente. Éste es el único que conoce cuáles son y cómo se usan los parámetros definidos. El manejador de recursos registra este recurso y cuando es solicitado sabe cómo, a partir de la especificación, entregar al solicitante el elemento ya instanciado o la información necesaria para poder instanciarlo.

En la figura 5-13 se muestra un ejemplo de una aplicación que provee un recurso llamado “ResourceAPI” que es de tipo “SessionBean” y tiene un parámetro llamado “jndiLocalHome” que tiene valor de la clave JNDI mediante la cual es posible acceder al componente EJB, en este caso, “ejb/PgDploy1/ResourceLocal”.

```
01. <provided-resources>
02.    <provided name=”ResourceAPI” type=”SessionBean”>
03.       <parameter name=”jndiLocalHome”>
04.          <simple type=”string”>
05.             ejb/PgDploy1/ResourceLocal
06.          </simple>
07.       </parameter>
08.    </provided>
09. </provided-resources>
```

Fig. 5-13 – Recurso expuesto en la especificación de recursos

En este mismo archivo son especificados los recursos requeridos por la aplicación. Para solicitar un recurso es necesario especificar su **nombre**, su **tipo** y la **aplicación** que lo provee. Estos parámetros son requeridos para que el Manejador de Recursos pueda ubicar el recurso

correctamente. Además de estos parámetros, también pueden ser especificados otros dos, estos son el **alias** y la **vista**.

El alias es el nombre dado al elemento, mediante el cual se referenciará al mismo al momento de realzarse la solicitud. En caso de no especificarse un alias, se le asume igual al nombre del recurso.

La vista indica la forma en que la aplicación solicitante espera recibir el recurso. Una vista se corresponde con una funcionalidad del tipo de recurso mediante la cual se indica cómo éste debe proveer el enlace con el mismo. Volviendo al ejemplo del tipo “SessionBean”, este puede proveer tres vistas: vista por defecto, vista “EJBHome” y vista “EJBLocalHome”. La vista por defecto siempre se corresponde con un objeto que encapsula toda la información necesaria para que la propia aplicación realice el enlace, es decir, los parámetros especificados en la declaración del recurso provisto. Especificando la vista “EJBlHome” se efectúa el trabajo adicional de localizar la Home Interface del Session Bean indicado, vía JNDI, y retornar el objeto instanciado. Especificando la vista “EJBLocalHome”, se procede de la misma manera pero a nivel local.

```
01. <required-resources>
02.    <required name="ResourceAPI" app="BasicResources"
03.              type="SessionBean" alias="basics"
04.              view="EJBLocalHome"/>
05. </required-resources>
```

Fig. 5-14 – Recurso requerido en la especificación de recursos

En la figura 5-14 se muestra un ejemplo de una aplicación que requiere un recurso de tipo “SessionBean”. El recurso se corresponde con aquel provisto por la aplicación “BasicResources” cuyo nombre es "ResourceAPI". Dicho recurso será referenciado por su alias cuyo valor indicado será "basics". Se indica una vista “EJBLocalHome”, por lo que se espera que el Manejador de Recursos realice la instanciación de la Local Home Interface del Session Bean. En la siguiente sección se verá cómo utilizar el Manejador para acceder a los recursos requeridos.

5.5.2 Enlace entre aplicaciones

Un enlace se produce en tiempo de ejecución cuando la aplicación solicitante utiliza el Manejador de Recursos para lograr acceder a un recurso especificado como requerido. El manejador utiliza tres elementos para realizar el enlace: la especificación de recurso requerido, la especificación del recurso provisto, y el manejador correspondiente al tipo de recurso a enlazar.

```
01.  ResourceManager manager = . . .
02.  ResourceAccessSolver solver = manager.getResourceAccess(“MiApp”);
03.  MiBeanLocalHome recursoHome;
04.  recursoHome = (MiBeanLocalHome) manejador.getResource(“basics”);
05.  MiBeanLocal bean = recursoHome.getMiBeanLocal();
06.  // Utilizar las funcionalidades del objeto bean
```

Fig. 5-15 – Seudo código de la solicitud de un recurso (ver Fig. 5-13) por parte de una aplicación

En la figura 5-15 se muestra cómo utilizar el Manejador de Recursos para acceder a un recurso requerido, en este caso el recurso “ResourceAPI” especificado en la figura 5-14, correspondiente al recurso provisto especificado en la figura 5-13. A partir del Manejador de Recursos (**ResourceManager)**, pueden solicitarse los recursos requeridos a través de una llamada a la función **getResourceAccess** (línea 02) cuyo parámetro es simplemente el identificador de la aplicación solicitante. Como resultado, se obtiene un objeto **ResourceAccessSolver** (solver) mediante el cuál es posible realizar una solicitud puntual de cada recurso indicado como requerido (línea 04). En el caso del ejemplo, la vista indicada es “EJBLocalHome”, por lo que la Local Home Interface es obtenida de manera directa.

Luego de obtener la referencia al elemento requerido, la aplicación puede utilizarlo como desee. De esta forma, se muestra que lo único que se necesita para utilizar un recurso expuesto por otra aplicación es especificarlo como requerido, y utilizar adecuadamente al Manejador de Recursos.

Si al momento de especificar un recurso requerido no se especifica ninguna vista, se entregaran los elementos necesarios para que la aplicación solicitante pueda obtener el recurso por si mismo. Las vistas dependen del tipo de recurso que se esta especificando y como este sea implementado por el manejador correspondiente.

Siguiendo con el ejemplo ahora se mostrará cual sería el procedimiento a seguir en el caso de no definirse ninguna vista. Supongamos que en la figura 5-14 no se especifico una vista, el Manejador de Recursos se debería utilizar como muestra la figura 5-16:

```
01.  ResourceManager manager = . . .
02.  ResourceAccessSolver solver = manager.getResourceAccess(“MiApp”);
03.  Map params = (Map) manejador.getResource(“basics”);
04.  String jndiKey = params.get(“jndiLocalHome”);
05.  MiBeanLocalHome recursoHome = jndiGetLocalHome(jndiKey);
06.  MiBeanLocal bean = recursoHome.getMiBeanLocal();
07.  // Utilizar las funcionalidades del objeto bean
```

Fig. 5-16 – Seudo código de la solicitud de un recurso (ver Fig. 5-13) por parte de una aplicación sin utilizar vistas

Cuando no se indica una vista, se espera que por medio del Manejador se obtenga un objeto mediante el cual obtener la información necesaria para que la propia aplicación realice el enlace. En este caso, en la línea 03 se obtiene dicho objeto, en la línea 04 se obtiene la clave JNDI que finalmente es utilizada en la línea 05 para realizar el enlace.

La principal ventaja de utilizar el manejador de recursos radica en que existe la posibilidad de reconfigurar los recursos sin necesidad de modificar la aplicación que los utiliza. De esta forma una aplicación puede cambiar las características de los recursos que provee sin afectar a las aplicaciones que los utilizan.

Otra ventaja radica en la ausencia de necesidad de saber cómo ubicar un recurso provisto por otra aplicación, simplemente se necesita saber cómo ubicar el manejador de recursos. Si solo se utiliza un recurso de otra aplicación la ventaja no es muy evidente, pero si los componentes que se necesitan son varios, resulta más sencillo y cómodo acceder a un solo componente capaz de resolver los enlaces requeridos, que tener que codificar el acceso a cada recurso sabiendo de antemano cómo obtenerlos.

5.6 Extensibilidad

En el capítulo 2 se mencionó como uno de los objetivos el proveer mecanismos de extensión que permitan una fácil personalización del producto. Alineado con dicho objetivo, el framework permite la inclusión de nuevas funcionalidades en la forma de plugins.

Los desarrolladores que utilicen los servicios de framework deberán personalizarlo incluyendo una serie de extensiones que brinden las funcionalidades básicas de la plataforma objetivo en términos de deployment. Posteriormente, desarrollando nuevas extensiones podrán incorporar nuevas funcionalidades conforme la plataforma objetivo vaya evolucionando en el tiempo.

Supongamos que la plataforma objetivo consiste en una serie de nodos basados en servidores J2EE. Sus administradores deciden que se implementen dos acciones sobre el framework. La primera permite instalar Enterprise Applications (EAR), mientras que la segunda acción es responsable de instalar Web Applications (WAR). Dichas acciones conforman entonces las acciones básicas a realizar al momento de deployment y son incluidas al sistema mediante configuración directa del framework.

Ahora supongamos que la plataforma objetivo es puesta en producción y luego de un cierto tiempo los administradores consideran importante incluir una funcionalidad que permita instalar un Web Service a partir de una especificación del mismo. Debido a que cada nodo ya se encuentra en producción, los administradores desean que la extensión se exponga en un servidor de paquetes (nodo soporte) y que ésta pueda ser descargada desde cada nodo objetivo para su posterior instalación. Basándose nuevamente en los servicios del framework, es posible desarrollar una acción de deployment responsable de realizar la tarea requerida, empaquetarla del mismo modo que una aplicación común y exponerla en un nodo soporte. Luego, desde un nodo objetivo, es posible realizar la transferencia e instalación de la misma quedando incorporada la nueva acción.

Cada plugin consiste en la suma de implementación de componentes basados en las interfaces provistas por el framework, más una especificación de los mismos, incluyendo datos descriptivos que permitan al sistema registrar los nuevos elementos incluidos.

Volviendo al ejemplo presentado anteriormente, los administradores del sistema basado en J2EE deciden implementar una nueva acción denominada “WSInstaller”, o Web Service Installer. Dicha acción recibe una especificación del Web Service y procede a registrarlo en el servidor de aplicaciones.

Una vez finalizada la implementación, se empaqueta la extensión como una aplicación, siguiendo un formato análogo al presentado en la Fig. 5-3 visto en la sección 5.3. Debido a que el plugin es una aplicación común, toda información descriptiva puede ser incluida en el Descriptor de Aplicación, y todo recurso requerido (o provisto) puede ser declarado en la Especificación de Recursos. Para registrar la nueva acción, en la Especificación de Deployment se debe incluir una acción de deployment específica para el registro de este tipo de plugin; dicha acción se denomina **PlugActionType**.

En la figura 5-17 se aprecia un fragmento de la Especificación de Deployment en donde se incluye la acción PlugActionType. Como puede apreciarse, la acción requiere una serie de parámetros para poder registrar correctamente la extensión; tal es el caso del nombre y la clase Java que implementa la interfaz de las acciones.

```
01. <action name="dAct1" type="PlugActionType">
02.   <parameter name="type">
03.     <simple type="string">
04.       WSInstaller
05.     </simple>
06.   </parameter>
07.   <parameter name="className">
08.     <simple type="string">
09.       myplatform.plugins.WSInstaller
10.     </simple>
11.   </parameter>
12. </action>
```

Fig. 5-17 – Ejemplo de registro de acciones usando PlugActionType

El mecanismo presentado permite además agregar otros tipos de elementos al sistema. Es posible incorporar plugins para los siguientes elementos:

- Tipos de acciones.
- Tipos de dependencias.
- Tipos de recursos.

Las acciones incorporadas al framework para registrar plugins se denominan **PlugActionType**, **PlugDependencyType**, **PlugResourceType** respectivamente, y sus parámetros, para todos los casos, son los siguientes:

type	Tipo del elemento. Se corresponde al nombre asociado al mismo.
className	Clase Java que implementa la interfaz correspondiente al elemento a registrar.

Volviendo al ejemplo presentado, una vez registrada la acción “WSInstaller”, la misma puede ser utilizada de la siguiente manera:

```
01. <action name="dAct1" type="WSInstaller">
02.   <parameter name="spec">
03.     <simple type="string">
04.       Specs/webservice.wsdl
05.     </simple>
06.   </parameter>
07. </action>
```

Fig. 5-18 – Referencia a la acción WSInsaller registrada como plugin

Como puede apreciarse, la acción es referenciada con el nombre del tipo (type) establecido al utilizar PlugActionType. A la acción se le pasa como parámetro una especificación del Web Service en base a la ruta relativa a la raíz del paquete del plugin.

La principal virtud del mecanismo propuesto es la capacidad de incorporar nuevas funcionalidades de una manera elegante y homogénea, es decir, se incorporan al sistema del mismo modo que se incorpora una nueva aplicación. La reconfiguración del sistema se realiza automáticamente por intermedio de las acciones mencionadas, quedando las nuevas funcionalidades inmediatamente disponibles luego de realizado el deployment.

Se incluye una explicación más detallada del mecanismo de extensión mediante plugins en el Anexo E.

6. Aplicación sobre la plataforma LinkAll

Ya se han presentado los principales conceptos, componentes y funcionalidades del framework, ahora es momento de explicar cómo el mismo puede ser personalizado para la utilización en una plataforma objetivo concreta: la plataforma LinkAll.

Explicar en detalle las características propias de la plataforma LinkAll no compete al presente informe, sin embargo, puede ser necesario aclarar algunos conceptos referentes a la mencionada plataforma objetivo para entender cómo el framework es aplicable para la elaboración de su componente de deployment. En resumen, durante el transcurso de las siguientes líneas, se detallará la aplicación del framework para la construcción de su componente de deployment, explicando brevemente algunos conceptos de la plataforma LinkAll, en caso de ser necesario.

6.1 El framework dentro de la arquitectura LinkAll

La plataforma LinkAll está compuesta por una serie de nodos que conforman una red colaborativa de datos y servicios. Los nodos de la red no tienen alguna jerarquía o distinción entre sí, a excepción del denominado nodo soporte, el cual ofrece servicios especiales a los restantes nodos en la red.

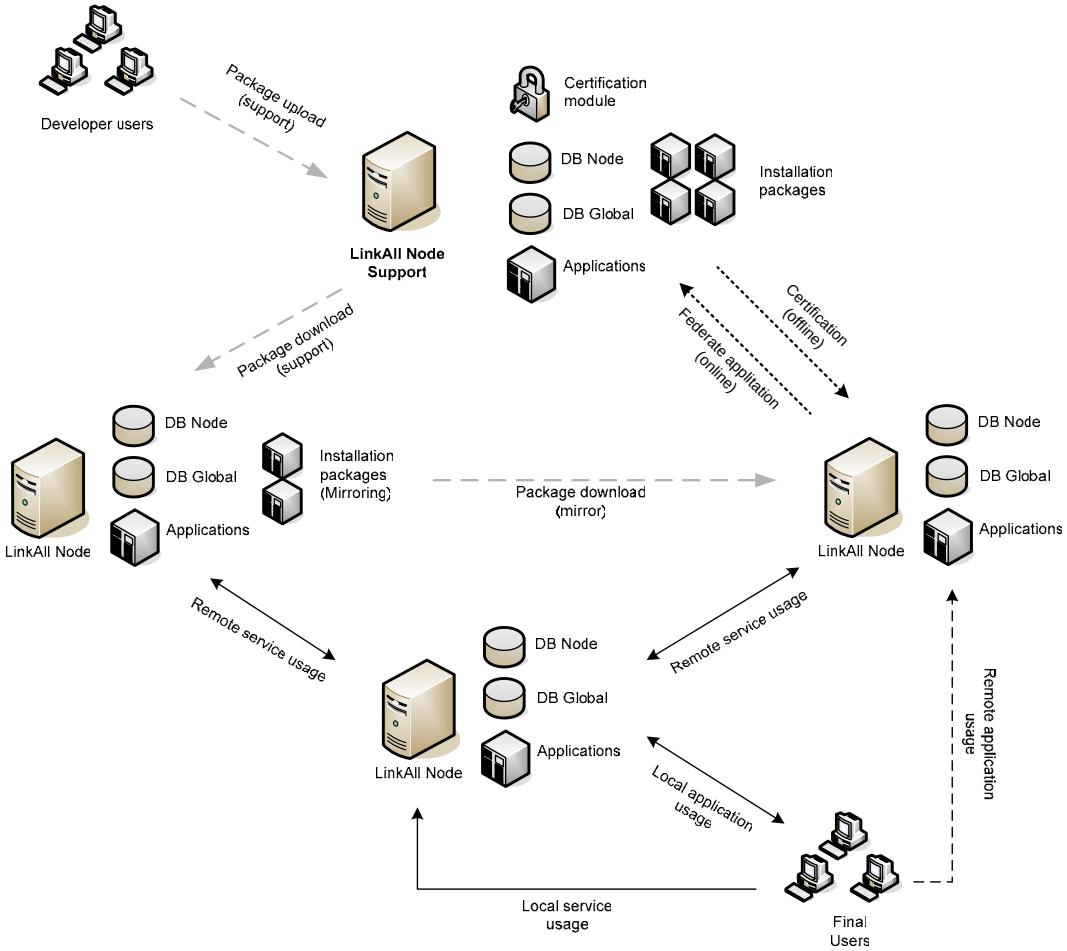


Fig. 6-1 – Red LinkAll

Dentro de este contexto, el framework es utilizado en ambos tipos de nodos: nodo soporte y nodo común, o aplicativo. En el nodo soporte brinda la infraestructura sobre la cual se exponen paquetes deployables, mientras que en los nodos aplicativos sirve de base para la implementación de las restantes actividades del ciclo de vida de deployment. Se distingue el

caso particular de aquellos nodos que ofrecen servicios de mirroring, en donde se exponen paquetes y se instalan aplicaciones. Para este caso, los nodos cuentan con funcionalidades similares a las de un nodo soporte, más todas las funcionalidades de un nodo aplicativo.

A continuación, se listan las más relevantes funcionalidades incluidas en cada tipo de nodo:

- **Nodo soporte**
  - Registro de paquetes.
  - Firmado de paquetes.
  - Exposición de paquetes.

- **Nodo aplicativo**
  - Descarga de paquetes.
  - Instalación de paquetes.
  - Gestión de recursos.

- **Nodo aplicativo (mirroring)**
  - Registro de paquetes (firmados por el nodo soporte).
  - Exposición de paquetes.
  - Descarga de paquetes.
  - Instalación de paquetes.
  - Gestión de recursos.

En la siguiente sección se verán estos puntos más en detalle, sin embargo, es importante realizar, previamente, algunas puntualizaciones con respecto a las funcionalidades de los nodos de mirroring. Este tipo de nodo, a grandes rasgos, es un nodo híbrido de un nodo soporte y un nodo aplicativo, brindando las funcionalidades asociadas a ambos nodos. La anterior afirmación es cierta pero no del todo correcta. Existe un servicio que solo es brindado por el nodo soporte; dicho servicio es el de firmado de paquetes.

Los paquetes son firmados mediante un sistema de cifrado de clave pública. El nodo soporte es el único nodo que posee la clave privada utilizada para firmar los paquetes, por tanto es el único capacitado para realizar dicha tarea. Los restantes nodos poseen la clave pública contra la cual certifican que los paquetes descargados sean auténticos.

Realizada la puntualización, en la siguiente sección se hará distinción simplemente entre nodo soporte y nodo aplicativo, asumiendo que ha quedado claro el hecho de que los nodos de mirroring se componen de la suma de las funcionalidades de ambos tipos de nodo, pero teniendo en consideración la aclaración realizada anteriormente.

6.2 Interacción con la plataforma

La plataforma LinkAll cuenta con una serie de componentes mediante los cuales es posible realizar diversas tareas. Algunos de ellos son frecuentemente utilizados por las aplicaciones instaladas, mientras que otros proveen servicios a componentes con funcionalidades de más alto nivel. Alineado con esa arquitectura, el componente de deployment se ubica en la plataforma como un componente que brinda servicios y que no interactúa directamente con las aplicaciones, salvo su Manejador de Recursos interno, el cual sí interactúa con aplicaciones que deseen resolver el acceso a algún recurso provisto por otra aplicación.

En la siguiente figura aparecen representados los principales componentes de un nodo aplicativo de la plataforma. En su interior puede apreciarse al componente de deployment como parte de los servicios de backend.

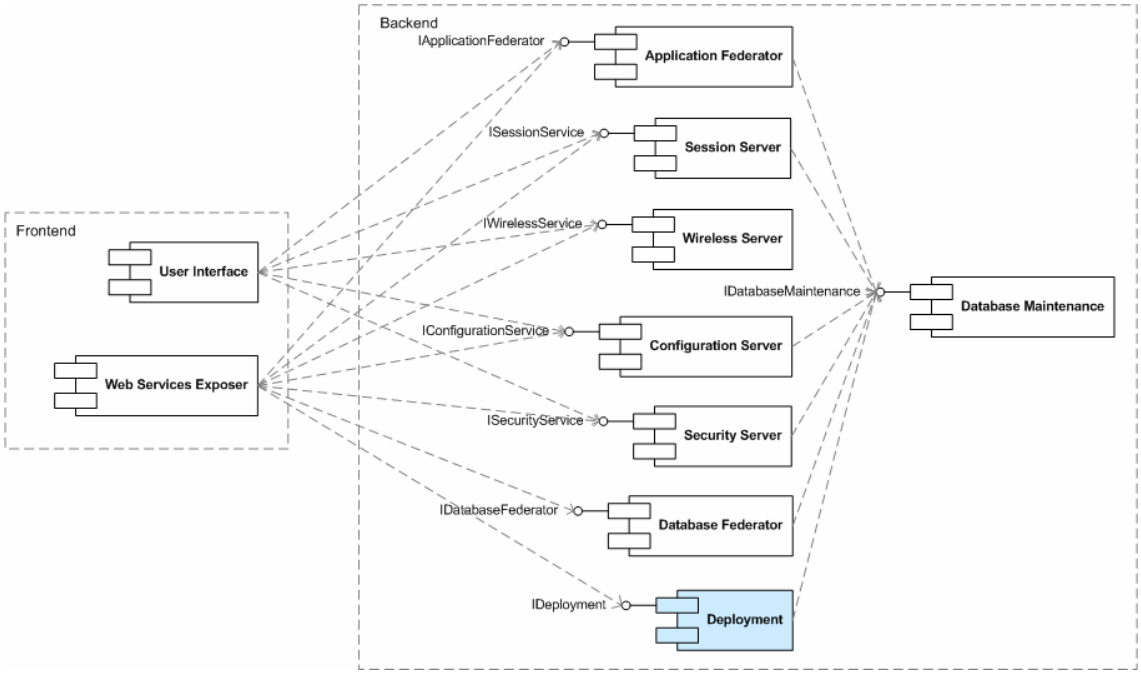


Fig. 6-2 – El componente de deployment dentro de la plataforma LinkAll

La organización interna del componente de deployment es en capas. Por un lado, resolviendo los servicios elementales, se encuentra el framework de deployment. Sobre él, una capa de más alto nivel brinda las funcionalidades específicas requeridas por la plataforma LinkAll, siguiendo sus lineamientos en materia de diseño e implementación.

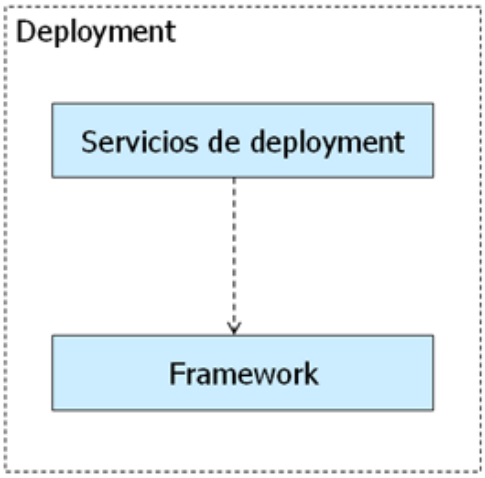


Fig. 6-3 – Estructura interna del componente de deployment

Los servicios de deployment para LinkAll, como se aprecia en la figura 6-3, y respetando el esquema propuesto por una organización en capas, se beneficia de los servicios del framework. En las siguientes secciones se analizará la personalización realizada para la elaboración del componente de deployment en función de los tipos de extensiones incorporables.

6.2.1 Representación de la plataforma

En la sección 5.2.1 se introdujo este concepto, el cual encapsula toda funcionalidad específica de la plataforma que pueda ser utilizada por alguna acción de deployment. La plataforma no es más que una clase que implementa la interfaz **IPlatform** y que provee un objeto **PlatformContext** que eventualmente puede contener información contextual.

Para el caso de la plataforma LinkAll, el componente de deployment posee una representación de la plataforma denominada **LinkAllPlatform** y su correspondiente contexto **LinkAllPlatformContext**. LinkAll se sustenta en los servicios de un servidor de aplicaciones J2EE denominado Jboss, debido a esto, la representación LinkAllPlatform posee operaciones para realizar instalaciones, y desinstalaciones, de componentes y aplicaciones dentro del servidor mencionado. Tomando en consideración la reusabilidad, la representación LinkAllPlatform se basa en una representación de la plataforma Jboss denominada **Jboss3_2_XPlatform** (correspondiente a la versión utilizada del servidor Jboss, la versión 3.2.x).

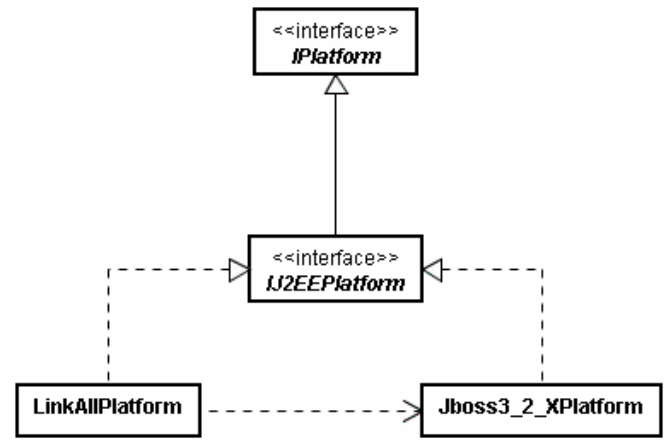


Fig. 6-4 – Diseño de la representación de la plataforma LinkAll

La representación LinkAllPlatform no solo delega el trabajo a su símil Jboss3_2_XPlatform, sino que incorpora otros elementos vinculados específicamente a la plataforma LinkAll. Tal es el caso del manejo de información contextual utilizada para el control de acceso a las funcionalidades de la plataforma en base a un esquema de permisos definido en la configuración de la misma.

Una aplicación LinkAll puede estar constituida por diversos componentes J2EE (WAR, JAR, EAR, etc.). Las acciones de deployment responsables de la instalación y desinstalación de dichos componentes, utilizan la clase LinkAllPlatform para tales fines. Este mecanismo es explicado más en detalle en la siguiente sección.

6.2.2 Acciones de deployment

Como parte del trabajo de análisis, se detectaron distintos tipos de componentes instalables dentro del servidor J2EE, los cuales se agrupan formando una aplicación LinkAll. Cada tipo de componente posee asociada una acción responsable de realizar la instalación de los mismos. En la siguiente tabla se listan los distintos tipos de componentes y sus correspondientes acciones:

Componente J2EE	Acción asociada
Enterprise application (EAR)	EARInstaller
Web application (WAR)	WARInstaller
Java module (JAR)	JARInstaller
EJB module (JAR)	JARInstaller
Jboss Service (SAR)	SARInstaller

Cada una de las acciones que aparecen en la tabla, delega a la representación de la plataforma LinkAll, la tarea de instalación de los componentes asociados, tal como se aprecia en la figura 6-5.

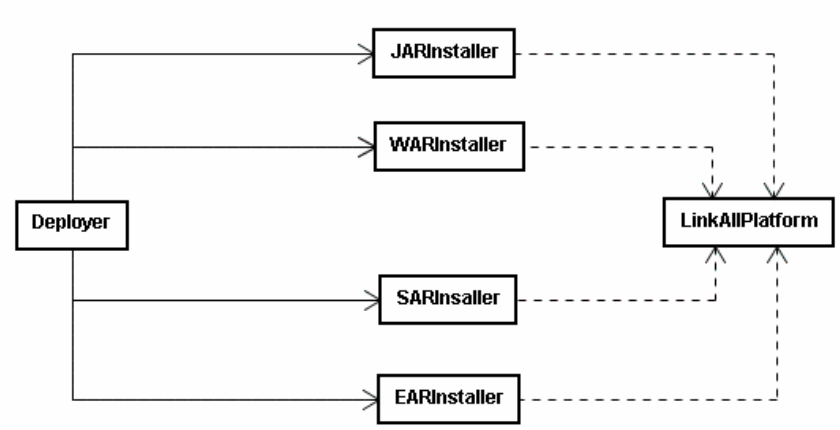


Fig. 6-5 – Interacción de las acciones de instalación de componentes con la plataforma LinkAll

El mecanismo de extensión soportado por el framework permite eventualmente extender el abanico de acciones básicas, tan solo realizando un plugin y exponiendo el mismo como una aplicación, ya sea en el nodo soporte o en un nodo oficiando de mirror.

6.2.3 Conexión entre nodos

En la sección 5.2.1, dentro de las características de un nodo objetivo, se mencionó al componente denominado **Node Connector** (conector), responsable de encapsular la comunicación hacia un nodo soporte, o eventualmente hacia un nodo mirror. En general, resuelve la conexión hacia un nodo servidor de paquetes.

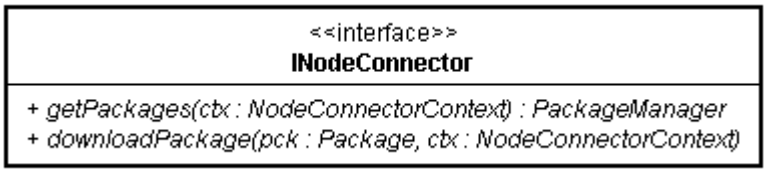


Fig. 6-6 – Interfaz del Node connector

Este componente debe ser implementado acorde a las características propias de la plataforma objetivo. La interfaz de los conectores define dos operaciones correspondientes a dos servicios elementales. En primer lugar, una operación de consulta al nodo servidor de paquetes acerca de cuáles son los paquetes disponibles. En segunda instancia, una operación de obtención de paquetes desde el lugar indicado por el servidor.

Para el caso de la plataforma LinkAll, el componente de deployment contiene una implementación de un conector denominada **LinkAllSupportNodeConnector**, el cual resuelve las operaciones de la siguiente forma:

- **Consulta de paquetes disponibles**: esta funcionalidad se resuelve mediante una consulta a un web service utilizando SOAP sobre HTTP. Todo nodo servidor de paquetes puede ser consultado por medio de este mecanismo utilizando como punto de conexión una dirección HTTP configurable.

- **Obtención de paquetes**: dentro de los datos retornados por un servidor de paquetes al consultar cuáles son los paquetes que expone, se incluye la URL correspondiente a cada uno de los paquetes. En base a esta información, la operación de obtención de paquetes los descarga desde la dirección indicada.

6.2.4 Tipos de recursos

Con respecto a los recursos, fueron identificados diferentes tipos, los cuales pueden ser provistos y requeridos por una aplicación LinkAll. Cada tipo de recurso cuenta con un manejador que lo interpreta y procesa. A continuación se listan los tipos de recursos existentes para distintos objetos J2EE.

Objeto J2EE	Tipo de recurso
Session Bean	SessionBean
Web Service	WebService
Interfaz Web	WebInterface

A continuación se presentarán los parámetros definidos para cada uno de estos recursos y para que son utilizados:

Session Bean

Nombre	Tipo	Descripción
`jndiHome`	`String`	Clave JNDI utilizada para obtener la Home Interface del Session Bean.
`jndiLocalHome`	`String`	Clave JNDI utilizada para obtener la Local Home Interface del Session Bean.

Web Service

Nombre	Tipo	Descripción
`uri`	`String`	URI utilizada para acceder al Web Service.

Interfaz Web

Nombre	Tipo	Descripción
`uri`	`String`	URI utilizada para acceder a la interfaz Web.

En lo que respecta a las vistas, actualmente el sistema provee de vistas solamente para los recursos de tipo SessioBean. Estas son “EJBHome” y “EJBLocalHome”, con las cuales se obtienen las interfaces Home y Local Home del Session Bean como se vio en **¡Error! No se encuentra el origen de la referencia.**.

Si se requiere extender los tipos de recursos básicos solamente se necesita realizar un plugin y exponerlo como una aplicación. Esto es posible gracias a los mecanismos de extensión provistos por el framework. El plugin debe de estar expuesto en el nodo soporte o en alguno de los nodos mirror existentes para que pueda ser utilizado.

7. Metodología de trabajo

En esta sección se verá principalmente cual fue la agenda del proyecto desde el punto de vista de las fechas, hitos e iteraciones. Antes de empezar con el tema central de la sección es necesario ver como es que se estimaron los tiempos del proyecto.

El grupo se enfrentó por primera vez a un proyecto de las características del actual. Debido a esto, las estimaciones se realizaron en base a la experiencia de los tutores ya que no son ajenos a la materia informática y cuentan con mayor experiencia en el área. Sin embargo, tras ingresar más a fondo en la realidad del proyecto se comprendió que a pesar de no tener puntos propios de referencia, de todas maneras le era posible realizar estimaciones propias en función del análisis realizado del problema, y aplicando las experiencias adquiridas en otros proyectos anteriores a pesar de no ser "tan" similares al actual.

Realizadas las puntualizaciones anteriores, se concluyó que el proyecto requerirá al menos hasta Junio de 2005 para su finalización. Si bien esta estimación fue realizada en concordancia con todos los interesados en el proyecto, el equipo presentó un plan de trabajo personalizado en donde se definen fases e iteraciones, cada una con sus propios objetivos y actividades. Dicho cronograma fue estudiado y aprobado en reuniones realizadas durante el mes de Septiembre de 2004 y fue ajustándose a lo largo del proyecto.

El grupo también realizó un análisis de riesgos y propuso diferentes formas de mitigarlos. Se utilizó la herramienta de configuración CVS para llevar el versionado de la documentación y el código. Esta fue una forma de mitigar uno de los riesgos con mayor probabilidad de ocurrencia que es el hecho de perder el control sobre las versiones de los documentos y el código. Se realizo un documento detallando todos los riesgos que podrían ocurrir durante el transcurso del proyecto y la forma de abordarlos en caso de ocurrencia (Anexo F).

7.1 Agenda del proyecto

Se definieron fases e iteraciones las cuales se corresponden con el modelo de proceso de referencia a utilizar, el cual en este caso se trata del modelo RUP. Debido a las características de proyecto y el número pequeño de integrantes del grupo, la instanciación del modelo se presenta bastante simplificada y no se realizan aquellas actividades que el grupo considera prescindibles o poco importantes. Lo que se intento realizar fue una adaptación del modelo RUP de acuerdo a la cantidad de integrantes del grupo.

Con respecto a las fases se definieron 4: Inicial, Elaboración, Construcción y Transición. Dentro de cada una de ellas se realizarán iteraciones en donde cada una de ellas tiene sus propias metas y actividades, del mismo modo y en concordancia con las metas y actividades de la fase que las contiene. Una descripción más fina de estas fases e iteraciones aparecerá mas adelantes en la sección.

Es importante destacar que el cronograma seguido por el grupo esta alineado con el cronograma general definido por el grupo de trabajo del proyecto LinkAll.

7.2 Plan de trabajo

En este punto se detallan las características de cada fase e iteración en función del modelo de referencia RUP. Se incluyen objetivos y principales actividades.

7.2.1 Inicial

Esta fase contó con dos iteraciones, en la primera iteración se relevaron los requerimientos del sistema, se comprendió cual era el sistema a construir, se detectaron los casos de uso prioritarios del sistema. Además se realizó un glosario para definir los distintos términos utilizados en los documentos.

En la segunda iteración de la fase se terminaron de relevar los requerimientos, los mismos fueron priorizados y validados. Se terminaron de definir todos los casos de uso y fueron priorizados. Se definió el alcance funcional del sistema así como también se estimó la duración del proyecto. En esta iteración se identificaron los riesgos del proyecto y se definió el entorno de desarrollo para el mismo.

7.2.2 Elaboración

En esta fase se analizó, diseñó e implementó la arquitectura del sistema, llegando al final de la fase con el ejecutable de la línea base de la arquitectura, el cual fue la base para las funcionalidades que se fueron sumando en la fase de Construcción y así llegar al ejecutable final del sistema.

Los riesgos identificados fueron monitoreados a la vez que se buscaron nuevos riesgos. La gestión y calidad del proyecto fue controlada. Se revisó la calidad de los productos y el ajuste al proceso. La gestión de configuración fue controlada, se controlan las versiones; para el control de la configuración se utilizo la herramienta CVS.

Se definió la vista de análisis con los principales paquetes subsistemas y clases. Se obtuvo una arquitectura, no tan estable como se había planificado pero con una vista del modelo de análisis, una del modelo de casos de uso y otra vista del modelo de distribución. Se implementó prototipo a nivel de la arquitectura que se tenía en el momento. Se realizó una planificación de la verificación y se definió un plan de pruebas para la fase de construcción.

Se continuó con la verificación de los documentos pero no se pudieron realizar las pruebas del sistema (prototipo) de acuerdo a lo planificado.

7.2.3 Construcción

En esta fase se completó el análisis, diseño e implementación de todas las funcionalidades del sistema. También fueron verificadas la mayoría de las funcionalidades del Sistema. La gestión y calidad del proyecto fue controlada, se revisó la calidad de los productos y el ajuste al proceso. La gestión de la configuración siguió siendo controlada.

En esta fase se agregó la vista del modelo de diseño y la vista del modelo de implementación a la arquitectura del sistema. Al igual que en la fase anterior se verificaron los documentos; también se realizaron pruebas sobre los componentes a medida que se iban implementando.

Esta fase contó con tres iteraciones, en la primera fueron implementadas las funcionalidades básicas del sistema, se diseñaron los subsistemas y clases para estas funcionalidades. La arquitectura fue ajustada agregándose a la misma la vista del modelo de diseño para las funcionalidades implementadas en esta iteración. Se realizaron pruebas sobre los componentes implementados.

En la segunda se implementaron las funcionalidades de nivel de calidad 5 y se diseñaron los subsistemas y clases de estos componentes. Se agregó a la vista del modelo de diseño de la arquitectura el diseño de las funcionalidades implementadas en esta iteración y se agregó vista de implementación. Se finalizaron y verificaron todos los modelos con lo que la arquitectura fue finalizada. Se realizaron pruebas sobre los componentes implementados en esta iteración.

En la tercera se culmino con todas las funcionalidades dentro de alcance del sistema, se finalizó el diseño del sistema y se verificaron los componentes implementados en esta iteración. Además de esto se planifico la implantación.

7.2.4 Transición

En esta fase se aseguró que el dejo disponible el software para el uso por parte de los usuarios, se ajustaron los errores y defectos encontrados, y se verificó que el producto cumpliera con las especificaciones. Se dejo disponible el software para el uso por parte de los usuarios.

Se realizaron pruebas del sistema, casos de uso y componentes, se implantó el sistema, se realizaron ajustes mínimos sobre los errores y defectos encontrados. Se finalizo con la documentación y se verificaron los documentos. Esta fase contó con una sola iteración debido a problemas en los tiempos.

8. Testing y calidad

La actividad de testing en el proyecto, requiere la ejecución de diversas tareas en función de niveles de calidad exigidos según el caso. Dicho niveles fueron definidos por el Ing. Diego Vallespir en su estudio sobre las actividades para mejorar la calidad del software en los proyectos de grado [5]. En este capítulo se comentará brevemente cuales son los niveles existentes, en que se basa cada uno y se profundizará en los niveles exigidos para el proyecto. Se verán cuales son las actividades relacionadas con cada nivel de calidad y como están fueron llevadas a cabo durante el desarrollo del proyecto.

Los niveles de calidad se definen en base a las actividades que hay que desarrollar en cada uno de ellos. Existen ocho niveles de calidad definidos. Es recomendable que para un proyecto de grado se establezca el Nivel 3 o el Nivel 4, sin embargo, si el producto a realizar en el proyecto de grado cuenta con una complejidad considerable se pueden exigir niveles más altos como el 5 y el 6. Los niveles 7 y 8 no son recomendados para proyectos de grado a menos que el sistema contenga algún aspecto crítico.

Debe tenerse en consideración que el alcance del proyecto puede verse disminuido al incrementar el nivel de calidad, ya que las actividades requieren tiempo. Sin embargo, si el nivel utilizado es 5 o 6 se puede resolver este problema realizando proyectos que continúen año a año lo realizado. Los niveles 5 y 6 aseguran que los productos sean fácilmente modificados y extensibles.

Las actividades a realizar no son lo único que determina la calidad de un producto. Hay algunos otros factores que también influyen en la calidad, como ser la gente que está desarrollando y las actividades que estos realicen fura de las actividades definidas. La ejecución de las actividades no garantiza estar en el nivel de calidad que la realización de las mismas indica, se podría llegar a estar en un nivel superior o en uno inferior. Al cumplir los objetivos de un nivel de calidad, se cumplen los logros de calidad del nivel anterior.

8.1 Niveles de calidad

A continuación se comentarán brevemente cuales serían los logros alcanzados al cumplir con las actividades de cada nivel de calidad.

Al cumplir con las actividades del nivel 1, se desconoce el funcionamiento del producto de software final. Incluso, este puede no cumplir con una sola de las funcionalidades especificadas correctamente. Claramente no es recomendable elegir este nivel. Es un nivel de alto riesgo.

Al cumplir con las actividades del nivel 2, se conoce el funcionamiento del producto final en casos determinados por el cliente. La variación en algunos datos de prueba aunque la funcionalidad sea la misma puede provocar fallas en el sistema. El orden de ejecución de los casos también puede afectar el buen funcionamiento del sistema y detectar fallas. Es un nivel de alto riesgo pero permite realizar demos controladas (usando los casos especificados en la prueba de aceptación). Este producto no esta cercano a un producto para ser puesto en producción sin correr altos riesgos.

En el nivel 3 deben de realizarse las siguientes actividades:

- Se conocen fallas del sistema y se pueden reproducir.
- Si las pruebas están automatizadas aumenta la capacidad de poder corregir las faltas en el código y comprobar que el sistema ya no falla en los casos automatizados (modificabilidad, mantenibilidad).
- Se conoce la cantidad de fallas relacionada con la cantidad de casos de prueba funcionales ejecutados.

- La confiabilidad en el sistema varía según el criterio de cubrimiento establecido y el nivel de adecuación que se alcanzó.

En este nivel se conocen fallas del sistema y cuanta funcionalidad del mismo ha sido cubierto por los casos de prueba. De esta manera se puede conocer que cosas el sistema no brinda (o de que manera determinadas funcionalidades no se deben ejecutar) y preparar futuras versiones sabiendo de antemano que es lo que hay que corregir. También se tiene una idea de que falta testear. Este nivel permite tener confianza en la funcionalidad del sistema puediendo hacer demos y teniendo conocimiento de algunas fallas del mismo, lo cuál permite "moverse" por el sistema con mayor flexibilidad.

El nivel 4 es recomendable para realizar demos y para establecer un prototipo que asegura un buen cumplimiento de las funcionalidades. Además se tiene una exigencia sobre la cantidad de fallas máximas que el sistema puede tener en los casos de prueba y una exigencia en el criterio de cubrimiento (y este se debe cumplir al 100%). El Plan de Pruebas puede alivianar el criterio de cubrimiento para funcionalidades o casos de uso de poca criticidad o aumentar el criterio en caso contrario.

En el nivel 5 deben de realizarse las siguientes actividades:

- Aumenta el nivel de calidad global del producto.
- Mayor posibilidad de reuso de componentes y del sistema entero.
- Mayor modificabilidad y mantenibilidad.
- Menor cantidad de faltas.
- Mayor posibilidad que el software funcione en casos no comunes.

En este nivel se resalta la disminución en la tasa de faltas en el sistema y la posibilidad de reuso. Es recomendable estar en este nivel si el producto se va a continuar en años subsiguientes o si se requiere un número relativamente bajo de fallas o esperar que estas no sean graves.

Tanto el nivel 6 como los posteriores hacen que el producto vaya reduciendo su probabilidad de fallar, su facilidad de modificación, adaptación, interoperabilidad con otros sistemas, facilidad de reuso entre otros. A partir de este nivel la probabilidad de encontrar fallas graves en el sistema es mucho más baja que estando en niveles inferiores.

Tanto el nivel 7 como el posterior hacen que el producto vaya reduciendo su probabilidad de fallar, su facilidad de modificación, adaptación, interoperabilidad con otros sistemas, facilidad de reuso entre otros.

El nivel 8 hace que el producto vaya reduciendo su probabilidad de fallar, su facilidad de modificación, adaptación, interoperabilidad con otros sistemas, facilidad de reuso entre otros.

8.2 Nivel de calidad del proyecto

Para el presente proyecto los tutores han decidido que el mismo deberá tener un nivel mixto de calidad, un nivel 5 para las funcionalidades consideradas críticas y un nivel 3 para el resto. Las funcionalidades definidas como de nivel 5 son las representadas por los casos de uso prioritarios. En resumen, las funcionalidades de nivel 5 son:

- Registro de paquetes en el nodo soporte
- Descarga de paquetes al nodo objetivo
- Deployment de aplicaciones
    - Infraestructura general
        - Motor de acciones
        - Motor de resolución de enlaces
    - Acciones específicas
        - Instalador de JARs
        - Instalador de WARs
    - Tipos de recursos específicos
        - SessionBean

A nivel de componentes, se considerará que todos aquellos componentes que provean funcionalidades de nivel 5, serán tomados completamente como de nivel 5 y no solo la funcionalidad critica que el mismo resuelve.

A continuación se describirán las actividades a realizar para los niveles seleccionados para el proyecto. Las actividades de **nivel 3** son las siguientes:

Especificaciones:

- Relevar Requerimientos y Priorizarlos
- Validar Requerimientos
- Generar Casos de Uso y Priorizarlos
- Validar Casos de Uso

Verificación y Validación:

- Testing de los Casos de Uso
- Testing Funcional del Sistema
- Test de Aceptación
- Establecimiento de Cubrimiento (de Req. y/o Casos de Uso)

Métricas y Medidas:

- Cantidad de Fallas/Casos de Prueba (de Req. y Casos de Uso)
- Midiendo el Cubrimiento (de Req. y/o Casos de Uso)

Las actividades a realizar para el **nivel 5** son las siguientes:

Especificaciones:

- Relevar Requerimientos y Priorizarlos
- Validar Requerimientos
- Generar Casos de Uso y Priorizarlos
- Validar Casos de Uso
- Especificación de Componentes y/o Especificación Formal de Componentes (con tendencia a usar semi-formalismos como OCL y diagramas de estados de componentes)

Verificación y Validación:

- Testing de los Casos de Uso
- Testing Funcional del Sistema
- Test de Aceptación
- Establecimiento de Cubrimiento (de Req. y/o Casos de Uso y Componentes)
- Test de Regresión
- Planificación del Testing (de Req y/o Casos de Uso y Componentes)
- Testing de Componentes
- Testing de Integración de Componentes

Métricas y Medidas:

- Cantidad de Fallas/Casos de Prueba (de Req. y Casos de Uso y Componentes)
- Midiendo el Cubrimiento (de Req. y/o Casos de Uso y Componentes)
- Midiendo el Diseño

Luego de definir el nivel de calidad deseado para el proyecto y ver cuales son las actividades de calidad a realizar durante desarrollo el mismo, es tiempo de ver como fueron aplicadas las mismas por el grupo, en que grado y que resultados fueron obtenidos.

8.2.1 Especificaciones

Durante el proceso de especificación fueron definidos los requerimientos funcionales del sistema, priorizados y validados. La estrategia seguida para recabar los requerimientos del sistema fue la de reuniones con los tutores, en las cuales se trataban de analizar y definir los mismos. Durante la recolección de los requerimientos se tuvieron algunos inconvenientes para que estos quedaran fijos ya que como se depende del proyecto Link All y esta plataforma recién estaba comenzando a ser construida. Los problemas se dieron porque al estar en proceso de definición la plataforma era bastante cambiante, lo que repercutía en los requerimientos del sistema.

A pesar de esto, las funcionalidades principales del sistema si pudieron ser definidas en los primeros momentos aunque luego tuvieran algunas modificaciones pero solamente detalles y no grandes cambios.

La priorización de los requerimientos se realizo junto con los tutores en una de las reuniones de requerimientos luego de que estos estaban prácticamente estables. Con respecto a la validación, esta se realizaba continuamente en cada reunión de requerimientos hasta que se llego a que los requerimientos estaban correctos y completos. Luego de esto se tuvo que realizar otra validación debido a que alguno de los requerimientos cambiaron, no fue un cambio total sino que se fueron ajustando detalles de acuerdo a lo que era se requería para la plataforma Link All. El resultado del análisis de requerimientos es el documento de requerimientos. Esta versión es con respecto a la primera versión del documento que se puso bajo control de configuración.

En lo que respecta a la definición de los casos de uso, su priorización y posterior validación, se realizo de manera similar a los requerimientos. La definición de los casos de uso se comenzó cuando los requerimientos se consideraban prácticamente estables y antes de su validación final. Al igual que sucedió son los requerimientos, los casos de uso debieron ser ajustados a medida que los requerimientos se modificaban y que los tutores iban definiendo más concretamente como querían que las funcionalidades del sistema fueran resueltas.

Por la misma razón que los requerimientos tuvieron que ser validados en una segunda oportunidad, para los casos de uso sucedió lo mismo. El resultado del análisis de casos de uso es el documento de casos de uso (Anexo B). Esta versión es con respecto a la primera versión del documento que se puso bajo control de configuración.

Con respecto a la especificación de componentes, se realizo sin la participación de los tutores. Esta actividad de calidad fue tomada como parte de la actividad de diseño por parte del grupo y estuvo prácticamente estable en el mes de febrero. Las razones para que la especificación de componentes no estuviera estable antes fueron varias. La primera es que los requerimientos fueron cambiando, como se explico antes, y el documento de requerimientos es la entrada para la actividad de especificación. Se tuvieron la mayoría de los componentes especificados en una primera etapa en la cual se pensaba que los requerimientos estaban estables, pero luego hubo que cambiar la especificación de los componentes.

Otra razón fue que a medida que el grupo se interiorizaba más en el sistema e iba diseñando e implementando los componentes, surgían cambios para la especificación de los mismos. En el mes de febrero el grupo tenia finalizado el diseño del sistema por lo que los componentes pasaron a estar estables, antes de esa fecha fue imposible tener una especificación estable de los componentes ya que durante el diseño surgieron cambios que afectaban la especificación de los mismos. El resultado de la especificación de componentes se describe en el documento de la arquitectura (Anexo C).

8.2.2 Verificación y validación

Lo primero que se hizo en cuanto a la verificación y validación del sistema fue definir un plan de testing (Anexo G), cuyo propósito fue definir una línea de trabajo que guíe las actividades de verificación a realizar durante el transcurso del proyecto.

Las actividades enmarcadas en este plan de Verificación y Validación fueron llevadas a cabo sobre los artefactos (componentes) del producto en construcción. Esto determinó que se verificarán los documentos generados en las líneas de trabajo básicas, es decir los referidos al producto en sí mismo y no aquellos puramente administrativos. Los componentes del sistema, las versiones intermedias y la versión implantable.

Las tareas del área de verificación fueron llevadas a cabo por los integrantes del grupo (dos miembros). Por ser un grupo pequeño, no se realizaron subdivisiones dentro del mismo para dedicar un subgrupo a la verificación y validación.

Se verificaron las últimas versiones de los documentos generados y discutidos con los tutores, con el objetivo de acelerar el proceso de corrección de errores. Luego de las reuniones con los tutores en las cuales se veían los documentos generados, el grupo realizaba reuniones para ver cuales eran los cambios en los documentos y cual era su repercusión sobre el sistema. En caso de detectar fallas en el producto, estas fueron corregidas lo más rápido posible.

Las metodologías seguidas para la verificación y validación fueron dos. Se utilizaron *inspecciones* con el objetivo de revisar el componente y la documentación del mismo. Con esto se trató de asegurar que se realizara lo que marcaba el diseño, que la codificación siguiera el estándar definido y que su documentación sea consistente y completa. Cada integrante del grupo revisaba el componente realizado y su documentación en forma individual. Esta metodología fue aplicada a los componentes críticos de nivel 5.

La otra metodología utilizada fue el *enfoque de caja negra*. Se consideró el componente como una caja negra al cual se le aplicaron diferentes casos de prueba. Los resultados obtenidos por la serie de pruebas se compararon con las salidas esperadas de acuerdo a la especificación del componente.

Se definieron cubrimientos para los requerimientos, los casos de uso y los componentes. El cubrimiento para la especificación de requerimientos elegido es: “se tendrá al menos un caso de prueba para cada requerimiento especificado”.

El cubrimiento de casos de uso elegido es: para los casos de uso que cubran funcionalidades de nivel de calidad 3, se realizara un cubrimiento del escenario del flujo normal. Para los casos de uso que cubran funcionalidades de nivel 5, se realizara un cubrimiento de todos los escenarios posibles del caso de uso.

El cubrimiento de componentes elegido es: “se cubrirán todas las funcionalidades que la componente brinda”.

Se realizo un documento de casos de prueba (Anexo G) en el que se especifican pruebas que satisfacen los cubrimientos elegidos para los requerimientos y casos de uso. También se realizó un documento de pruebas de componentes (Anexo G) en el que se especifican las pruebas a realizar sobre cada componente a fin de cumplir con el cubrimiento especificado para los mismos, estas pruebas también incluyen tests de integración de componentes. La implementación de los test de componentes fue realizada con JUnit.

El criterio de aceptación del producto, definido por los tutores, es que el mismo provea todas sus funcionalidades requeridas para la plataforma LinkAll en materia de deployment. Con esto los tutores darían por aceptado el sistema.

Durante el transcurso del proyecto se realizaron tests de regresión para asegurar que los defectos encontrados durante el desarrollo del software fueran mitigados y no aparecieran en nuevas versiones del mismo. Estos tests fueron ejecutados cada vez que se contaba con una nueva versión de un componente. Los mismos se utilizaron para probar las funcionalidades de los componentes del sistema versión a versión y así evitar la reaparición de fallas.

8.2.3 Métricas y Medidas

La medida del cubrimiento es de un cien por ciento en lo que respecta a requerimientos, casos de uso y componentes. Esto quiere decir que los tests diseñados deben satisfacer completamente los cubrimientos especificados.

Las pruebas del sistema y sus resultados esperados fueron especificadas en el plan de verificación y validación. Las pruebas sobre los componentes fueron definidas en el documento de pruebas de componentes. (Se adjuntan todos en el Anexo G)

Las pruebas de componentes fueron realizadas mientras se desarrollaba el sistema. Cabe destacar que en un principio los procedimientos de testing no fueron realizados formalmente pese a que esta era la intención del grupo. Con testing formal nos referimos a test implementado con alguna herramienta de verificación, como por ejemplo JUnit, que fue la herramienta finalmente utilizada por el grupo. Esta demora en utilizar herramientas formales de testeo se debió a los tiempos acotados con los que se contaba. El grupo generó tests JUnit en cuanto pudo.

Pese a no utilizar una herramienta formal para el testeo de los componentes, se crearon tests con los cuales pudieron ser detectadas fallas en las funcionalidades provistas por los componentes. Estos test también fueron utilizados para realizar los test de regresión.

Las pruebas funcionales fueron ejecutadas parcialmente en cada liberación del sistema y totalmente en la versión final del mismo. En las versiones intermedias se probaron las funcionalidades pero no en la profundidad que se realizó en el testeo final. La diferencia se encuentra en que en las pruebas finales se testeó la robustez del sistema además de sus funcionalidades. En las pruebas intermedias se probaron más las funcionalidades del sistema que su robustez.

A continuación se presentarán los resultados obtenidos en el testeo de la versión final del sistema. Se presentarán los resultados discriminados en pruebas de componentes y pruebas funcionales de casos de uso y requerimientos.

- Pruebas de componentes

  Del testeo realizado a los componentes se obtuvo el siguiente índice:

  Cantidad de Fallas/Casos de Prueba = 0.044

  Este índice satisface el requerido para el nivel de calidad 5. Las fallas se dieron principalmente en el control de errores dentro de los componentes.

- Pruebas funcionales

  Del testing funcional se obtuvo el siguiente índice:

  Cantidad de Fallas/Casos de Prueba = 0.086

  Este índice satisface el requerido para el nivel de calidad 5. En este caso las fallas fueron variadas pero en lo que respecta a casos de uso y requerimiento el resultado obtenido fue menor al expuesto anteriormente. Las fallas radicaron principalmente en problemas de configuración del sistema e interfaz de usuario, principalmente detalles en la misma.

9. Conclusiones

La oportunidad de desarrollo del componente de deployment de la plataforma LinkAll, se presentó como una buena razón para estudiar y comprender los distintos enfoques vinculados con el deployment de aplicaciones. Como parte del trabajo de análisis del estado del arte, se pudo estudiar los detalles del ciclo de vida de deployment [4] pudiendo tener una referencia clara de cuáles son las funcionalidades que un sistema de deployment de aplicaciones debe brindar para cubrir cada actividad propuesta en el mismo. Este relevamiento permitió comparar las capacidades del framework con otras soluciones existentes. Pero antes de realizar conclusiones en base a la comparación con otras propuestas, examinemos cuál es el grado de cumplimiento de los objetivos establecidos para el presente trabajo.

A continuación veremos los objetivos generales y como estos son satisfechos por el sistema:

- **Deployment de nuevos componentes por parte de usuarios no técnicos**

  El sistema realiza el deployment de nuevas aplicaciones automáticamente sin necesidad de una posterior configuración de la plataforma objetivo. Por otro lado, el control de dependencias permite identificar sencilla y rápidamente las aplicaciones que faltan instalar según las necesidades de una aplicación que se quiere deployar. Las causas de los errores pueden ser fácilmente identificadas mediante el mecanismo de logging incorporado en el sistema, y mediante los reportes que retornan todas las funcionalidades del sistema. Cada funcionalidad que es ejecutada por el sistema genera un reporte conteniendo un informe de cada tarea realizada y de los problemas que pudieron haber surgido durante la ejecución de la misma. Dicho reporte se compone por una serie de errores, advertencias y notas.

- **Configuración y registro del modelo de datos y metadatos a ser usado por los componentes a ser instalados**

  Las acciones desarrolladas dentro del presente trabajo, permiten la incorporación de nuevas aplicaciones de manera automática. Así también brindan servicios de registro de sus correspondientes modelos de datos y metadatos. Debido a la flexibilidad en el desarrollo e incorporación de nuevas acciones, cualquier nueva acción que incorpore funcionalidad adicional para la actividad de configuración puede ser ¡mplementada y publicada como una aplicación común. Posteriormente, la nueva aplicación (plugin) puede ser instalada en un nodo objetivo para efectivizar la incorporación de la nueva acción.

- **Re-deployment y des-deployment (eliminación) de aplicaciones, manteniendo la base de datos y su configuración**

  En lo que respecta al Re-deployment y Des-deployment, es posible afirmar que los objetivos son satisfechos por el sistema, incluso pueden hacerse “en caliente”, es decir, sin necesidad de reiniciar la plataforma objetivo (para el caso de la plataforma LinkAll, un servidor J2EE). Con respecto al Upgrade, caso particular de Re-Deployment, el objetivo esta cumplido pero puede mejorarse. En la versión actual del sistema el Upgrade de una aplicación implica realizar un Undeployment de la versión instalada (versión anterior), y posteriormente realizar un Deployment de la nueva versión. Debido a esto, no es posible mantener la configuración y datos de la anterior versión. Resolver este asunto, agregando un manejo elegante y elaborado de esta problemática, forma parte del trabajo a realizar a futuro.

- **Registro del estado de cada instalación de forma de poder recomendar la instalación de nuevas liberaciones**

  El sistema provee un interfaces de consulta que permiten, desde la interfaz de usuario, informar al mismo acerca de la existencia de nuevas versiones de una aplicación, ya sea para descargar desde un servidor de paquetes, o para instalar en un nodo objetivo. Adicionalmente, también como trabajo a futuro, es posible utilizar el mismo mecanismo de consulta para asesorar al usuario acerca de la necesidad de instalar ciertas aplicaciones en función de las que seleccione para descargar o instalar. Para ello, es posible valerse del mecanismo de control de dependencias para determinar los requerimientos de cada nueva aplicación seleccionada.

Con respecto al objetivo secundario establecido, **desarrollar utilitarios que permitan monitorear el estado de los deployments de forma de poder retomar la misma en caso de fallas** no es una funcionalidad en la que el sistema actualmente provea soporte, pero es posible tomar dicha problemática como trabajo a realizar a futuro. Con respecto a **deshacer los pasos realizados dejando el sistema en su estado original en caso de fallas**, dicha propiedad se asegura mediante un control de errores a nivel del framework y mediante la realización de un rollback de los cambios realizados sobre la base administrativa LinkAll para el caso de aplicación en dicha plataforma.

Sobre los objetivos relacionados con algunos factores de calidad del producto a construir, el producto cuenta con una interfaz de usuario amigable, de fácil utilización, y tal que cualquier usuario inexperto la pueda utilizar sin mayores inconvenientes. Además, el sistema cumple con el objetivo de extensibilidad ya que es fácilmente extensible mediante la utilización de plugins. Dicho mecanismo permite agregar funcionalidades que cubren ampliamente todos los servicios brindados por el framework permitiendo dotar al sistema objetivo de mayor poder en función de los nuevos requerimientos que se presenten. Esta característica, por otra parte, permite una fácil y práctica personalización del sistema.

Una vez realizada la comparación con respecto a los objetivos establecidos al comienzo del presente proyecto, ahora es interesante realizar una comparación con otras soluciones existentes. En la siguiente tabla, se presentan diferentes soluciones existentes (las principales) en función de las actividades del ciclo de deployment. Un "**-**" significa que la solución no brinda soporte a la funcionalidad, "0" significa que brinda un soporte parcial o cierto nivel de soporte, mientras que un "**+**" significa que brinda un soporte total o mayor al nivel promedio.

		PgDploy1	*NetInstall*	*NetDeploy*	*InstallShield*	*RPM*	*HP-UX SD*
Release	***Package***	**0**	**0**	**0**	**0**	**+**	**+**
	Advertise	**+**	**-**	**-**	**-**	**-**	**-**
Install	***Transfer***	**+**	**0**	**0**	**-**	**0**	**0**
	Configure	**0**	**0**	**0**	**+**	**+**	**+**
Activate		**-**	**-**	**-**	**-**	**-**	**-**
Update	***Transfer***	**+**	**-**	**0**	**-**	**0**	**-**
	Re-configure	**0**	**-**	**0**	**-**	**+**	**+**
Adapt		**0**	**-**	**-**	**-**	**0**	**0**
Deactivate		**-**	**-**	**-**	**-**	**-**	**-**
De-install		**+**	**+**	**+**	**+**	**+**	**+**
De-release		**0**	**-**	**-**	**-**	**-**	**-**

Fig. 9-1 - Comparación con otras soluciones

De la tabla anteriormente presentada, se desprende como principal conclusión el hecho de brindar soporte parcial, o mayor al promedio, de casi la totalidad del ciclo. Sin embargo, vale hacer una importante aclaración. Las herramientas contra las cuales se realiza la comparación brindan una solución a un problema definido y prácticamente no proveen mecanismos que permitan realizar personalizaciones dependiendo de la plataforma objetivo, ni tampoco permiten agregar nuevas funcionalidades en base a plugins. Esta consideración es de suma importancia debido a dos aspectos:

- Debido a que el framework brinda soporte para una amplia parte del ciclo, y debido a que asume que quien haga uso del mismo se encargará de personalizarlo según sus necesidades, ciertas funcionalidades como la activación y desactivación, o aquellas que son medianamente soportadas, pueden ser implementadas sobre él cumpliendo con la totalidad del ciclo de deployment siempre que el desarrollador se lo proponga. Por tanto, brinda la infraestructura sobre la cual construir una solución completa, a diferencia de otras soluciones.
- La capacidad de incorporar nuevas funcionalidades mediante el mecanismo de extensión mediante plugins, incorpora una nueva dimensión no considerable para el caso de las demás soluciones actuales existentes. Esta cualidad dota al framework de un elemento de suma importancia que lo destaca de las demás soluciones.

En definitiva, el framework desarrollado se presenta una alternativa bastante completa y potente sobre la cual realizar la solución de deployment para una cierta plataforma objetivo. En posteriores trabajos sería posible perfeccionarlo y eventualmente dotarlo de nuevas capacidades para lograr completar el cubrimiento del ciclo de deployment completo. Este trabajo puede ser tomado como puntapié inicial para estudiar más detenidamente la problemática de delpoyment de aplicaciones para así lograr alcanzar una solución más general y completa.

10. Trabajo futuro

Si bien el sistema provee las funcionalidades requeridas en un comienzo, muchos aspectos pueden ser mejorados, o bien, nuevos elementos pueden ser considerados. Estas consideraciones son incluidas en este capítulo que se centra en aquellas funcionalidades que pueden ser mejoradas o bien incorporadas en posteriores versiones del producto, quedando planteadas como trabajo a futuro.

En primer lugar detallamos algunas mejoras a realizar sobre el sistema existente.

- **Dependencias**

  El mecanismo actual de control dependencias no conoce de antemano cuáles son los tipos de dependencias existentes ya que está basado en un mecanismo de plugins. En tal sentido, no se hace diferencia entre dependencia hacia una versión de una cierta aplicación, de lo que es un requerimiento, ya sea de hardware o de cualquier otro tipo de elemento en el ambiente objetivo. Un algoritmo de control de dependencias entre aplicaciones es mucho más sofisticado que uno de chequeo de requerimientos, puesto que el segundo no requiere control contra las restantes aplicaciones sino contra la configuración de la plataforma objetivo. Esta diferencia conlleva a que con el mecanismo actual, que verifica entre aplicaciones, se esté realizando un trabajo innecesario si consideramos, por ejemplo, una dependencia que deba chequear que la memora RAM del sistema sea 512 MB. Además, conceptualmente son términos de naturaleza distinta el de “dependencia”, del de “requerimiento”. Lo que se propone mejorar a futuro, es el mecanismo de control de dependencias, separando lo que son dependencias hacia otras aplicaciones de lo que son requerimientos. Las dependencias se manejarían de manera estática, mientras que los requerimientos se basarían en el mecanismo de plugins para su implementación.

- **Upgrade**

  Actualmente el mecanismo de Upgrade es muy rudimentario y consta en desinstalar la versión vieja e instalar la nueva versión. Con esta solución existen varios problemas, entre otros, no es posible conservar datos de una versión a la otra y no es posible conservar componentes de una versión a la otra, lo que optimizaría el tiempo de instalación de la nueva versión. Este aspecto es claramente mejorable quedando también como trabajo a futuro.

- **Modificación de la configuración de la plataforma objetivo**

  Muchos aspectos de configuración de la plataforma objetivo, en lo referente a las aplicaciones, se realizan de manera automática cumpliendo con los requerimientos presentados al comienzo de este informe. El framework provee, a su vez, de la infraestructura sobre la cual poder realizar modificaciones “en caliente” de la misma. Sin embargo, dichos servicios no son aprovechados debido a que optimizando el tiempo disponible para el desarrollo del proyecto, ciertas funcionalidades no fueron reflejadas en la interfaz de usuario. En este sentido se propone extender la interfaz de usuario permitiendo al usuario reconfigurar la plataforma objetivo “en caliente”.

- **Seguridad en la creación de nodos**

  Actualmente la configuración del tipo de nodo (soporte, mirror, objetivo) se realiza a través de los archivos de configuración del framework. Este aspecto, no brinda la seguridad de que alguien pueda cambiar un nodo configurado como mirror, por ejemplo, para que tenga las características de un nodo soporte. Esta es una debilidad del sistema por lo que se propone buscar algún mecanismo para obtener permisos al momento de agregar un nodo mirror o soporte. También se propone buscar una solución de seguridad para el acceso a los nodos servidores de paquetes desde otros nodos.

- **Mecanismos de firmado**

  Con respecto al mecanismo de firmado existente, el mismo es un tanto rudimentario. Como propuesta de mejora, pueden estudiarse más detenidamente las funcionalidades provistas por estándares en materia de cifrado y firmas digitales que estén incluidos en la propia API de Java, o particularmente, para el caso de la plataforma J2EE, estudiar los servicios en materia de seguridad que la misma proporciona.

- **Persistencia**

  Actualmente la persistencia se realiza en archivos XML. Este mecanismo puede ser mejorado de tal manera de que sea más personalizable para seguir dotando al framework de mayor flexibilidad e independencia de la implementación concreta donde se aplique. En otros contextos, podría requerirse la utilización de un motor de base de datos, u otros medios de persistencia.

- **Diseño**

  El diseño podría ser revisado para mejorarse, si se cree necesario, en su modularidad, preformace, etc. Claro está, que lo desarrollado dentro del marco del presente proyecto, puede considerarse un prototipo sobre el cual iterar para obtener una mejor solución.

Dentro de las funcionalidades que serían interesantes para agregar al sistema, una de ellas es la inclusión de un **Empaquetador de Aplicaciones**. Esta funcionalidad permitiría, dada una aplicación, crear su paquete deployable. Esto incluiría generar los tres archivos de de configuración (Descriptor de Aplicación, Especificación de Deployment y Especificación de Recursos). Eventualmente esta funcionalidad también podría firmar los paquetes. Añadiendo esta funcionalidad, se daría soporte completo al proceso de ***Release*** definido para el ciclo de vida de deployment.

Otro elemento a considerar a futuro, es la realización de un caso de estudio sobre otro tipos de plataformas ya sea J2EE, o de otro tipo. Este ensayo serviría para verificar que las características deseadas de generalidad se cumplen tal cual se espera que lo hagan. De no ser así, serviría como una buena oportunidad de encontrar nuevos elementos que permitan mejorar el trabajo realizado.

11. Referencias

[1] Proyecto LinkAll, http://www.link-all.org.

[2] Java 2 Platform, Enterprise Edition (J2EE), Sun Microsystems, Inc., http://java.sun.com/j2ee.

[3] Jboss AS, Jboss Inc., http://www.jboss.org/products/jbossas.

[4] "Software Deployment – Extending Configuration Management Support into the Field", André van der Hoek, Richard S. Hall, Antonio Carzaniga, Dennis Heimbigner, Alexander L. Wolf, http://www.stsc.hill.af.mil/crosstalk/1998/02/deployment.asp. 30/05/2005.

[5] "Actividades tendientes a mejorar la calidad de los productos de software", Ing. Diego Vallespir, http://www.fing.edu.uy/~dvallesp/Tesis/webActividades/index.htm. 30/05/2005.

[6] "A conceptual foundation for Component-Based Software Deployment", Allen Parrish, Brandon Dixon, David Cordes. Department of Computer Science, The University of Alabama, Junio 2000.

[7] "Requerimients for Software Deployment Languages and Schema", Richard S. Hall, Dennis Heimbigner, Alexander L. Wolf. Software Engineering Laboratory, Department of Computer Science, University of Colorado, 1998.

UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

ANEXO A

GLOSARIO

Estudiantes
Nicolás Doroskevich
Sebastián Moreira

Tutores
Federico Piedrabuena
Raúl Ruggia

Proyecto de grado 2004
Facultad de Ingeniería - Universidad de la República

Índice general

1. Conceptos y términos

1.1 Application (Aplicación)

Una aplicación se entiende como un conjunto de archivos

1.2 Deployable package (DP)

Archivo que empaqueta/contiene todos los archivos que conforman una aplicación a instalar (deployar). Es análogo a los archivos RPM de Linux o MSI de Windows.

1.3 Resource (Recurso)

Un recurso es cualquier elemento requerido por una aplicación y que es provisto por otra aplicación instalada en un mismo nodo objetivo. De esta manera, una aplicación pude proveer acceso a una de sus funcionalidades mediante un webservice y considerarlo un recurso provisto. Otra aplicación podrá especificar que requiere dicho recurso y el sistema le brindará de los mecanismos que puedan resolver dicho enlace.

1.4 Link (Enlace)

Por enlace se entiende como todo aquel canal que permita a una aplicación conectarse con otra haciendo uso de un cierto recurso que la segunda provee y que ésta requiere. Ambas se consideran "enlazables" si forman parte de un mismo nodo objetivo.

1.5 AD - Application descriptor (Descriptor de Aplicaciones)

Archivo que describe el DP indicando nombre, autor, versión, dependencias, entre otros elementos descriptivos. Es utilizado para realizar registrar la información de las aplicaciones al momento de deployment y para determinar previamente si las dependencias son satisfechas (por más detalles ver Doc. técnica – Descriptor de paquetes (IDT-descr)).

1.6 DS - Deployment specification (Especificación de Deployment)

Archivo en donde se especifican las acciones a tomar al momento de realizar el deployment de aplicaciones y también, al momento de realizar el undeployment de las mismas (por más detalles ver Doc. técnica – Especificación de Deployment (IDT-depSpec)).

1.7 RS - Resources specification (Especificación de Recursos)

Archivo en donde se especifican los recursos requeridos por las aplicaciones que se instalan mediante esta plataforma de deployment. Especifica recursos requeridos tanto como provistos, entendiéndose como recurso a cualquier elemento en la plataforma objetivo que se pueda realizar algún tipo de enlace mediante programación (EJB, WebService, etc.) así como cualquier información que se quiera dejar accesible públicamente para ser accedida por alguna aplicación que la considera como un recurso requerido (por más detalles ver Doc. técnica – Especificación de Recursos (IDT-resSpec)).

1.8 Client node (Nodo cliente)

El nodo cliente consiste simplemente en un PC de escritorio con capacidades que le permitan ejecutar un browser para ingresar a las aplicaciones web que ofician de punto de entrada a las funcionalidades provistas por los demás nodos del sistema.

1.9 Support node (Nodo soporte)

El nodo de soporte es donde reside el subsistema de ***Gestión de DP's*** el cual es responsable de administrar los DP's disponibles para ser bajados desde los nodos objetivo y posteriormente instalados en ellos. Consta de un servidor de aplicaciones J2EE que provee el ambiente de ejecución para el subsistema mencionado, un servidor web (o uno incorporado al servidor de aplicaciones) y un servidor FTP mediante el cual acceder y descargar de los DP's.

1.10 Target node (Nodo objetivo)

En este nodo se encuentra el servidor de aplicaciones J2EE objetivo para el deployment de DP's. Éstos son descargados desde el nodo soporte asociado y posteriormente son instalados. Contiene al subsistema de ***Descarga de paquetes***, es decir, el responsable de atender pedidos desde un cliente administrador del nodo y resolver la descarga de DP's desde el nodo soporte. En un nivel más bajo, reside el subsistema de ***Deployment*** el cual es responsable de realizar la instalación local de los DP's descargados por el subsistema de descarga de paquetes.

1.11 Support/Target node (Nodo intermedio)

Este nodo es un híbrido entre el nodo objetivo y el de soporte. Ejecuta sus propias aplicaciones (eventualmente obtenidas de un nodo soporte) y a su vez ofrece DP's para que otros nodos objetivo puedan descargar los mismos.

UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

ANEXO B

CASOS DE USO

Estudiantes
Nicolás Doroskevich
Sebastián Moreira

Tutores
Federico Piedrabuena
Raúl Ruggia

Proyecto de grado 2004
Facultad de Ingeniería - Universidad de la República

Índice general

1. Actores

A continuación aparece la descripción de cada uno de los actores que existen en el sistema a construir, que pueden ser algo o alguien que se encuentra fuera del sistema y que interactúa con él.

1.1 Administrador del nodo objetivo (TNA)

Es responsable de gestionar las aplicaciones a ser instaladas o desinstaladas en un cierto nodo objetivo. También es capaz de dar inicio o eventualmente detener la ejecución de las diversas aplicaciones instaladas. Posee permisos de acceso al los servidores de descargas (nodos intermedios y/o nodo soporte, desde donde se descargan las aplicaciones a instalar así también como su información descriptiva).

1.2 Administrador del nodo soporte LinkAll (SNA)

Es responsable de la administración de los paquetes en el nodo soporte LinkAll. Con administración nos referimos a dar de alta y de baja paquetes del nodo soporte. Este usuario también puede actuar como TNA.

2. Casos de uso

2.1 Diagramas de casos de uso

A continuación presentamos el diagrama de casos de uso. Posteriormente entraremos en detalle a cada caso de uso del sistema que se encuentra representado en el siguiente diagrama.

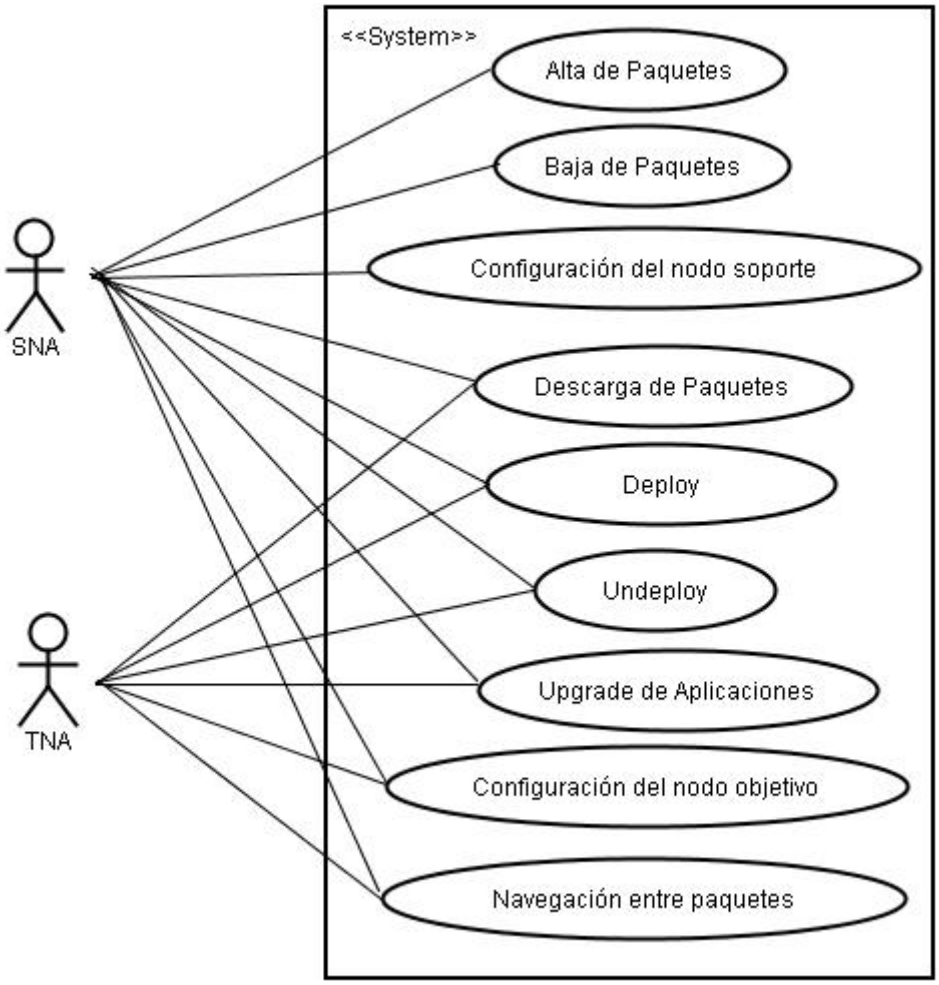


Figura 1 – Diagrama de casos de uso

2.1.1 Inclusiones de casos de uso

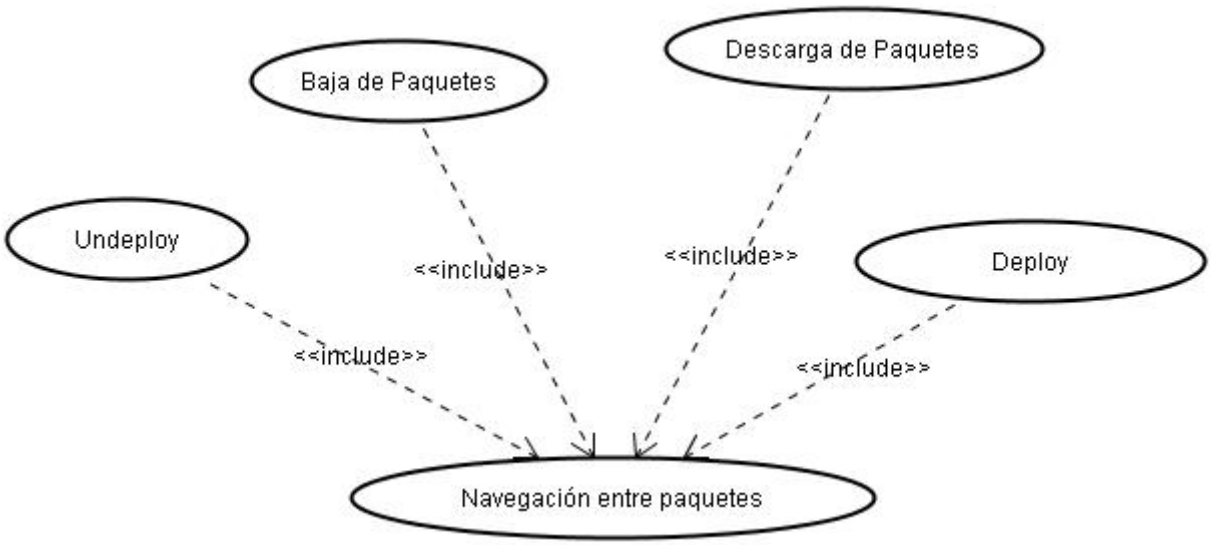


Figura 2 – Diagrama de inclusión de casos de uso

2.2 Alta de paquetes en el nodo soporte

El SNA selecciona la opción alta de paquetes, el sistema le pide que ingrese el camino al paquete y el nombre del mismo y lo da de alta en el nodo soporte.

2.2.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo soporte.

2.2.2 Flujo Principal

SNA	SISTEMA
1. Selecciona la opción de “Alta de paquetes”	
	2. Pide el path del paquete y el nombre del mismo
3. Ingresa el path y el nombre del paquete	
	4. Hace el upload del paquete
	5. Verifica la autenticidad del paquete
	6. Verifica que las aplicaciones Link All de las cuales depende la aplicación contenida en el paquete a dar de alta, estén disponibles para la descarga en el nodo soporte
	7. Registra el paquete en el nodo soporte
	8. Le comunica al usuario que la operación termino con éxito
	9. Termina el caso de uso

2.2.3 Flujo Alternativo

3.A El usuario no ingresa todos los campos requeridos

3.A.1 El sistema le comunica al usuario que debe ingresar todos los datos. Vuelve a 2.

5.A Falla la autenticación del paquete

5.A.1 El sistema le comunica al usuario que fallo la autenticación del paquete

5.A.2 Termina el caso de uso

6.A No están disponibles para la descarga todas las aplicaciones necesitadas por la aplicación contenida en el paquete

6.A.1 El sistema le muestra los warnings de lo sucedido al usuario.

6.A.2 Sigue en 7.

7.A El sistema no puede registrar el paquete en el nodo soporte

7.A.1 Le comunica al usuario que ocurrió un error.

7.A.2 Termina el caso de uso

G) El usuario puede cancelar en cualquier momento

2.3 Baja de paquetes en el nodo soporte

El SNA selecciona los paquetes que quiere dar de baja del nodo soporte de una lista de paquetes que el sistema le despliega. El sistema da de baja el paquete seleccionado.

2.3.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo soporte.

2.3.2 Flujo Principal

SNA	SISTEMA
1. Selecciona la opción de “Baja de paquetes”	
	2. Obtiene los descriptores de los paquetes disponibles para la descarga en el nodo soporte.
	3. Incluye el caso de uso “Navegación entre paquetes”.
4. Selecciona los paquetes a dar de baja	
	5. Pide al usuario que confirme la operación
6. Confirma la operación	
	7. Da de baja los paquetes seleccionados del nodo soporte
	8. Le comunica al usuario que la operación termino con éxito.
	9. Termina el caso de uso

2.3.3 Flujo Alternativo

6.A El usuario no confirma la operación

6.A.1 Termina el caso de uso

7.A No se puede dar de baja algún paquete del nodo soporte

7.A.1 El sistema elimina los paquetes que pueda y despliega un mensaje indicando los paquetes que no pudo dar de baja.

7.A.2 Termina el caso de uso.

G) El usuario puede cancelar en cualquier momento

2.4 Navegación entre paquetes

El sistema muestra las aplicaciones disponibles y las acciones que se le pueden aplicar, una acción de deploy, undeploy o upgrade. El usuario selecciona una aplicación y el sistema le muestra una descripción de la funcionalidad de la aplicación y la información de la misma (autor, fecha de creación, etc.). También se muestra cual es la versión más nueva de la aplicación existente en el nodo.

2.4.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo objetivo.

El usuario le indico al sistema que deseaba navegar entre los paquetes.

El sistema tiene seleccionados los descriptores de los paquetes a mostrar.

2.4.2 Flujo Principal

SNA/TNA	SISTEMA
	1. Analiza los descriptores de las aplicaciones disponibles observando su información para presentarla correctamente al usuario.
	2. Despliega en pantalla las aplicaciones, mostrando la versión, la fecha de creación. Distingue los paquetes instalados de los ugradeables y de los no instalados. También muestra si el paquete que contiene la aplicación se encuentra o no en el nodo objetivo.
3. El susuario selecciona una aplicación	
	4. Muestra la información de la aplicación (autor, empresa, etc.)
	5. Termina el caso de uso

2.4.3 Flujo Alternativo

G) El usuario puede cancelar en cualquier momento

2.5 Descarga de paquetes

El TNA se conecta al servidor de descargas utilizando la opción de descargas y el sistema le muestra los paquetes que están disponibles para la descarga asi como los que ya han sido descargados, se muestra la versión y la fecha. El usuario selecciona las aplicaciones que desee descargar y el sistema descarga los paquetes que contienen las aplicaciones al nodo objetivo LinkAll.

2.5.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo objetivo.

2.5.2 Flujo Principal

SNA/TNA	SISTEMA
1. Selecciona la opción de descarga de paquetes	
	2. Se conecta al servidor de descargas
	3. Obtiene los descriptores de los paquetes disponibles para la descarga. junto con sus dependencias.
	4. Muestra los paquetes existentes en el nodo soporte indicando si se encuentran o no en el nodo.
5. Selecciona los paquetes a descargar y luego selecciona la opción “descargar”.	
	6. Verifica que se cumplan las dependencias.
	7. Descarga los paquetes seleccionados en el directorio destinado al almacenamiento de los paquetes locales (repositorio local). Verifica la autenticidad de cada paquete.
	8. Se actualizan los descriptores de paquetes en el nodo objetivo agregando los paquetes descargados.
	9. Le comunica al usuario que la descarga se completo con éxito.
	10. Termina el caso de uso.

2.5.3 Flujo Alternativo

2.A No se puede conectar con el servidor de descargas

2.A.1 El sistema le comunica al usuario que no se pudo establecer la conexión

2.A.2 Termina el caso de uso

3.A No puede obtener los descriptores de los paquetes disponibles para la descarga

3.A.1 El sistema le comunica al usuario que hubo algún error al obtener la descripción de los paquetes

3.A.2 Termina el caso de uso.

6.A Falla la verificación de dependencias

6.A.1 El sistema le comunica al usuario que no puede descargar los paquetes por falla en las dependencias

6.A.2 Termina el caso de uso

7.A No puede descargar los paquetes

7.A.1 El sistema le comunica al usuario que no puede descargar los paquetes

7.A.2 Termina el caso de uso

7.B Falla la autenticación del paquete

7.B.1 El sistema le comunica al usuario que fallo la autenticación

7.B.2 El sistema elimina el paquete descargado

7.B.3 Termina el caso de uso

G) El usuario puede cancelar en cualquier momento

2.6 Deployment

El TNA selecciona la opción “Deployment”, selecciona las aplicaciones a instalar y el sistema instala la aplicación retornándole al TNA un reporte del proceso.

2.6.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo objetivo.

El paquete conteniendo la aplicación debe estar previamente descargado.

2.6.2 Flujo Principal

SNA/TNA	SISTEMA
1. Selecciona la opción “Deploy”	
	2. Obtiene los descriptores de los paquetes disponibles en el nodo objetivo.
	3. Incluye el caso de uso “Navegación entre paquetes”.
4. Selecciona las aplicaciones a instalar	
	5. Verifica que para todas las aplicaciones seleccionadas por el usuario para instalar, estén instaladas las aplicaciones de las cuales depende o que estas (las que no estén instaladas) hayan sido seleccionadas por el usuario para instalar
	6. Ejecuta las acciones de instalación de la aplicación
	7. Le comunica al usuario que termino la operación con éxito y le muestra un reporte del proceso
	8. Termina el caso de uso

2.6.3 Flujo Alternativo

2.A No hay paquetes descargados

2.A.1 Le comunica al usuario que no hay aplicaciones para instalar

2.A.2 Termina el caso de uso

5.A Falla la verificación

5.A.1 El sistema le avisa al usuario que aplicaciones no pueden ser instaladas porque faltan instalar aplicaciones de las cuales depende y estas no han sido seleccionadas para instalar. Le muestra que aplicaciones faltan instalar

5.A.3 Vuelve a 2

6.A Fallo la ejecución de alguna de las acciones de instalación

6.A.1 El sistema registra el error en el reporte, realiza un rollback de esa instalación y continua con la instalación del resto de las aplicaciones

6.A.2 Le comunica al usuario que hubo problemas durante la operación, le muestra un reporte

6.A.3 Termina el caso de uso

G) El usuario puede cancelar en cualquier momento

2.7 Undeployment

El TNA selecciona la opción “Undeployment”, selecciona las aplicaciones a desinstalar y el sistema desinstala las aplicaciones retornándole al TNA un reporte del proceso. El TNA tiene la opción de desinstalar todas las aplicaciones que dependen de esta aplicación ya que no funcionarán correctamente luego del undeployment; lo mismo se hará extensivo para aquellas aplicaciones que dependen transitivamente.

2.7.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo objetivo.

2.7.2 Flujo Principal

SNA/TNA	SISTEMA
1. Selecciona la opción de undeploy	
	2. Obtiene los descriptores de las aplicaciones instaladas
	3. Incluye el caso de uso “Navegación entre paquetes”.
4. Selecciona las aplicaciones a desinstalar	
	5. Chequea que luego de las desinstalaciones no hayan aplicaciones instaladas que dependan de alguna aplicación de las que se van a desinstalar
	6. Ejecuta las acciones de desinstalación para las aplicaciones seleccionadas
	7. Le comunica al usuario que termino la operación con éxito
	8. Termina el caso de uso

2.7.3 Flujo Alternativo

2.A No hay aplicaciones instaladas

2.A.1 Le comunica al usuario que no hay aplicaciones instaladas

2.A.2 Termina el caso de uso

5.A Falla la verificación de dependencias

5.A.1 Le comunica al usuario que aplicaciones son las que no se pueden desinstalar

5.A.2 Termina el caso de uso

6.A No se puede desinstalar una aplicación u ocurrió algún error durante la desinstalación

6.A.1 El sistema registra el error en el reporte y continua con la desinstalación del resto de las aplicaciones

6.A.2 Le comunica al usuario que hubo problemas durante la operación, le muestra un reporte

6.A.3 Termina el caso de uso

G) El usuario puede cancelar en cualquier momento

2.8 Upgrade de aplicaciones

El TNA selecciona la opción upgrade, selecciona la aplicación a la que le va a hacer el upgrade, selecciona la ubicación del paquete que contiene el upgrade y el sistema instala la nueva versión retornándole al TNA un reporte del proceso.

2.8.1 Pre-Requisito

El paquete conteniendo el upgrade debe estar previamente descargado.

2.8.2 Flujo Principal

SNA/TNA	SISTEMA
1. Selecciona la opción upgrade	
	2. Obtiene los descriptores de las aplicaciones
	3. Incluye el caso de uso “Navegación entre paquetes”.
4. Selecciona la aplicación a la cual le quiere hacer el upgrade	
	5. Desinstala la versión instalada
	6. Instala la nueva versión
	7. Le comunica al usuario que termino la operación con éxito
	8. Termina el caso de uso

2.8.3 Flujo Alternativo

2.A No hay aplicaciones instaladas

2.A.1 Le comunica al usuario que no hay aplicaciones instaladas

2.A.2 Termina el caso de uso

G) El usuario puede cancelar en cualquier momento

2.9 Configuración del nodo objetivo

El TNA selecciona la opción de “Configurar”, cambia los datos que desee y acepta los cambios, el sistema aplica el cambio en la configuración. Lo que se puede configurar es:

- dirección de acceso al servidor de descargas
- dirección de almacenamiento local de paquetes
- dirección de almacenamiento de especificaciones de deployment
- dirección del almacenamiento temporal
- usuario, password y string de conexión a la base de datos que contiene la información sobre los recursos y los links entre ellos.
- mantenimiento de enlaces entre recursos

2.9.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo objetivo.

2.9.2 Flujo Principal

SNA/TNA	SISTEMA
1. Selecciona la opción de “Configurar”	
	2. Muestra la configuración actual para el nodo objetivo
3. Modifica la configuración	
4. Acepta los cambios	
	5. Actualiza la configuración para el nodo objetivo
	6. Termina el caso de uso

2.9.3 Flujo Alternativo

5.A El sistema no puede aplicar los cambios

5.A.1 El sistema notifica al usuario sobre la imposibilidad de aplicar los cambios detallando el por qué.

5.A.2 El sistema vuelve a 2.

G) El usuario puede cancelar en cualquier momento

2.10 Configuración del nodo soporte

El SNA selecciona la opción de "Configurar dirección de almacenamiento", ingresa la nueva dirección para la carpeta de almacenamiento de paquetes y acepta los cambios, el sistema toma esta dirección como la dirección por defecto para el almacenamiento de los paquetes.

2.10.1 Pre-Requisito

El usuario se encuentra correctamente autenticado en el nodo soporte.

2.10.2 Flujo Principal

SNA	SISTEMA
1. Selecciona la opción de "Configurar dirección de almacenamiento"	
	2. Muestra la dirección actual de la carpeta de almacenamiento
3. Modifica la dirección de almacenamiento	
4. Acepta los cambios	
	5. Toma la dirección ingresada como la dirección por defecto para el almacenamiento de los paquetes
	6. Termina el caso de uso

2.10.3 Flujo Alternativo

5.A El sistema no puede aplicar los cambios

5.A.1 El sistema notifica al usuario sobre la imposibilidad de aplicar los cambios detallando el por qué.

5.A.2 El sistema vuelve a 2.

G) El usuario puede cancelar en cualquier momento

UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

ANEXO C

DESCRIPCIÓN DE LA ARQUITECTURA

Estudiantes
Nicolás Doroskevich
Sebastián Moreira

Tutores
Federico Piedrabuena
Raúl Ruggia

Proyecto de grado 2004
Facultad de Ingeniería - Universidad de la República

Índice general

1 Introducción

El proyecto “Utilitarios para deployment de aplicaciones en un servidor basado en J2EE” forma parte del proyecto LinkAll [1, 2], teniendo como objetivo estudiar y construir herramientas que permitan realizar tareas de deployment en un servidor J2EE como el que se utilizará para contener las diversas aplicaciones que se ejecutarán dentro de la plataforma LinkAll en uno de sus nodos aplicativos [4]. En tal sentido, el sistema a desarrollar, deberá resolver el problema planteado de tal manera de que pueda ser utilizado en la plataforma LinkAll pero sin dejar de lado la intención de que como resultado se construya una herramienta de uso genérico y aplicable en otros contextos similares.

El presente documento provee una vista de alto nivel de la arquitectura del sistema de deployment de aplicaciones en un servidor J2EE, incluyendo objetivos y restricciones, los casos de uso más significativos, los patrones de arquitectura aplicados y las principales decisiones de diseño.

1.1 Propósito

Este documento tiene como objetivo brindar al lector un panorama general del sistema a construir, mediante la presentación de una serie de vistas de la arquitectura que detallan diferentes aspectos del mismo. También tiene como propósito primordial capturar y cubrir las decisiones más significativas, que se hayan tomado.

Está dirigido a cuatro distintos tipos de actores: desarrolladores de aplicaciones J2EE, desarrolladores LinkAll, docentes supervisores del proyecto y desarrolladores de éste sistema de deployment.

Un desarrollador de aplicaciones J2EE puede utilizar este documento para entender cómo se estructura y organiza la herramienta, y así facilitar la comprensión acerca de cómo se utiliza, cómo se extiende y cómo aplicarla a contextos similares al de la plataforma LinkAll.

Los desarrolladores LinkAll pueden utilizar este documento para entender la arquitectura de uno de los componentes de la plataforma. Por otra parte, pueden hacer uso del mismo bajo similares motivaciones que un desarrollador de aplicaciones J2EE, comprendiendo cómo hacer que una aplicación para la plataforma sea capaz de instalarse mediante el uso del componente de deployment (que sea *deployable* - en inglés -, para contar con un término más corto).

Los desarrolladores del sistema de deployment pueden encontrar en este documento una referencia continua para el diseño en más bajo nivel y para la propia implementación de mismo. Por otra parte, puede ser de utilidad para la asignación de tareas en función de la estructura y organización del sistema, y para comprender el avance en función de las subpartes del mismo que se vayan implementando.

Los docentes supervisores podrán contar con un panorama en alto nivel, lo cual les puede ser de interés para la comprensión del producto, para entender el cronograma y la asignación de tareas del equipo de desarrollo y para poder determinar el avance en función de las subpartes del sistema que se vayan implementando. Es importante destacar que, para el caso de este proyecto, los docentes supervisores son los propios clientes y por lo tanto se los podría denominar de esta forma.

1.2 Alcance

El proyecto incluye la construcción de utilitarios que faciliten las tareas relacionadas al problema de deployment explicado brevemente al inicio del presente documento. Una herramienta como éstas puede ser un empaquetador de aplicaciones en donde el usuario pueda reunir en un único elemento todos los archivos de una aplicación deployable y realizar tareas de configuración que le permitan incluir una descripción de la aplicación, sus dependencias, entre otras cosas. En el presente documento no se incluirá ningún tipo de información detallada acerca de utilitarios de deployment que se puedan elaborar y solo se explayará en temas vinculados a lo que el equipo considera más relevante, es decir, se describirá el motor de deployment responsable de instalar aplicaciones en un servidor J2EE, y el mecanismo de obtención de paquetes (conteniendo aplicaciones) desde un servidor que exponga y permita obtener los mismos. Esta decisión se debe a que el equipo entiende que cualquier aplicación de éste tipo nunca será suficientemente grande y compleja como para que amerite contar con una descripción de alto nivel de su arquitectura (probablemente se estaría abarcando toda la aplicación). Por otro lado, la inclusión de esta información no ayudaría al lector a comprender los elementos más importantes del sistema a construir.

1.3 Definiciones, abreviaciones y acrónimos

Es necesario que el lector tenga conocimiento de ciertas definiciones relevantes para la comprensión de este documento. Así mismo, es de utilidad manejar abreviaciones y acrónimos. Todos estos elementos se encuentran definidos en el Glosario del proyecto (ver [9]) y en particular, dentro de este documento destacamos los siguientes elementos:

- ***DP (Deployable package)***
- ***AD (Application descriptor)***
- ***DS (Deployment specification)***
- ***RS (Resources specification)***
- ***Nodo cliente***
- ***Nodo soporte***
- ***Nodo objetivo***

En relación a los casos de uso, los actores y su respectivas definiciones se encuentran detalladas en el Modelo de casos de uso (RMC) (ver [5]).

1.4 Referencias

Es de interés incluir las siguientes referencias:

[1] Sitio oficial del proyecto LinkAll (2003)
http://www.link-all.org

[2] Descripción del proyecto LinkAll (2003)
http://www.fing.edu.uy/inco/proyectos/linkall/Documentos/Descripcion-LinkAll.doc
INCO – Facultad de Ingeniería – UdelaR

[3] Documento de requerimientos del sistema
Código: RDR (v1.5)
http://www.fing.edu.uy/~pgdploy1/docs/requerimientos/RDRv1.5.zip
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[4] Documento de descripción de la arquitectura LinkAll (v1.7.4 - Enero 2005)
INCO – Facultad de Ingeniería - UdelaR

[5] Documento de casos de uso
Código: RMC (v1.5)
http://www.fing.edu.uy/~pgdploy1/docs/requerimientos/RMCv1.5.zip
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[6] Doc. técnica – Descriptor de aplicaciones
Código: IDT-DESCR (v1.1)
http://www.fing.edu.uy/~pgdploy1/docs/implementacion/IDT-DESCRv1.1.zip
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[7] Doc. técnica – Especificación de Deployment
Código: IDT-DEPSPEC (v1.1)
http://www.fing.edu.uy/~pgdploy1/docs/implementacion/IDT-DEPSPECv1.1.zip
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[8] Doc. técnica – Especificación de Recursos
Código: IDT-RESSPEC (v1.1)
http://www.fing.edu.uy/~pgdploy1/docs/implementacion/IDT-RESSPECv1.1.zip
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[9] Glosario
Código: RG (v1.2)
http://www.fing.edu.uy/~pgdploy1/docs/requerimientos/RGv1.2.zip
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[10] Unified Modeling Language, OMG (2004)
http://www.omg.org/uml

[11] Architectural Blueprints – The "4+1" View Model of Software Architecture
P. Krutchen
Racional Software Corp. (1995)

1.5 Organización

El presente documento se estructura de la siguiente manera: en el capítulo 3 se incluyen metas y restricciones de la arquitectura, el capítulo 4 se describe cómo será la representación de la arquitectura la cual es desarrollada posteriormente en los siguientes capítulos.

En el capítulo 5 se incluye la vista de casos de uso del sistema, y en el capítulo 6 se muestra la trazabilidad entre los casos de uso presentados y los conceptos de diseño incluidos en la vista lógica que se encuentra detallada en el capítulo 7.

En el capítulo 8 se incluye la trazabilidad de los conceptos de diseño incluidos en la vista lógica hacia los componentes de la vista de implementación la cual es detallada en el capítulo 9.

Finalmente en el capítulo 10 se incluye la vista de deployment,

2 Metas y Restricciones de la Arquitectura

Existen ciertos requerimientos de gran importancia a nivel de la arquitectura, así también como ciertas restricciones que influyen en ella. A continuación se listan dichos requerimientos y restricciones:

- Permitir el deployment/undeployment de aplicaciones en un servidor J2EE siguiendo la especificación de deployment (DS).
- Resolver enlaces que permitan que una aplicación recién instalada se pueda conectar con otros recursos, entendiéndose como recurso a cualquier otro elemento que se desee que las aplicaciones tengan acceso. Los enlaces son definidos en un archivo de especificación de recursos (RS).
- Brindar un mecanismo de instalación remota de tal manera de que un usuario administrador del servidor pueda indicar qué nuevos aplicativos instalar. El sistema deberá controlar temas vinculados con el manejo de versiones y control de dependencias entre aplicaciones.
- Se deberá automatizar el deployment de tal forma de que ésta tarea pueda ser realizada por usuarios no expertos en los propios aplicativos a instalar.
- Se deberá proveer de aplicaciones que permitan monitorear el estado de las aplicaciones, principalmente al momento de deployment y undeployment, lo cual permita al usuario administrador encontrar errores y buscar soluciones.
- Aunque el sistema a construir se desea que sea genérico y aplicable a otros problemas similares a LinkAll, éste deberá dar soluciones para la mencionada plataforma y por lo tanto se encuentra influenciado por las diferentes decisiones arquitectónicas, y quizás de más bajo nivel, que surjan como parte del desarrollo de la misma.

3 Representación de la Arquitectura

El sistema de deployment es una aplicación utilitaria embebida en un servidor J2EE, la cual permite la obtención de aplicaciones desde un servidor que oficia de repositorio y que posteriormente permite la instalación de las mismas en el servidor objetivo.

Debido a la naturaleza del problema, el sistema cuenta con una serie reducida de casos de uso, más aún, al tomar los más relevantes para que ilustren la arquitectura del sistema, éstos son aún menos. El sistema entonces cuenta con tan solo dos casos de uso relevantes. Por otro lado, la complejidad de la herramienta hace que a pesar de la muy reducida cantidad de casos de uso relevantes, realizarlos resulte en una tarea bastante complicada. Debido a esto, la vista lógica será la más extensa.

La arquitectura está representada mediante la inclusión de diferentes vistas utilizando la notación UML [10] y siguiendo el framework arquitectónico "4+1" presentado en [11]. De esta manera, se puede visualizar, comprender y razonar la arquitectura desde diversas perspectivas. En la siguiente sección de describen brevemente las vistas utilizadas.

3.1 Vistas de la arquitectura

Como se mencionó anteriormente, la arquitectura se representa en términos de vistas en donde la vista de casos de uso ilustra las restantes, tal como se sugiere en [11]. Las vistas incluidas más una breve descripción son detalladas a continuación.

- **Vista de los Casos de Uso**
  Presenta los actores y los casos de uso más relevantes para la arquitectura.

- **Vista Lógica**
  Describe la arquitectura del sistema presentando varios niveles de refinamiento. Indica los módulos lógicos principales, sus responsabilidades y dependencias.

- **Vista de Implementación**
  Describe los componentes de deployment construidos y sus dependencias.

- **Vista de Deployment**
  Presenta aspectos físicos como topología, infraestructura informática, e instalación de ejecutables. Incluye además plataformas y software de base.

4 Vista de los Casos de Uso

4.1 Casos de Uso Relevantes a la Arquitectura

En esta sección presentamos los casos de uso relevantes para la Arquitectura del sistema. A continuación, se incluye el correspondiente diagrama de casos de uso, mostrando aquellos relevantes y los actores que intervienen en ellos.

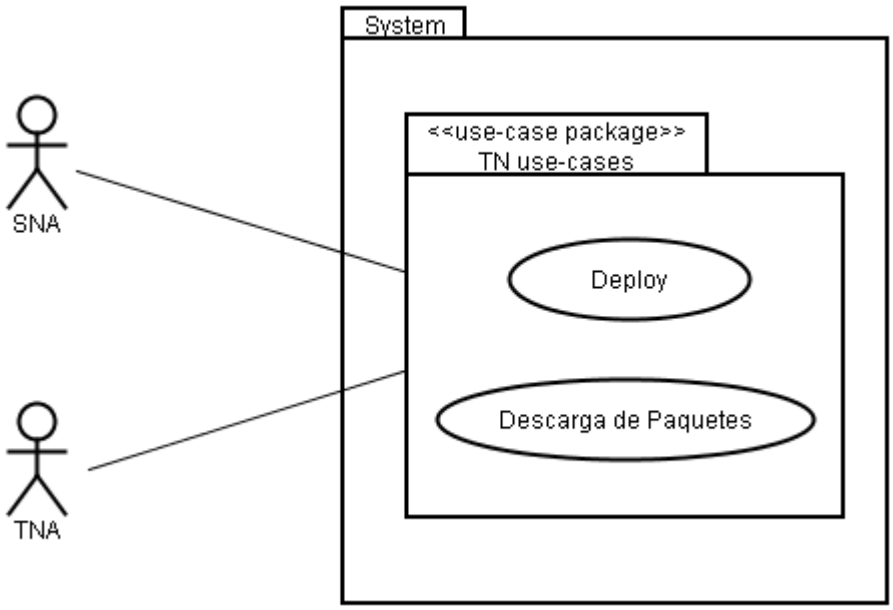


Diagrama 5.1-1 – CU's relevantes para la arquitectura

4.1.1 Deploy

El actor SNA o el TNA (Administrador del nodo soporte o Administrador del nodo objetivo) selecciona la opción "Deployment", selecciona las aplicaciones a instalar y el sistema instala la aplicación retornándole al actor un reporte del proceso. El sistema ejecutará las acciones definidas en la DS y registrará la información referente a los recursos por medio de la RS. También controlará los prerrequisitos y advertirá al usuario en caso de no cumplirse con las condiciones de instalación, dando la opción de continuar a pesar de que eso suceda.

4.1.2 Descarga de paquetes

El actor SNA o el TNA (Administrador del nodo soporte o Administrador del nodo objetivo) se conecta al servidor de descargas utilizando la opción de descargas, y muestra las aplicaciones que puede descargar junto con una descripción de la funcionalidad de la aplicación. También se muestra para cada aplicación la lista de aplicaciones que necesita para ejecutarse correctamente. El usuario selecciona las aplicaciones que desee descargar y el sistema descarga los paquetes que contienen las aplicaciones al nodo objetivo. También controlará los prerrequisitos y advertirá al usuario en caso de no cumplirse con las condiciones para la posterior instalación, dando la opción de continuar a pesar de que eso suceda.

5 Trazabilidad de Vista de Casos de Uso a la Vista Lógica

En esta sección se muestra, mediante un diagrama, la trazabilidad entre los Casos de Uso relevantes a la Arquitectura y los paquetes encontrados en el Análisis.

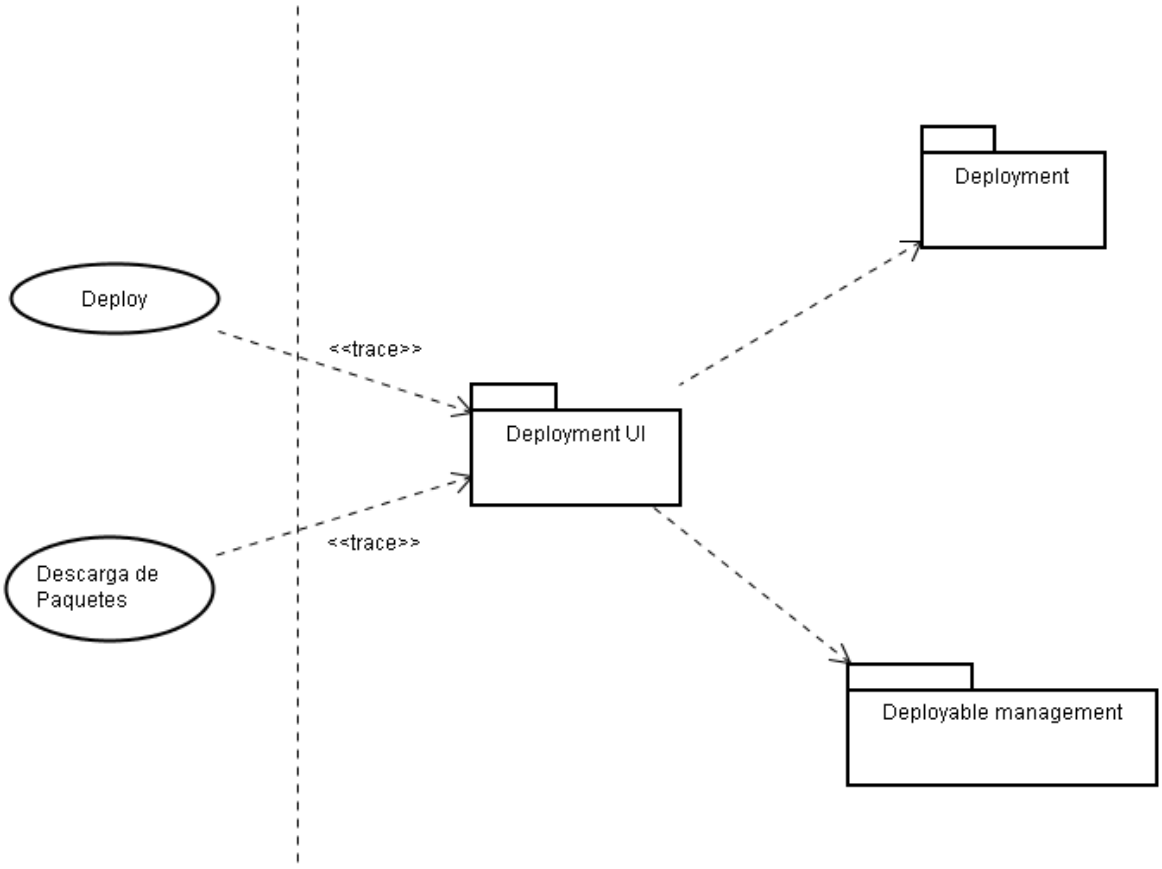


Diagrama 6-1 –Trazabilidad desde el modelo de casos de uso al modelo de análisis

6 Vista Lógica

Esta vista presenta la arquitectura para el sistema de deployment, presentada desde diferentes niveles de abstracción. Cada nivel corresponde a un refinamiento del nivel anterior, en donde el primer nivel presenta la arquitectura en términos de patrones de arquitectura, el siguiente nivel introduce los subsistemas participantes en el sistema, y finalmente en el último nivel de abstracción se presentan más detalladamente las principales clases intervinientes y sus interfaces, incluyendo diagramas de interacción que ilustren el comportamiento de los mismos.

Esta sección se organiza de la siguiente manera:

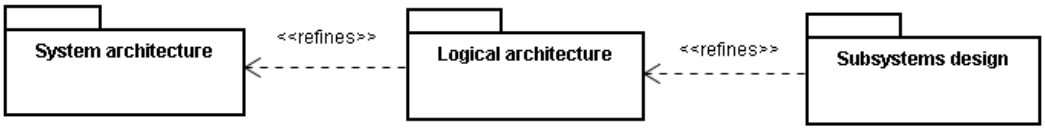


Diagrama 7-1 – Organización de la vista lógica

6.1 Arquitectura del sistema

En esta sección se presenta al sistema desde el nivel más alto de abstracción. En tal sentido se describe la estructura del mismo en términos de patrones de arquitectura.

El sistema de deployment se organiza siguiendo un patrón de arquitectura en capas puro. Por tratarse de una arquitectura en capas, cada capa podrá acceder solamente a los servicios brindados por capas inferiores, y por tratarse de una aplicación pura del patrón cada capa accede solamente a la capa inmediata inferior.

Para el caso del sistema a construir, la división cuenta con tres capas organizadas de la siguiente manera:

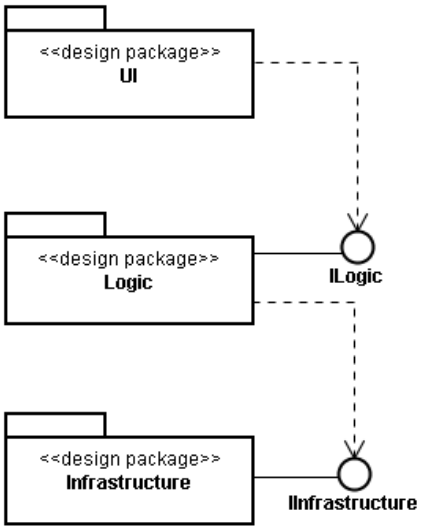


Diagrama 7.1-1 – Arquitectura en capas

La capa **UI** se corresponde con la interfaz de usuario. En esta capa es donde se resuelven los diálogos con el usuario y que le permiten realizar los diversos casos de uso.

En la capa **Logic (Lógica)** reside la mayor parte importante del sistema. Resuelve y expone todas las funcionalidades de deployment que el sistema brinda.

Finalmente, la capa **Infrastructure (Infraestructura)** brinda servicios que permiten abstraer el almacenamiento de la información persistente, la comunicación, entre otras tareas de más bajo nivel.

6.2 Arquitectura lógica

Una vez presentada la arquitectura en más alto nivel, en esta sección se describe la misma con un mayor nivel de detalle, comentando muy brevemente los principales subsistemas participantes y cuáles son sus responsabilidades más destacables. Un nivel aún más bajo de abstracción se incluye en la sección 7.3.

6.2.1 Descomposición en subsistemas

A continuación, se muestra la descomposición del sistema en términos de sus subsistemas más relevantes para la Arquitectura y las dependencias entre ellos, además para cada uno se muestra la Capa de la Arquitectura en la que se encuentra (todos los subsistemas poseen el estereotipo `<<design subsystem>>` pero no se visualizan para una mayor legibilidad del diagrama).

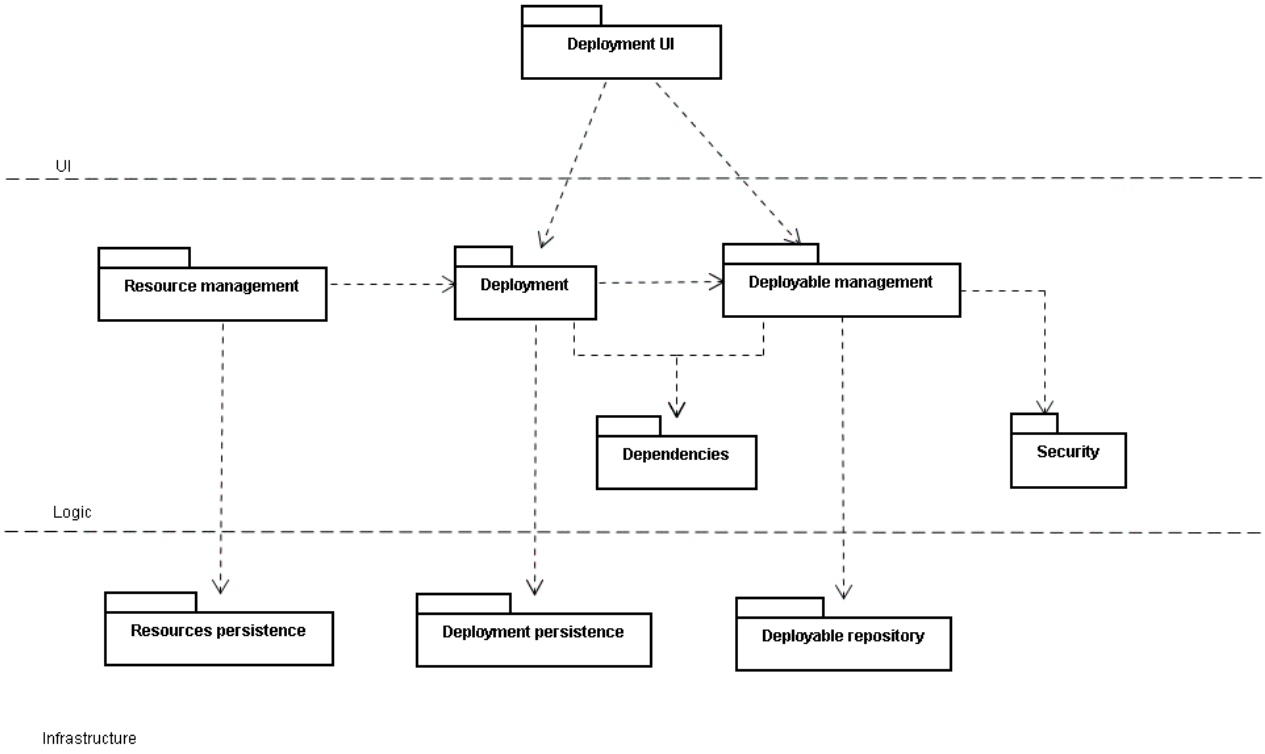


Diagrama 7.2-1 – Descomposición en subsistemas

6.2.1.1 Deployment UI

Interfaz de usuario con funciones de manejo de paquetes deployables. Realiza todos los casos de uso que aparecen en vista de casos de uso. Se basa en el patrón MVC.

- Utiliza el paquete ***Deployment*** para indicar el deployment/undeployment de paquetes. Además permite consultar al sistema acerca de las aplicaciones instaladas.
- Utiliza ***Deployable management*** para indicar descargas a un nodo objetivo considerando las dependencias (prerrequisitos).

6.2.1.2 Deployable management

Este subsistema puede dividirse en dos partes en función del rol que posea el nodo en donde se encuentre funcionando. Si forma parte de un nodo soporte, es responsable de exponer los DP's disponibles para ser descargados. Si forma parte de un nodo objetivo, es responsable de la descarga de DP's y de la gestión de aquellos paquetes ya descargados.

- Utiliza ***Deployable repository*** para almacenar paquetes deployables. Además podrá utilizarlo para acceder al contenido de los DP's, como por ejemplo el descriptor de la aplicación (AD) a instalar.

- Utiliza ***Dependencies*** para obtener información de las dependencias que poseen los paquetes y posibilitar el control de las mismas.
- Utiliza ***Security*** para controlar la validez de los paquetes en función de las firmas que posean. Además se utiliza para autenticar el nodo objetivo con el nodo de descargas (ver *sección 9* por más detalles al respecto de los nodos del sistema).
- Desde la perspectiva del nodo objetivo, utiliza ***Deployable management*** en el nodo soporte para obtener los paquetes almacenados en el repositorio de dicho nodo.

6.2.1.2.1 Principales interfaces

En esta parte se detallan las principales interfaces expuestas por el subsistema.

- **ISupportDeployableAdmin**
  Responsable de administrar y exponer los paquetes disponibles para descarga en caso de tratarse de un nodo soporte.
- **ITargetDeployableAdmin**
  Responsable de la gestión del repositorio de DP's en el nodo objetivo y de brindar acceso a la información contenida en cada uno de sus descriptores de aplicación.

6.2.1.3 Deployment

Subsistema responsable de la instalación de aplicaciones. Al momento de deployment, permite realizar una serie de acciones especificadas en la especificación de deployment (DS), y realiza similar tarea al indicarse un undeployment de aplicaciones.

- Utiliza ***DeployableManagement*** para tener acceso a los paquetes descargados y sus datos descriptivos.
- Utiliza ***Dependencies*** para el control de dependencias que cada aplicación a instalar posea..
- Utiliza ***Deployment persistence*** para encapsular el mecanismo de persistencia de la información referente al deployment de aplicaciones/paquetes.

6.2.1.3.1 Principales interfaces

En esta parte se detallan las principales interfaces que expone el subsistema.

- **IDeployer**
  Responsable de la ejecución de las acciones especificadas en la DS de cada aplicación al momento de deployment y de realizar otra serie de acciones al momento de undeployment (también especificadas en el DS).

6.2.1.4 Dependencies

Subsistema encargado del chequeo de dependencias entre las aplicaciones. Revisa que se puedan instalar aplicaciones en función del cumplimiento de los prerrequisitos.

6.2.1.4.1 Principales interfaces

En esta parte se detallan las principales interfaces que expone el subsistema.

- **IDependenciesChecker**
  Esta clase se encarga de verificar el cumplimiento de las precondiciones declaradas en el AD de cada aplicación deployable. Para hacerlo, deberá tener en cuenta una serie de aplicaciones a instalar y una descripción general de las aplicaciones ya instaladas.

6.2.1.5 Resource management

Subsistema encargado de manejar los recursos declarados al momento de deployment y proveer mecanismos para que las aplicaciones puedan resolver los enlaces a otros recursos.

- Utiliza ***Resources persistence*** para encapsular el mecanismo de persistencia de la información referente a recursos requeridos y provistos por parte de las aplicaciones.
- Utiliza ***Deployment*** puesto que mediante su API contribuye al sistema con un tipo de acción a ejecutar al momento de deployment, la cual es responsable de procesar la especificación de recursos (RS). Dicha acción se denomina ***ManageResources***.

6.2.1.5.1 Principales interfaces

En esta parte se detallan las principales interfaces que expone el subsistema.

- **IResourceManager**
  Clase que oficia de punto de entrada al paquete y a todas sus funcionalidades. Es responsable de registrar los recursos declarados (requeridos o provistos) por las aplicaciones al momento de deployment (contenidos en el RS).
- **IResourceAccessSolver**
  Esta clase permite a las aplicaciones obtener el acceso a los diversos tipos de recursos requeridos por las mismas. Utiliza la información registrada por el ***ResourceManager*** para resolver estos accesos.

6.2.1.6 Security

Subsistema con utilitarios relacionados con la seguridad. Provee funcionalidades que le permiten al ***Deployment management*** chequear la autenticidad de los paquetes descargados y que se encuentran prontos para ser deplorados. Además brinda funciones para certificar el nodo objetivo en el nodo soporte al momento de requerir información del repositorio de paquetes.

6.2.1.7 Deployment persistence

Contiene funcionalidades vinculadas con la persistencia de la información referente al deployment de aplicaciones.

6.2.1.8 Resources persistence

Contiene funcionalidades vinculadas con la persistencia de la información acerca de cuáles son los recursos provistos y requeridos por las aplicaciones deployadas.

6.2.1.9 Deployable repository

Subsistema que implementa un repositorio de paquetes deployables. Provee mecanismos para poder agregar y borrar paquetes, más funcionalidades que permitan acceder a la información contenida en ellos (descriptores, archivos, etc.).

6.2.2 Análisis de Casos de Uso

6.2.2.1 Descarga de paquetes

En el siguiente diagrama se muestran todos los subsistemas intervinientes en el caso de uso:

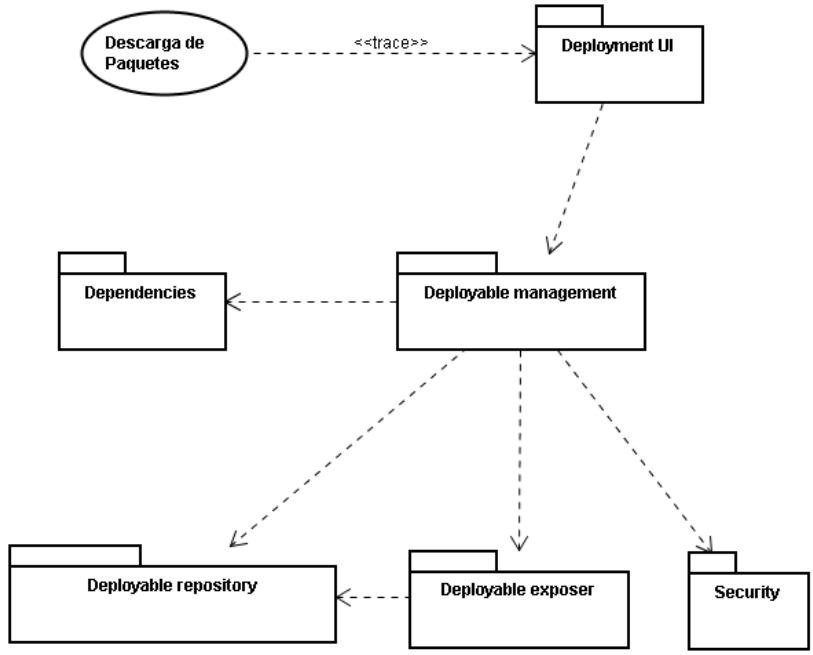


Diagrama 6.2-1 – Subsistemas intervinientes en el CU "descarga de paquetes".

6.2.2.2 Deploy

En el siguiente diagrama se muestran todos los subsistemas intervinientes en el caso de uso:

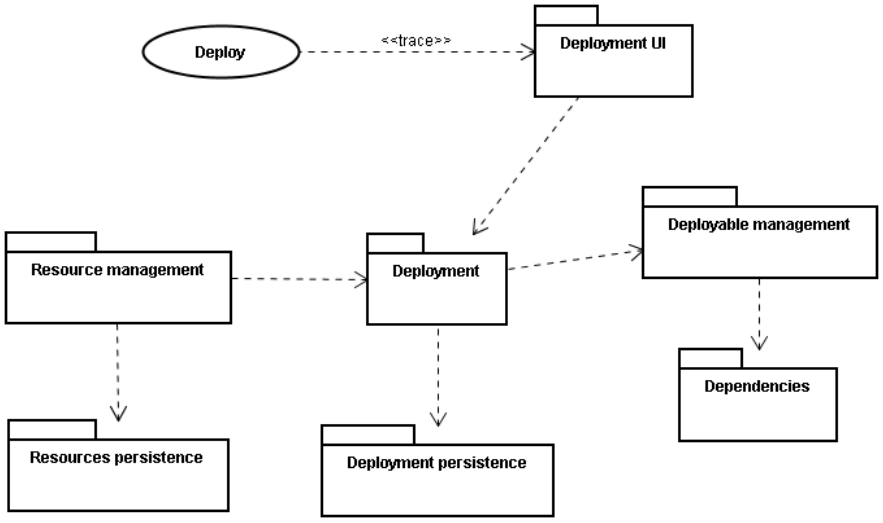


Diagrama 6.2-2 – Subsistemas intervinientes en el CU "deploy".

6.3 Diseño de subsistemas

En esta sección se incluye una descripción en más bajo nivel de abstracción de los principales subsistemas presentados en la sección 7.2. En cada sección se presenta una descripción del subsistema, sus interfaces, los principales elementos que lo componen (interfaces y clases), y cómo interactúan entre ellos y con los demás subsistemas para resolver aquellos servicios más relevantes en términos de la Arquitectura.

6.3.1 Deployable management

El subsistema es responsable de la administración y descarga de ***Deployable packages (DP)*** en un nodo objetivo, así como también la administración de DPs en el nodo soporte. Para cumplir con su función, el subsistema utiliza el archivo ***Application descriptor (AD)*** para el chequeo de dependencias tanto al descargar paquetes del nodo soporte como al subir aplicaciones al mismo. En esta sección se detalla la forma en que interactúa el subsistema con otros para resolver las diversas funcionalidades.

6.3.1.1 Interfaces

Las interfaces identificadas son presentadas a continuación.

Diagrama 7.3-1a – Interfaz *ISupportDeployableAdmin*

La interfaz ***ISupportDeployableAdmin*** representa el punto de acceso a los distintos servicios vinculados con la administración de paquetes en el nodo soporte. Posee operaciones para el registro de paquetes, eliminación de paquetes y para consultar cuáles paquetes están disponibles en el nodo. Un resumen de las operaciones aparece a continuación:

Operación	**RegisterPackage**
Parámetros	packName – Nombre del paquete a registrar en el nodo (Strings).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Registra un paquete existente el repositorio del nodo. Busca el paquete cuyo nombre fue pasado como parámetro y lo registra. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• El paquete cuyo nombre es pasado como parámetro debe existir en el repositorio del nodo.
Poscondiciones	• El paquete queda registrado en el nodo como disponible para la descarga.

Operación	`RemovePackage`
Parámetros	`id` – Identificador del paquete a eliminar en el repositorio del nodo (Strings).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Desregistra y elimina el paquete del repositorio del nodo. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• El paquete debe existir en el repositorio del nodo y debe estar registrado en el mismo.
Poscondiciones	• El paquete es eliminado del nodo.

Operación	`GetPackages`
Parámetros	No tiene.
Retorno	Estructura conteniendo la información de los paquetes disponibles y registrados en el nodo.
Descripción	Consulta al subsistema acerca de cuáles son los paquetes disponibles en el nodo.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Para las operaciones ***registerPackage***, ***Load*** y ***removePackage***, en caso de producirse errores, estos se registran en el reporte retornado. Todo hecho acontecido al ejecutar estas operaciones es incluido, en forma de nota, advertencia y error, en el reporte que se retorna y mediante el cuál es posible consultar cada uno de los mismos una vez terminada la ejecución de la operación.

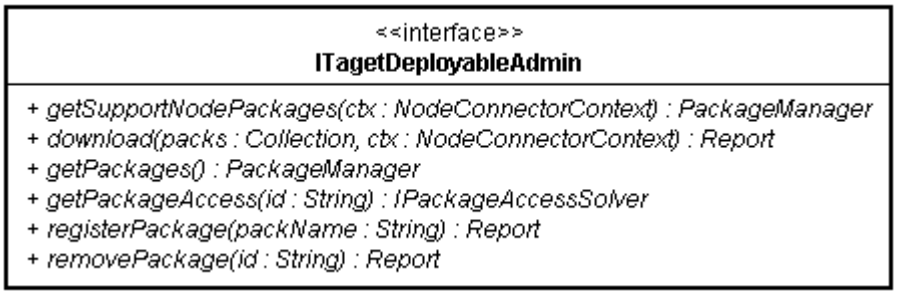


Diagrama 7.3-1b – Interfaz *ITargetDeployableAdmin*

La interfaz ***ITargetDeployableAdmin*** representa el punto de acceso a los distintos servicios vinculados con la administración de paquetes en el nodo objetivo. Posee operaciones para el registro de paquetes, eliminación de paquetes, descarga de paquetes, para consultar cuáles paquetes están disponibles en el nodo soporte y objetivo, y operaciones para acceder al contenido de los paquetes del nodo. Un resumen de las operaciones aparece a continuación.

Operación	`RegisterPackage`
Parámetros	`packName` – Nombre del paquete a registrar en el nodo (Strings).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Registra un paquete existente el repositorio del nodo. Busca el paquete cuyo nombre fue pasado como parámetro y lo registra. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• El paquete cuyo nombre es pasado como parámetro debe existir en el repositorio del nodo.
Poscondiciones	• El paquete queda registrado en el nodo como disponible para deplorar y si el nodo es mirror, también queda disponible para la descarga.

Operación	`RemovePackage`
Parámetros	`id` – Identificador del paquete a eliminar en el repositorio del nodo. (Strings).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Desregistra y elimina el paquete del repositorio del nodo. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• El paquete debe existir en el repositorio del nodo y debe estar registrado en el mismo.
Poscondiciones	• El paquete es eliminado del nodo.

Operación	`GetPackages`
Parámetros	No tiene.
Retorno	Estructura conteniendo la información de los paquetes disponibles y registrados en el nodo.
Descripción	Consulta al subsistema acerca de cuáles son los paquetes disponibles en el nodo.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Operación	`Download`
Parámetros	`packs` – Colección con los identificadores de los paquetes a descargar. (Strings). `ctx` – contexto para el conector al nodo soporte (dicho conector se verá a continuación).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Descarga los paquetes cuyos identificadores son pasados como parámetros y los registra en el nodo. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• Los paquetes cuyos identificadores son pasados como parámetros deben estar registrados en el nodo desde donde se van a descargar.
Poscondiciones	• El paquete queda en el repositorio del nodo y registrado en el mismo como disponible para deplorar. Si el nodo es mirror también queda registrado para la descarga.

Operación	GetPackageAccess
Parámetros	id – Identificador del paquete al cual se desea acceder. (String).
Retorno	Objeto que encapsula el acceso a un paquete dado.
Descripción	Retorna un objeto que provee acceso al paquete cuyo identificador es pasado como parámetro.
Excepciones	No tiene.
Precondiciones	• El paquete debe existir en el repositorio del nodo y debe estar registrado en el mismo.
Poscondiciones	• El objeto devuelto provee el acceso al paquete solicitado.

Operación	GetSupportNodePackages
Parámetros	ctx – contexto para el conector al nodo soporte (dicho conector se verá a continuación).
Retorno	Estructura conteniendo la información de los paquetes disponibles y registrados en el nodo al cual se conecta.
Descripción	Consulta al subsistema acerca de cuáles son los paquetes disponibles en el nodo al que se conecta.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Para las operaciones ***registryPack***, ***removePack***, ***Load*** y ***download***, en caso de producirse errores, estos se registran en el reporte retornado. Todo hecho acontecido al ejecutar estas operaciones es incluido, en forma de nota, advertencia y error, en el reporte que se retorna y mediante el cuál es posible consultar cada uno de los mismos una vez terminada la ejecución de la operación.

Las interfaces presentadas permiten acceder a funcionalidades provistas, en lo referente a la administración de paquetes, en cada tipo de nodo. Como fue presentado anteriormente, a través de la interfaz ***ITargetDeployableAdmin***, desde un nodo objetivo es posible consultar al nodo soporte acerca de cuáles paquetes expone y posteriormente se pueden realizar descargas de aquellos paquetes que sean requeridos. El framework dispone de un componente que encapsula estas tareas. Dicho componente se denomina ***Node Connector*** (conector), y su interfaz es la siguiente:

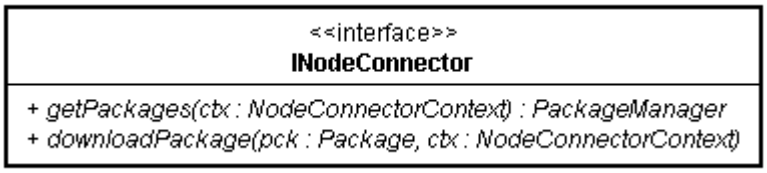


Diagrama 7.3-1c – Interfaces *INodeConnector*

El conector al nodo soporte, resuelve el enlace nodo objetivo-nodo soporte. Implementando adecuadamente este componente, es posible personalizar el mecanismo de comunicación entre los nodos, resolviendo el mismo de diversas maneras y permitiendo remplazarlo, o modificarlo, tantas veces sea requerido y de la forma en que se desee, siempre y cuando se respete la interfaz presentada en el diagrama 7.3-1b.

Las operaciones indicadas en la interfaz ***INodeConnector*** reciben un objeto conteniendo información contextual denominado ***NodeConnectorContext***. Es el punto propicio en donde ubicar toda información adicional requerida para establecer el enlace entre los nodos. Tal es el caso de direcciones de red, URLs, etc. Recordemos la interfaz ***ITargetDeployableAdmin*** (diagrama 7.3-1b). Las operaciones ***getSupportNodePackages*** y ***download*** reciben este contexto inicializado con la información requerida.

Un resumen de las operaciones aparece a continuación.

Operación	GetPackages

Parámetros	`ctx` – contexto para el conector al nodo soporte (dicho conector se verá a continuación).
Retorno	Estructura conteniendo la información de los paquetes disponibles y registrados en el nodo al cual se conecta.
Descripción	Consulta al subsistema acerca de cuáles son los paquetes disponibles en el nodo al que se conecta.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Operación	**`Download`**
Parámetros	`pck` – Información del paquete a descargar. `ctx` – contexto para el conector al nodo soporte (dicho conector se verá a continuación).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Descarga los paquetes cuyos identificadores son pasados como parámetros y los registra en el nodo. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• Los paquetes cuyos identificadores son pasados como parámetros deben estar registrados en el nodo desde donde se van a descargar.
Poscondiciones	• El paquete queda en el repositorio del nodo y registrado en el mismo como disponible para deplorar. Si el nodo es mirror también queda registrado para la descarga.

6.3.1.2 Estructura y comportamiento

Para comprender los elementos que componen el subsistema y como se comportan, es necesario resumir brevemente los pasos que se deben realizar al momento de registrar un paquete en el nodo soporte y descargar un paquete al nodo objetivo; los cuales conforman las principales funcionalidades del subsistema.

Al momento de registrar un paquete, el subsistema realiza básicamente las siguientes tareas:

1. Controlar dependencias basándose en el AD del paquete.
2. Registra el paquete.

Se procede de manera similar al momento de descargar pero en este caso si falla el control de dependencias, el paquete no será descargado.

Opcionalmente, dependiendo de la configuración del nodo, puede ser requerido y ejecutado un control sobre la autenticación del paquete. Esto se realiza verificando que el paquete este firmado y que la firma sea válida.

En función de los pasos descriptos, se pueden identificar necesidades de comunicación con otros subsistemas. Para el control de dependencias se requiere obtener el acceso a un "chequeador" por intermedio del subsistema ***Dependencias.***

- **DOWNLOAD**

El siguiente diagrama de secuencia ilustra el proceso de descarga de paquetes:

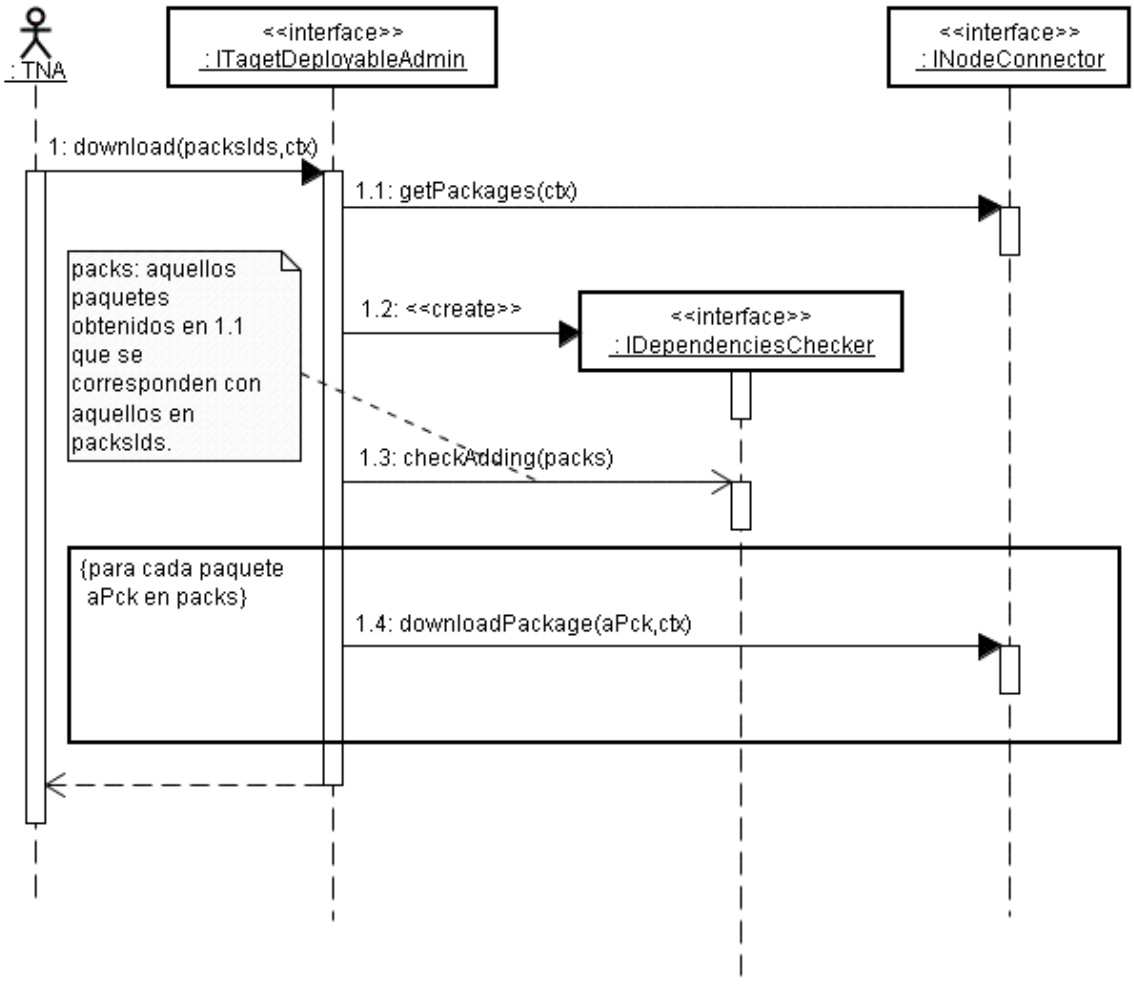


Diagrama 7.3-2 – Secuencia *ITargetDeployableAdmin.download()*

El proceso comienza cuando se solicita al subsistema que descargue un conjunto de paquetes según sus respectivos identificadores. De los paquetes cuyos identificadores son pasados, sólo se seleccionaran para descargar aquellos que no hayan sido previamente descargados al nodo objetivo. La operación de descarga recibe información contextual utilizada por el ***Node Connector*** para establecer el enlace con el nodo soporte que expone los paquetes requeridos.

El primer paso consiste en obtener los AD correspondientes a los paquetes indicados. Para ello es necesario consultar al nodo soporte acerca de cuáles paquetes expone, por lo tanto, utilizando el conector, se procede a consultarle.

Luego, para cada aplicación contenida en los paquetes a descargar, se procede al chequeo de las dependencias contenidas en los mismos. Se utiliza un objeto ***IDependenciesChecker*** para que se chequeen las dependencias de las aplicaciones a descargar. La operación ***checkAdding*** controla que sea posible agregar las aplicaciones indicadas. En caso de no ser satisfechas las dependencias, se genera reporte con los problemas de dependencias por los cuales no se pueden descargar los paquetes y se aborta la descarga reportando el problema.

Si el chequeo de dependencias es satisfecho, entonces se procede a la descarga de los paquetes a través del ***Node Connector***. Dicha labor se realiza mediante la ejecución de la operación ***downloadPackage*** que recibe la información del paquete a descargar, más la información contextual requerida por el conector.

Si durante el proceso de descarga algún paquete no puede ser descargado, no se producen excepciones sino que se reporta cualquier error en el reporte que es retornado por el método.

6.3.2 Deployment

El subsistema de deployment es responsable de la realización de la instalación y desinstalación de aplicaciones en un cierto nodo objetivo. Para ello, se debe contar con paquetes deployables prontos para ser abiertos, y posteriormente para que se instale la aplicación que el mismo contiene. Dentro de cada paquete se encuentran dos archivos que son procesados por el subsistema y que son los de mayor importancia desde este punto de vista. Dichos archivos son el correspondiente al ***Application descriptor (AD)*** y a la ***Deployment specification (DS)***. El primero es utilizado principalmente para el chequeo de dependencias, y el segundo especifica todas aquellas acciones a ejecutar al momento de deployment para la correcta instalación de la aplicación y al momento de undeployment para la eliminación de la misma. En esta sección se detallará cómo se procesan estos elementos y cuál es la forma en que interactúa el subsistema con otros para realizar aquellas funcionalidades relevantes para la arquitectura.

6.3.2.1 Interfaces

Las interfaces identificadas son presentadas a continuación.

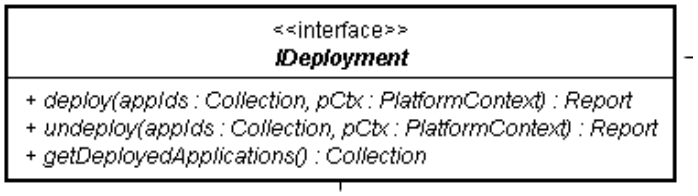


Diagrama 7.3-3 – Interfaces del subsistema *Deployment*

La interfaz ***IDeployment*** representa el punto de acceso a los distintos servicios vinculados con el deployment de aplicaciones. Posee operaciones para realizar deployments y undeployments, y para consultar cuáles aplicaciones ya han sido deployadas. Un resumen de las operaciones aparece a continuación.

Operación	**Deploy**
Parámetros	appIds – Colección de identificadores de aplicaciones a instalar (Strings). pCtx – Información contextual utilizada por el componente de abstracción de la plataforma (ver 1.1.1.1).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Realiza el deployment de un conjunto de aplicaciones. Determina un plan de deployment que permita instalarlas en el orden correcto en función de las dependencias. Posteriormente inicia el proceso de deployment según dicho plan, ejecutando todas las acciones de deployment indicadas en la especificación (DS). En caso de producirse un error al ejecutar una acción, se realiza el rollback de todas aquellas acciones, de la aplicación en cuestión, que ya hayan sido ejecutadas previamente y que sean revertibles. No se realiza el rollback de las aplicaciones ya instaladas según el plan pero se detiene el deployment de las restantes en dicho plan. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• Los paquetes conteniendo las aplicaciones indicadas deben encontrarse en el repositorio del nodo objetivo donde se realiza el deployment. • En conjunto, todas las aplicaciones deben satisfacer las dependencias indicadas en cada uno de sus descriptores (AD).
Poscondiciones	• Todas las aplicaciones indicadas son deployadas en el nodo objetivo. • Se persiste AD y DS, para ser consultada la información, para monitorear las aplicaciones y para posibilitar su posterior undeployment.

Operación	**undeploy**

Parámetros	`appIds` – Colección de identificadores de aplicaciones a desinstalar (Strings). `pCtx` – Información contextual utilizada por el componente de abstracción de la plataforma (ver 1.1.1.1).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Realiza el undeployment de un conjunto de aplicaciones. Determina un plan de undeployment que permita desinstalarlas en el orden correcto en función de las dependencias. Posteriormente inicia el proceso de undeployment según dicho plan, ejecutando todas las acciones de undeployment indicadas en la especificación (DS). No se toman medidas en caso de producirse un error al ejecutar una acción de undeployment para una cierta aplicación, pero se detiene el undeployment de las restantes en el plan de undeployment. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• Las aplicaciones indicadas deben estar deployadas en el nodo objetivo donde se ejecuta ésta operación. • Al desinstalarse en conjunto, todas las aplicaciones deben dejar el sistema en un estado donde todas las aplicaciones restante aún puedan continuar funcionando, es decir, que continúen satisfaciendo sus dependencias indicadas en cada uno de sus descriptores (AD).
Poscondiciones	• Todas las aplicaciones indicadas son desinstaladas del nodo objetivo. • Se eliminan los registros dentro de la persistencia de deployment.

Operación	**`getDeployedApplications`**
Parámetros	No tiene.
Retorno	Colección de descriptores (AD) de cada una de las aplicaciones que se encuentran deployadas y funcionando.
Descripción	Consulta al subsistema acerca de cuáles son las aplicaciones deployadas y funcionando.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Como se puede apreciar, y siguiendo un patrón común en el diseño del sistema, las operaciones de deployment y undeployment no lanzan excepciones en caso de producirse un error, sino que se registran (en caso de producirse) en el reporte retornado. Todo hecho acontecido al ejecutar estas operaciones es incluido, en forma de nota, advertencia y error, en el reporte que se retorna y mediante el cuál es posible consultar cada uno de los mismos una vez terminada la ejecución de la operación.

6.3.2.2 Estructura y comportamiento

Para comprender los elementos que componen el subsistema y como se comportan, es ilustrativo resumir brevemente los pasos que se deben realizar al momento de deployment y undeployment; los cuales conforman las principales funcionalidades del subsistema.

Al momento de deployment, el subsistema realiza básicamente las siguientes tareas:

3. Controlar dependencias basándose en cada AD.
4. Planificar el deployment.
5. Ejecutar el deployment de cada aplicación en el plan. Para cada aplicación:
    a. Validar la DS: determinar que todo parámetro es correcto tanto para las acciones de deployment como para las de undeployment.
    b. Ejecutar las acciones de deployment especificadas en la DS.
6. Persistir la información.

Se procede de manera similar al momento de undeployment pero sin efectuar validaciones puesto que fueron realizadas al instalar las aplicaciones, y ejecutando (obviamente) las acciones de undeployment.

En función de los pasos descriptos, se pueden identificar necesidades de comunicación con otros subsistemas:

- Para el acceso a los archivos dentro de los paquetes donde residen cada una de las aplicaciones, se requiere una comunicación con ***Deployable management*** para obtenerlo (a través de un ***IPackageAccessSolver***).
- Para el control de dependencias y la creación de un plan de deployment/undeployment se requiere obtener el acceso a un "chequeador" (***IDependenciesChecker***) por intermedio del subsistema ***Dependencies***.
- Para persistir los datos vinculados con el deployment, se requiere una comunicación con el subsistema ***Deployment persistence***.

Las funcionalidades del subsistema brindan la posibilidad de realizar múltiples deployments y undeployments, es decir, considerando varias aplicaciones a la vez. En la interna, el objeto ***IDeployment*** realiza tareas de inicialización de contexto y de control de dependencias, delegando el deployment y el undeployment de cada aplicación a un objeto de tipo ***IDeployer***. Éste requiere un objeto ***DeployerContext*** para obtener información común de deployment. Los mencionados conceptos se ilustran en el siguiente diagrama:

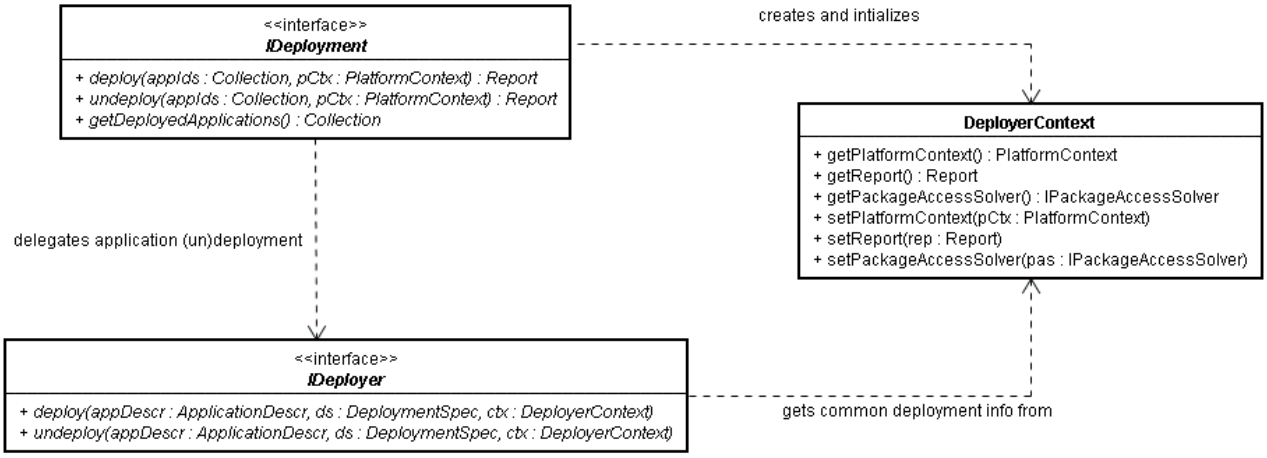


Diagrama 7.3-4 – *IDeployment*, *IDeployer*, DeployerContext

Por otro lado, en relación a la instanciación y ejecución de acciones de deployment/undeployment, es posible identificar algunos conceptos:

- Una **acción (IAction)** consiste en una tarea específica a realizar. Un ejemplo puede ser instalar un archivo JAR en el servidor de aplicaciones objetivo o ejecutar un script SQL sobre una base de datos.
- Se debe contar con un **manejador de acciones (IActionManager)**, el cual conozca todos los tipos de acciones existentes y que sepa instanciar adecuadamente las mismas en función de una especificación (incluida en la DS).
- Cada acción debe saber validar sus parámetros.
- Las acciones deben recibir un **contexto (ActionContext)** de ejecución que les permita obtener información útil para realizar la tarea que encapsulan.
- Tanto al validar como al ejecutar, las acciones deben reportar todo lo acontecido (notas, advertencias y errores) en el reporte que será retornado en las operaciones de deployment y de undeployment.
- Se debe contar con un mecanismo que permita revertir automáticamente acciones ya ejecutadas (en caso de error).

Las clases e interfaces asociadas con los conceptos mencionados se presentan en el siguiente diagrama:

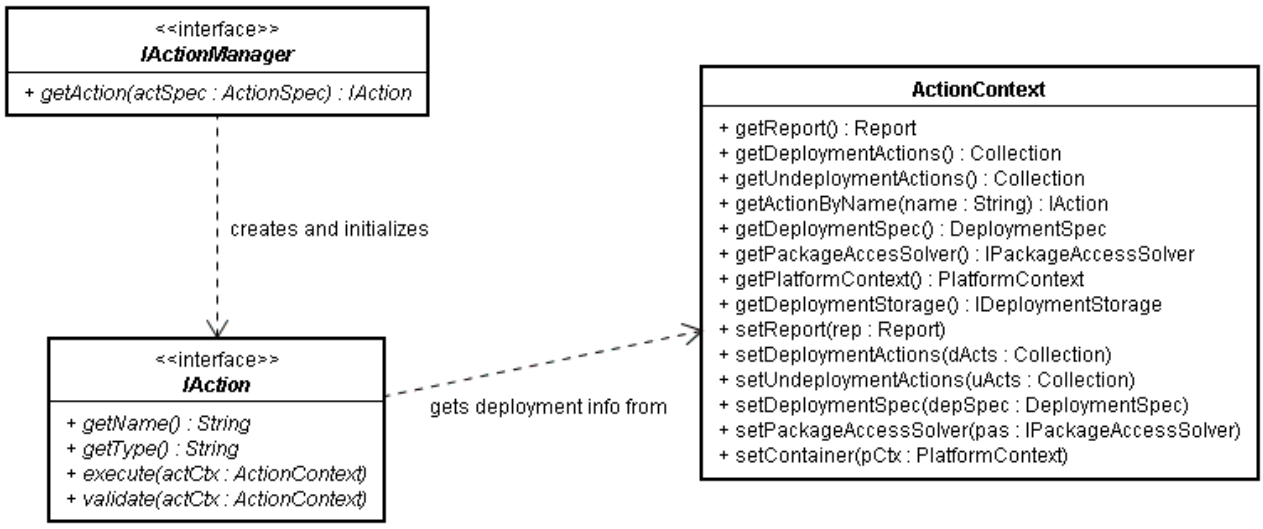


Diagrama 7.3-5 – IAction, IActionManager, ActionContext

Toda acción realiza la interfaz ***IAction*** y ser registrada dentro del manejador de acciones, el cual realiza la interfaz ***IActionManager***.

Resta presentar la interfaz de las acciones revertibles. Toda acción revertible deberá realizar, además de la interfaz ***IAction***, la interfaz ***IRevertible*** que aparece a continuación.

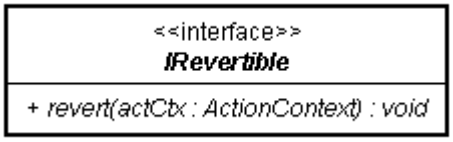


Diagrama 7.3-6 – IRevertible

En función de las clases e interfaces mencionadas, a continuación se presenta, por medio de una breve descripción y diagramas de secuencia, cómo se resuelve dentro del subsistema la operación ***deploy*** y ***undeploy***.

- **DEPLOY**

A continuación se incluyen diagramas de secuencia que pretenden ilustrar en gran medida el proceso de deployment. En el primero se muestra la secuencia iniciada en ***IDeployment*** y en el siguiente se muestra cómo esta continúa por delegación en un objeto ***IDeployer***.

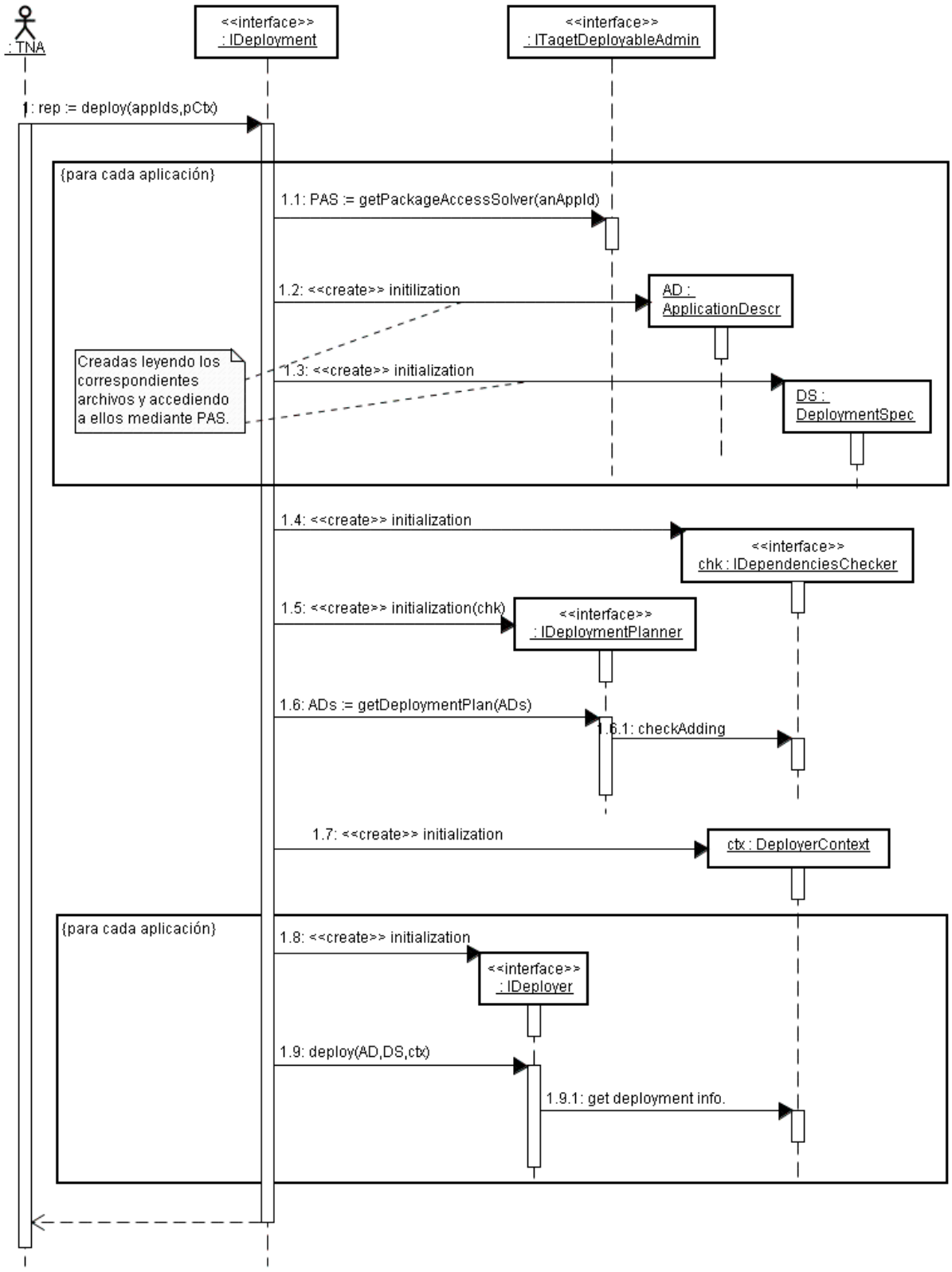


Diagrama 7.3-7 – Secuencia *IDeployment.deploy()*

El proceso comienza cuando se solicita al subsistema que realice el deployment de un conjunto de aplicaciones según sus respectivos identificadores. Éste dato es el único que se necesita debido a que como

precondición se requiere que se encuentren las mismas empaquetadas y almacenadas dentro del repositorio local de paquetes.

Toda información adicional es accedida directamente desde el paquete correspondiente por medio de un objeto de tipo ***IPackageAccessSolver*** el cual se obtiene desde el subsistema ***Deployable management*** a través de la interfaz ***ITargetDeployableAdmin*** tan solo con indicarle el identificador de la aplicación en cuestión. Como se indico en 7.3.2, el mencionado subsistema es responsable de la administración de paquetes y el acceso a su contenido.

Una vez obtenido el acceso a los paquetes, se obtienen cada uno de los AD's y DS's. Con los primeros se procede realizar un chequeo de dependencias que determine si es posible instalar todas las aplicaciones indicadas; para ello se utiliza un objeto ***IDependenciesChecker***. Si las dependencias no se puede satisfacer, se reporta lo sucedido y se cancela el proceso de deployment.

Si las dependencias son satisfechas, se procede a generar un plan de deployment por intermedio de un objeto ***IDeploymentPlanner***. Teniendo el plan de deployment se procede a realizar el deployment de cada aplicación, delegándole la responsabilidad a un objeto de tipo ***IDeployer*** y pasándole un contexto de deployment (***DeploymentContext***) inicializado.

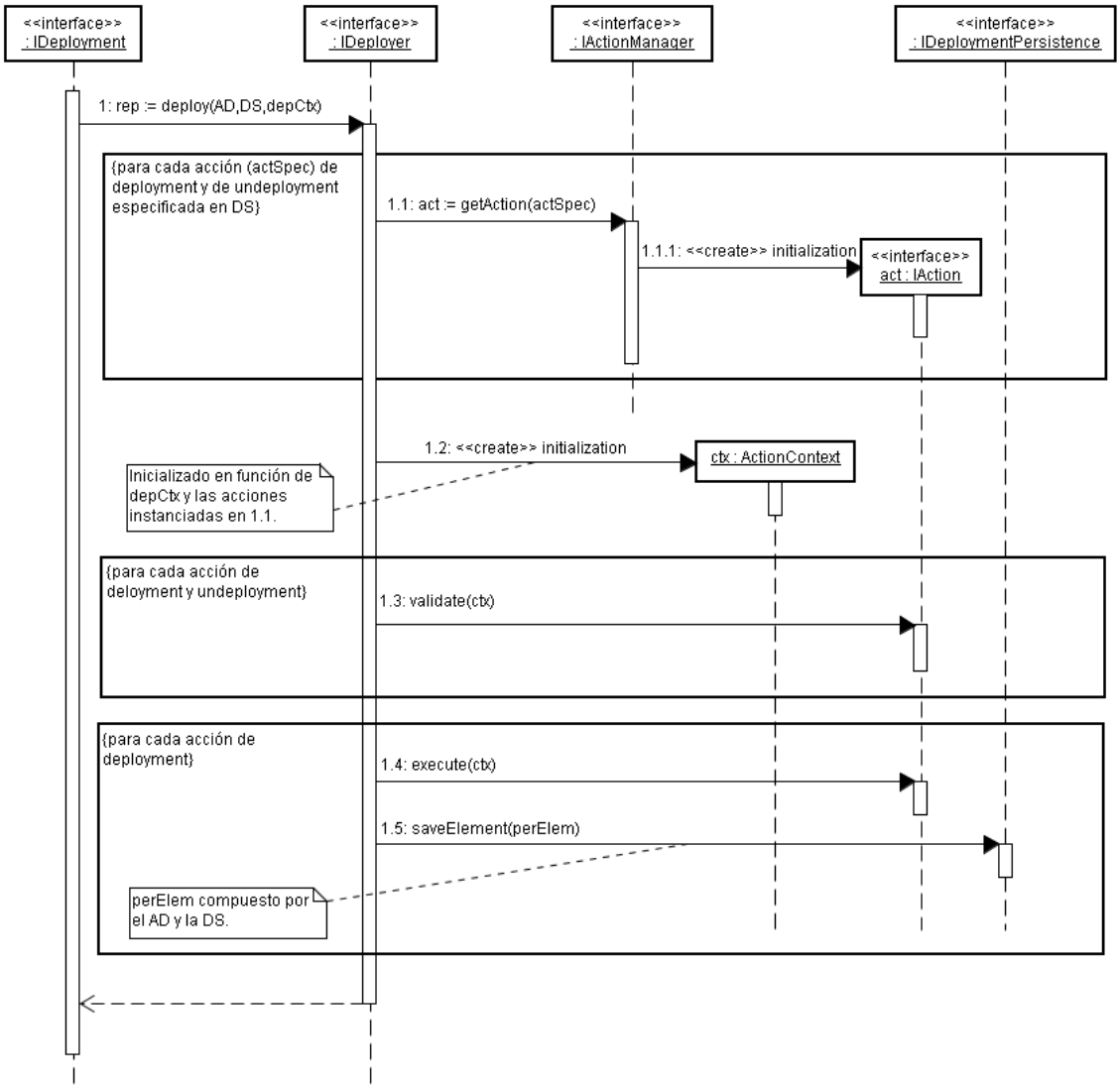


Diagrama 7.3-8 – Secuencia *IDeployer.deploy()*

Inicialmente se procede a instanciar todas las acciones (***IAction***) de deployment y undeployment para poder indicar la validación de sus parámetros. Luego se procede a inicializar, en función del ***DeployerContext***, al contexto de las acciones (***ActionContext***) que permite obtener toda información

necesaria para poder brindar a las acciones la flexibilidad suficiente para poder realizar una gran variedad de tareas; esto incluye: el reporte, el descriptor de aplicaciones (AD) la especificación de deployment (DS), las acciones instanciadas, entre otros elementos.

El siguiente paso es realizar la validación de la totalidad de las acciones mediante el uso de la operación ***validate*** y ejecutar aquellas de deployment por medio de la operación ***execute***. Las mismas lo harán en el orden indicado en la DS.

Finalmente, se persiste el AD y la DS para poder realizar consultas, tareas de mantenimiento y para realizar un posterior undeployment. Ésta tarea se realiza por medio del subsistema ***IDeploymentPersistence***.

Si durante este proceso alguna acción resulta fallida, no se producen excepciones sino que se reporta cualquier error en el reporte contenido en el contexto. Luego se procede a revertir toda acción ya ejecutada que sea revertible, es decir, que implemente la interfaz ***IRevertible***. Los anteriores deployments exitosos no son revertidos pero sí se detiene el proceso en el punto de error, por lo tanto, las restantes aplicaciones en el plan no son deployadas si se presenta este escenario.

- **UNDEPLOY**

Si bien el caso de uso “Undeployment de aplicaciones” no es considerado relevante para la arquitectura, es conveniente hacer una pequeña mención de cómo es resuelto dicho caso.

El mecanismo se basa en los mismos elementos intervinientes en el deployment, por lo que sería lógico pensar que la secuencia de undeployment es análoga a la presentada anteriormente con la salvedad de que las acciones que se ejecutan son aquellas de undeployment. Esta afirmación no es del todo correcta ya que ciertos pasos son sutilmente distintos o simplemente no realizados al momento de undeployment.

El plan de undeployment debe asegurar que las aplicaciones puedan ser desinstaladas sin producirse errores intermedios ocasionados por el incumplimiento (temporal) de dependencias. Éste criterio difiere del de deployment en donde se debe asegurar lo mismo en sentido inverso, es decir, que las aplicaciones puedan ser instaladas sin producirse errores por el no cumplimiento de dependencias.

Con respecto a la validación de acciones, no se realiza nuevamente debido a que se realizaron al momento de deployment. Por último, la información persistente es eliminada.

6.3.2.3 Componente de abstracción de la plataforma

Este componente fue incluido en el diagrama 7.3-5 como parte de la información contextual asociada a una acción, sin embargo, aún no se ha incluido una descripción detallada de su cometido. El componente de abstracción de la plataforma, o simplemente Plataforma, se concibe como el punto de interacción entre las acciones y la plataforma objetivo en donde se realizan deployments. La Plataforma, puede ser obtenida a través de su correspondiente información contextual, la cual es accesible desde las acciones. De esta forma, si la Plataforma brinda una operación denominada ***unaCiertaOperacion***, la cual puede ser de utilidad para implementar una acción específica, ésta puede ser accedida desde la información contextual asociada, o ***PlatformContext***.

El diagrama presentado a continuación, presenta la interfaz ***IPlataform*** correspondiente a la abstracción de la plataforma, y la información contextual ***PlatformContext***.

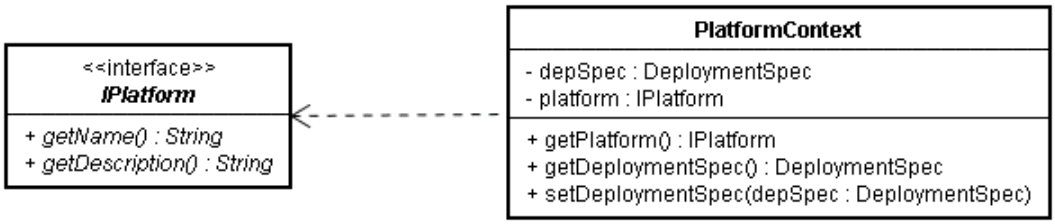


Diagrama 7.3-9 – Interfaz *IPlatform* y *PlatformContext*

Para implementar una plataforma propia, se debe configurar el sistema adecuadamente indicando cuál es la clase que implementa la interfaz ***IPlaform***.

Con este mecanismo, cualquier servicio de la plataforma objetivo puede ser expuesto para ser utilizado por una acción que lo requiera, ampliando las capacidades realizables desde las mismas.

6.3.3 Dependencies

Este subsistema es responsable de la realización del chequeo de dependencias. Para ello, utiliza el archivo ***Application descriptor (AD)*** procesado y un colección de AD, estos representan a las aplicaciones existentes. En esta sección se detallará cómo se procesan estos elementos y cuál es la forma en que interactúa el subsistema con otros para resolver las diversas funcionalidades.

6.3.3.1 Interfaces

Las interfaces identificadas son presentadas a continuación:

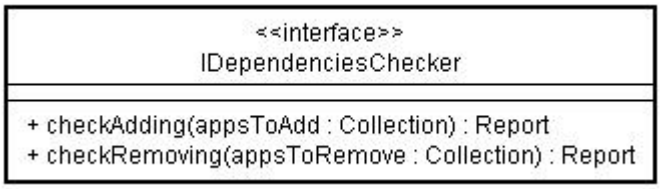


Diagrama 7.3-10 – Interfaces del subsistema *Dependencies*

La interfaz ***IDependenciesChecker*** representa el punto de acceso a los distintos servicios vinculados con el control de dependencias. Posee operaciones para realizar el control de las mismas. Un resumen de las operaciones aparece a continuación:

Operación	**`CheckAdding`**
Parámetros	AppToAdd – Colección de identificadores de aplicaciones a instalar.
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Realiza el control de dependencias de un conjunto de aplicaciones a instalar. En base a los AD de las aplicaciones cuyos identificadores son pasados por parámetro y a las aplicaciones existentes, se chequea cada una de las dependencias para cada una de las aplicaciones. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Operación	**`CheckRemoving`**
Parámetros	AppToRemove – Colección de identificadores de aplicaciones a desinstalar.
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Realiza el control de dependencias de un conjunto de aplicaciones a desinstalar. En base a los AD de las aplicaciones cuyos identificadores son pasados por parámetro y a las aplicaciones existentes, se chequea cada una de las dependencias para cada una de las aplicaciones. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

6.3.3.2 Estructura y comportamiento

Para comprender los elementos que componen el subsistema y como se comportan, es necesario resumir brevemente los pasos que se deben realizar al momento chequear; los cuales conforman las principales funcionalidades del subsistema.

Al momento del chequeo, el subsistema realiza básicamente las siguientes tareas:

1. Se le provee al chequeador de las aplicaciones deployadas y de las aplicaciones a deployar (con aplicaciones nos referimos a los AD).
2. El chequeador retorna un reporte del chequeo.

En relación a la instanciación y ejecución del chequeador, es posible identificar algunos conceptos:

- Una dependencia (dependency) consiste en un objeto de dependencia. Por ejemplo un identificador de aplicación y una versión.
- Se debe contar con un **manejador de dependencias (dependency manager)**, el cual conozca todos los tipos de dependencias existentes y que sepa instanciar adecuadamente las mismas en función de una especificación (incluida en el AD).
- Cada dependencia debe saber chequearse.
- Las dependencias deben recibir un **contexto (context)** de chequeo que les permita obtener información útil para realizar el control de la dependencia que encapsula.
- Al chequear, las dependencias deben reportar todo lo acontecido (notas, advertencias y errores) en un ítem de reporte que será capturado por el chequer y luego retornado.

Las clases e interfaces asociadas con los conceptos mencionados se presentan en el siguiente diagrama:

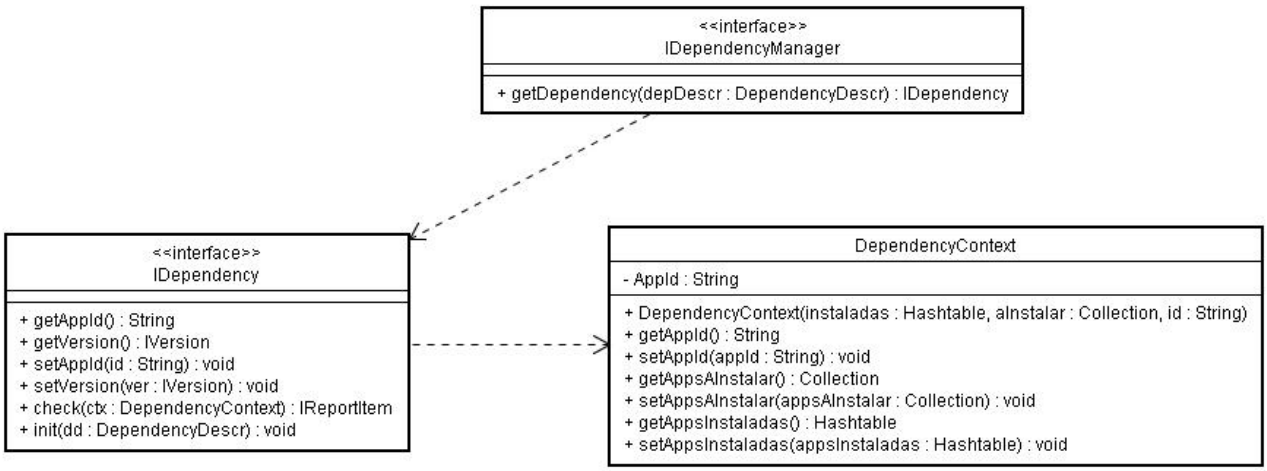


Diagrama 7.3-11 – IDependency, IDependencyManager, DependencyContext

Toda nueva dependencia debe realizar la interfaz ***IDependency*** y ser registrada dentro del manejador de dependencias, el cual deberá realizar la interfaz ***IDependencyManager***.

6.3.4 Resource management

Este subsistema es responsable de la administración de los recursos provistos y requeridos declarados en el momento del deployment. Este subsistema es el que provee operaciones para que las aplicaciones se puedan enlazar. El subsistema utiliza el archivo ***Resources specification (RS)***, en el cual se declaran lo recursos que la aplicación provee y los recursos que requiere. En esta sección se detallará cómo se procesan estos elementos y cuál es la forma en que interactúa el subsistema con otros para resolver las diversas funcionalidades.

6.3.4.1 Interfaces

Las interfaces identificadas son presentadas a continuación:

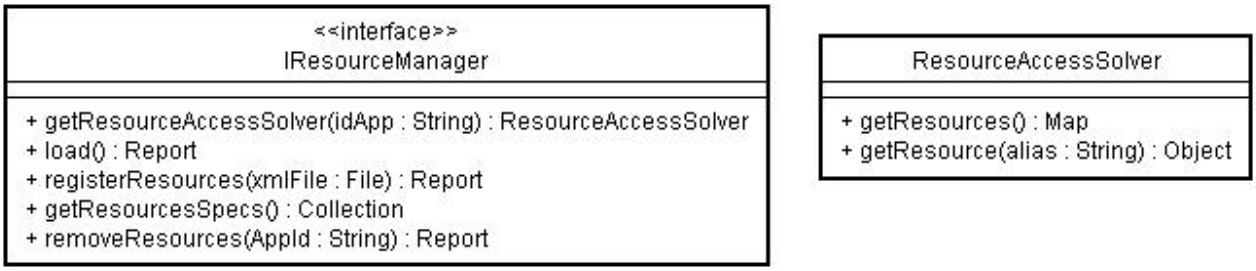


Diagrama 7.3-12 – Interfaces del subsistema ***Resource management***

La interfaz ***IResourceManager*** representa el punto de acceso a los distintos servicios vinculados con el manejo de recursos. Posee operaciones para realizar registrar y desregistrar recursos, y para obtener un acceso a los recursos. Un resumen de las operaciones aparece a continuación.

Operación	**registerResources**
Parámetros	file – Archivo RS conteniendo recursos provistos y requeridos (File)
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Registra los recursos provistos y reueridos que son declarados en el archivo bajo la aplicación cuyo identificador viene dado en el archivo.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	• Los recursos contenidos en el archivo, tanto provistos como requeridos quedan registrado en el nodo.

Operación	**RemoveResources**
Parámetros	idApp – identificador de la aplicación a la cual se le desea desregistrar los recursos provistos y requeridos. (String)
Retorno	Reporte conteniendo el resultado de la operación.
Descripción	Desregistra los recursos provistos y requeridos por la aplicación.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	• La aplicación no tendrá más recursos provistos ni requeridos.

Operación	**GetResourceAccessSolver**
Parámetros	idApp – identificador de la aplicación para la cual deseo tener acceso a los recursos requeridos.
Retorno	Un objeto ResourceAccessSolver que encapsula los recursos requeridos por la aplicación para que puedan ser accedidos.
Descripción	Encapsula en el objeto retornado los recursos requeridos por la aplicación para que puedan ser utilizados por esta en tiempo de ejecución.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	• El objeto retornado encapsula los recursos requeridos por la aplicación.

Operación	GetResourcesSpecs
Parámetros	No tiene.
Retorno	Colección de RS de todos los recursos provistos y requeridos registrados por cada aplicación
Descripción	Consulta cuales son las aplicaciones que tienen recursos registrados y retorna los respectivos RS
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Operación	Load
Parámetros	No tiene
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Registra todos los recursos especificados en la persistencia. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	Los recursos especificados en la persistencia son registrados en el sistema.

La clase ***ResourceAccessSolver*** representa el punto de acceso por parte de las aplicaciones (en tiempo de ejecución) a los recursos requeridos por las mismas, registrado en tiempo de deployment. Esta clase permite la realización de los "enganches" entre las aplicaciones. Un resumen de las operaciones aparece a continuación:

Operación	GetResources
Parámetros	No tiene.
Retorno	Un Map con las aplicaciones requeridas por la aplicación. La key es el "alias" que se le dio al recurso requerido en el archivo RS o el nombre en caso de no haberse declarado ningún "alias". El elemento es un Objeto (si se declaro una vista en el RS) o un Map con los elementos necesarios para realizar el enganche. Por ejemplo, si el recurso requerido es un session bean cuyo nombre es "SBean", y se decaro una vista "EJBHome", el elemento en el Map para la key "SBean" será un objeto que extienda EJBHome; sino, si no se declara una vista, el Map retornado contendra por ejemplo la jndi key del session bean y todos los elementos necesarios para poder instanciarlo.
Descripción	Retorna un map con los recursos requeridos por la aplicación, listos para ser utilizados.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	• El Map retornado provee los "enganches" a los recursos requeridos por una aplicación.

Operación	GetResource
Parámetros	alias – "alias" del recurso requerido para engancharse.
Retorno	Un objeto especifico (si se declaro una vista para él en el RS) o un map con los elementos necesarios para realizar el enganche.
Descripción	Busca el recurso solicitado, lo instancia si se declaro una vista o retorna un map con lo elementos para hacer el enganche.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	• El objeto retornado es el que se requiere por la aplicación solicitante para realizar el enganche.

6.3.4.2 Estructura y comportamiento

Uno de los principales elementos del subsistema es la acción ManageResourcesAction. La misma es la encargada de registrar en el subsistema los recursos provistos y requeridos por una aplicación dada. Los recursos registrados son los que vienen dados en el archivo ***Resources specification (RS)***. Esta acción es ejecutada por el deployer en el momento del deployment.

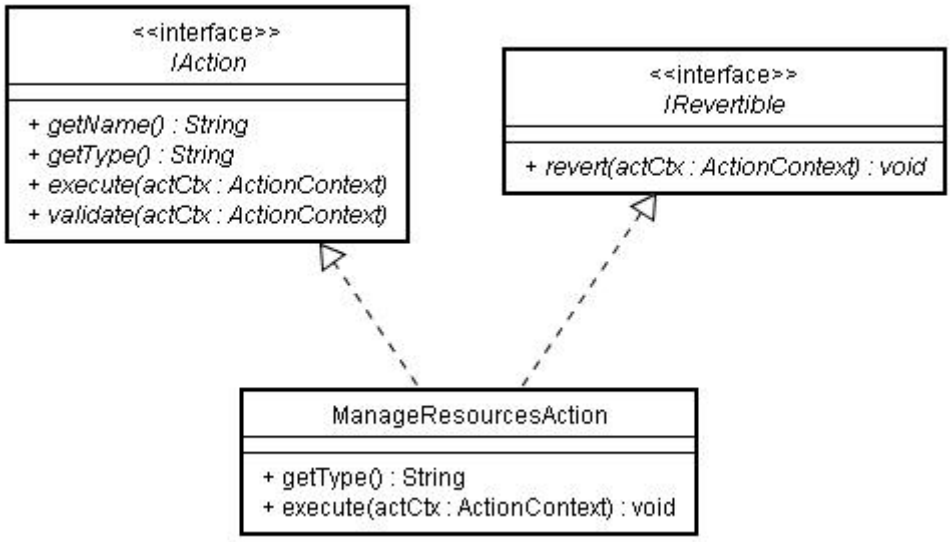


Diagrama 7.3-13 – Acción ManageResourcesAction

El ResourceAccessSolver es el encargado de proveer los mecanismos de enlaces por eso resulta interesante ver los pasos a realizar al momento de utilizar el ResouceAccessSolver; el cual provee la principal funcionalidad del subisistema, los enlaces entre aplicaciones.

1. En tiempo de ejecución, la aplicación que se esta ejecutando, le solicita al subsistema una forma de enlazarse con los recursos que requiere y que fueron declarados al momento del deployment.
2. El subsistema le da un ResourceAccessSolver para que pueda enlazarse.
3. Al tener un ResourceAccessSolver, le solicita al mismo los enlaces a los recursos que necesite.

En relación a la creación de “enganches”, es posible identificar algunos conceptos:

- Un recurso **(resource)** consiste en el punto de entrada a una/s funcionalidad/es provista/s por una aplicación. Un ejemplo puede ser un session bean que expone una aplicación.
- Se debe contar con un **manejador de recursos (manager)**, el cual conozca todos los tipos de recursos existentes y que sepa instanciar adecuadamente los mismos en función de una especificación (incluida en la RS).
- Cada recurso debe saber proveer un objeto de enlace a él.
- Los recursos deben recibir un **contexto (ProvidedResourceSpec, RequiredResourceSpec)** que les permita obtener información útil para realizar el objeto de enlace.
- Tanto al validar como al ejecutar, las acciones deben reportar todo lo acontecido (notas, advertencias y errores) en el reporte que será retornado en las operaciones de deployment y de undeployment.

Las clases e interfaces asociadas con los conceptos mencionados se presentan en el siguiente diagrama:

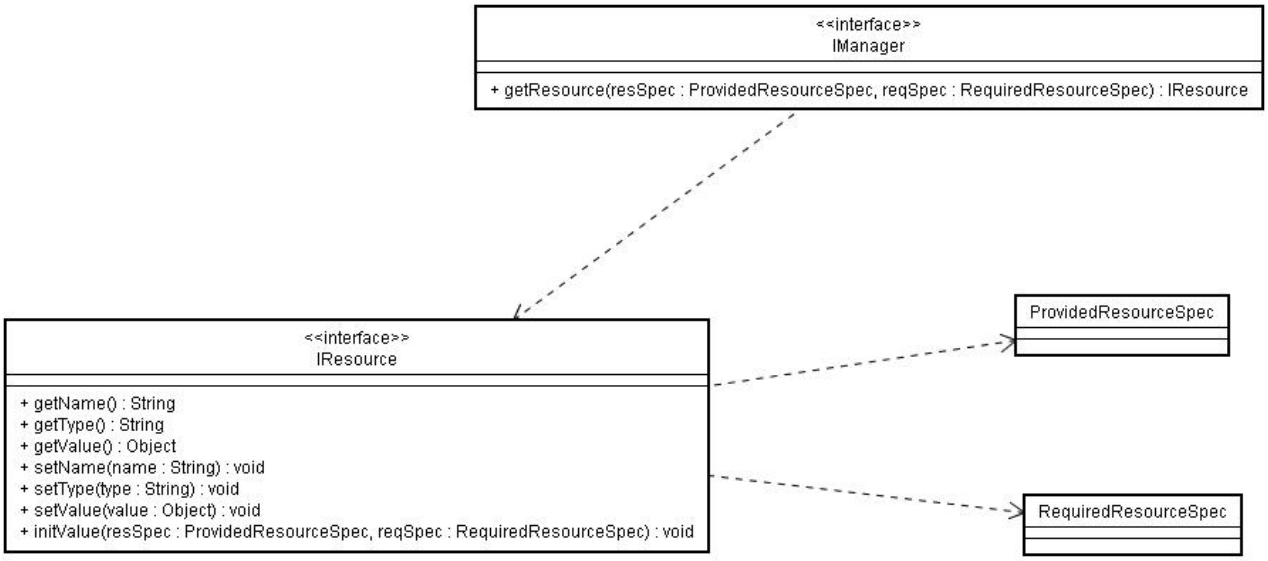


Diagrama 7.3-14 – IResource, IManager, ProvidedResourceSpec, RequiredResourceSpec

Toda nuevo recurso debe realizar la interfaz ***IResource*** y ser registrado dentro del manejador de recursos, el cual deberá realizar la interfaz ***IManager***.

6.3.5 Deployment persistence

Este subsistema es responsable de administrar la información persistente del servicio de deployment. La información persistente consiste en el ***Application descriptor (AD)*** y la ***Deployment specification (DS)*** de cada una de las aplicaciones instaladas. El modelo de los datos utilizado sigue el descrito en [6] y [7], sin embargo, es importante destacar el hecho de que, debido a la decisión de encapsular el mecanismo de persistencia de la información de deployment, la forma en que se almacenan los datos puede ser modificada sencillamente sin alterar el resto del sistema siempre y cuando se respete la interfaz expuesta por este subsistema.

En esta sección se describe el subsistema ***Deployment persistence*** incluyendo solamente aquellos elementos considerados relevantes para la arquitectura.

6.3.5.1 Interfaces

Las interfaces identificadas son presentadas a continuación:

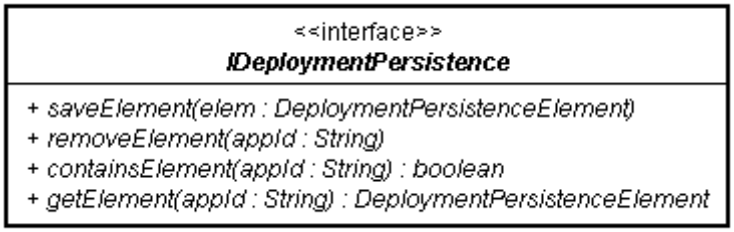


Diagrama 7.3-15 – Interfaces del subsistema *Deployment persistence*

La interfaz ***IDeploymentPersistence*** representa el punto de acceso a los distintos servicios vinculados con la persistencia de la información de deployment. El tipo de dato ***DeploymentPersistenceElement*** está formado por un AD y una DS correspondiente a una cierta aplicación.

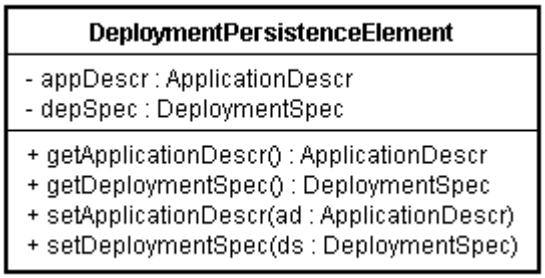


Diagrama 7.3-16 – Tipo de dato *DeploymentPersistenceElement*

Un resumen de las operaciones del subsistema aparece a continuación:

Operación	**`saveElement`**
Parámetros	`elem` – Elemento persistente de deployment.
Retorno	Reporte incluyendo notas, advertencias y errores.
Descripción	Persiste la información de deployment (AD y DS) correspondiente a una cierta aplicación.
Excepciones	`DeploymentPersistenceException`
Precondiciones	No tiene.
Poscondiciones	Se agrega la información contenida en el elemento de deployment persistente a la base de datos de deployment.

Operación	**`removeElement`**
Parámetros	`appId` – Identificador de la aplicación.
Retorno	No tiene.
Descripción	En función del identificador de la aplicación pasado como parámetro, elimina de la base de datos de deployment toda información referente a la misma.
Excepciones	`DeploymentPersistenceException`
Precondiciones	• Existe información persiste asociada al identificador de la aplicación.
Poscondiciones	• Se elimina de la base de datos de deployment toda información asociada al identificador de aplicación pasado como parámetro.

Operación	**`getElement`**
Parámetros	`appId` – Identificador de la aplicación.
Retorno	Elemento de persistencia de deployment correspondiente al identificador indicado.
Descripción	Consulta la base de datos en busca de la información asociada a la aplicación identificada con el identificador pasado como parámetro y retorna un elemento de persistencia de deployment conteniendo la información solicitada. Si `contains(appId)` es falso, retorna NULL.
Excepciones	`DeploymentPersistenceException`
Precondiciones	N/A
Poscondiciones	N/A

Operación	**`containsElement`**
Parámetros	`appId` – Identificador de la aplicación.
Retorno	Valor booleano indicando si existe algún elemento de persistencia de deployment asociado a la aplicación de identificador igual al pasado como parámetro.
Descripción	Consulta a la base de datos de deployment verificando la existencia de algún elemento de persistencia de deployment asociado a la aplicación de identificador igual al pasado como parámetro.
Excepciones	`DeploymentPersistenceException`
Precondiciones	N/A
Poscondiciones	N/A

6.3.6 Resources persistence

Este subsistema es el encargado de manejar la persistencia del resource manager. La información persistida son los ***Resources specification (RS)*** de cada una de las aplicaciones instaladas. Al encapsular la persistencia de la información del resource manager, la forma en que se persisten los datos puede ser modificada sin demasiado esfuerzo siempre y cuando se respete la interfaz expuesta el subsistema.

En esta sección se describe el subsistema ***Resources persistence*** incluyendo solamente aquellos elementos considerados relevantes para la arquitectura.

6.3.6.1 Interfaces

Las interfaces identificadas son presentadas a continuación:

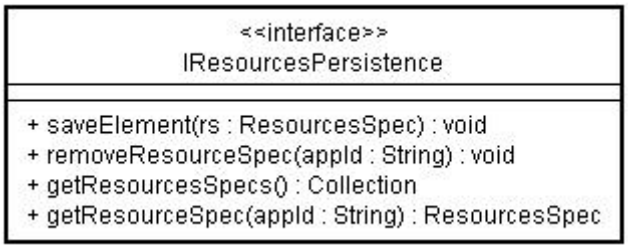


Diagrama 7.3-17 – Interfaces del subsistema *Resources persistence*

La interfaz ***IResourcesPersistence*** es el punto de acceso a los distintos servicios vinculados con la persistencia de la información de los recursos.

Un resumen de las operaciones del subsistema aparece a continuación:

Operación	**SaveElement**
Parámetros	elem – recursos provistos y requeridos por una aplicación, RS. (ResourcesSpec)
Retorno	Reporte incluyendo notas, advertencias y errores.
Descripción	Persiste la información de los recursos (RS) correspondiente a una cierta aplicación.
Excepciones	ResourcesPersistenceException
Precondiciones	No tiene.
Poscondiciones	Se agrega la información contenida en el elemento RS la persistencia de recursos.

Operación	**RemoveResourceSpec**
Parámetros	appId – Identificador de la aplicación a la cual se le eliminaran los recursos de la persistencia.
Retorno	No tiene.
Descripción	En función del identificador de la aplicación pasado como parámetro, elimina de la persistencia de recursos toda información referente a la misma.
Excepciones	ResourcesPersistenceException
Precondiciones	• Existe información persiste asociada al identificador de la aplicación.
Poscondiciones	• Se elimina de la persistencia toda información asociada al identificador de aplicación pasado como parámetro.

Operación	`GetResourceSpec`
Parámetros	`appId` – Identificador de la aplicación.
Retorno	RS correspondiente al identificador indicado.
Descripción	Consulta la persistencia en busca de la información asociada a la aplicación identificada con el identificador pasado como parámetro y retorna RS conteniendo la información solicitada. Si `contains(appId)` es falso, retorna NULL.
Excepciones	ResourcesPersistenceException
Precondiciones	N/A
Poscondiciones	N/A

Operación	`GetResourcesSpecs`
Parámetros	`No tiene`
Retorno	Colección de los RS que se encuentran en la persistencia.
Descripción	Consulta la persistencia y retorna todos los recursos persistidos en una colección de RS.
Excepciones	ResourcesPersistenceException
Precondiciones	N/A
Poscondiciones	N/A

6.3.7 Deployables Repository

Este subsistema es el encargado del manejo de los paquetes al nivel mas bajo. La principal funcionalidad es la administración del repositorio. La interfaz encargada de esto es ***IDeployablesRepository*** a continuación se mostrarán las principales funcionalidades de la interfaz.

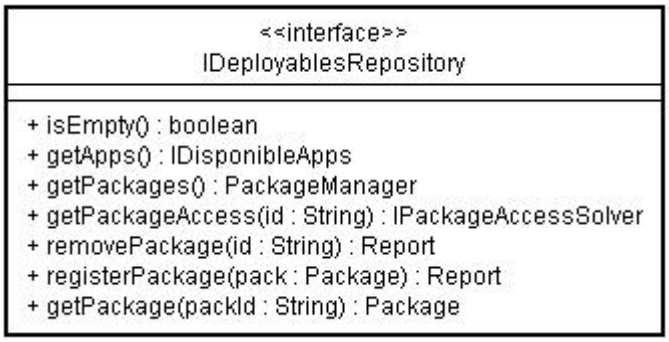


Diagrama 7.3-18 – Tipo de dato *IDeployablesRepository*

Operación	**`RegisterPackage`**
Parámetros	`name` – Nombre del paquete a registrar en el nodo (Strings).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Registra un paquete existente el repositorio del nodo. Busca el paquete cuyo nombre fue pasado como parámetro y lo registra. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• El paquete cuyo nombre es pasado como parámetro debe existir en el repositorio del nodo.
Poscondiciones	• El paquete queda registrado en el nodo como disponible para deplorar y si el nodo es mirror, también queda disponible para la descarga.

Operación	**`RemovePackage`**
Parámetros	`id` – Identificador del paquete a eliminar en el repositorio del nodo. (Strings).
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores.
Descripción	Desregistra y elimina el paquete del repositorio del nodo. Se notifica todo hecho acontecido en el reporte que se retorna como resultado.
Excepciones	No tiene.
Precondiciones	• El paquete debe existir en el repositorio del nodo y debe estar registrado en el mismo.
Poscondiciones	• El paquete es eliminado del nodo.

Operación	**`GetPackages`**
Parámetros	No tiene.
Retorno	Estructura conteniendo la información de los paquetes disponibles y registrados en el nodo.
Descripción	Consulta al subsistema acerca de cuáles son los paquetes disponibles en el nodo.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Operación	`GetPackage`
Parámetros	`id` – Identificador del paquete al cual se desea acceder. (String).
Retorno	Objeto que encapsula la información del paquete.
Descripción	Retorna un objeto que provee la información del paquete cuyo identificador es pasado como parámetro.
Excepciones	No tiene.
Precondiciones	• El paquete debe existir en el repositorio del nodo y debe estar registrado en el mismo.
Poscondiciones	• El objeto devuelto provee la información del paquete solicitado.

Operación	`GetPackageAccess`
Parámetros	`id` – Identificador del paquete al cual se desea acceder. (String).
Retorno	Objeto que encapsula el acceso a un paquete dado.
Descripción	Retorna un objeto que provee acceso al paquete cuyo identificador es pasado como parámetro.
Excepciones	No tiene.
Precondiciones	• El paquete debe existir en el repositorio del nodo y debe estar registrado en el mismo.
Poscondiciones	• El objeto devuelto provee el acceso al paquete solicitado.

Operación	`GetApps`
Parámetros	No tiene.
Retorno	Estructura conteniendo la información de las aplicaciones contenidas en los paquetes disponibles y registrados en el nodo.
Descripción	Consulta al subsistema acerca de cuáles son las aplicaciones contenidas en los paquetes disponibles en el nodo.
Excepciones	No tiene.
Precondiciones	N/A
Poscondiciones	N/A

Para las operaciones ***registryPack*** y ***removePack*** en caso de producirse errores, estos se registran en el reporte retornado. Todo hecho acontecido al ejecutar estas operaciones es incluido, en forma de nota, advertencia y error, en el reporte que se retorna y mediante el cuál es posible consultar cada uno de los mismos una vez terminada la ejecución de la operación.

Este subsistema también es el encargado de proveer el acceso al contenido de los paquetes físicos y manejar todo lo concerniente a filesystems. Al encapsular la forma en que se accede a los paquetes, la forma de acceso puede ser modificada sin demasiado esfuerzo siempre y cuando se respete la interfaz expuesta el subsistema.

La interfaz del subsistema que provee acceso a los paquetes deplorables físicos es ***IPackageAccessSolver*** la cual provee acceso al contenido de los paquetes. Quien use esta interfaz no necesitara conocer el lugar donde se encuentra el paquete, solo bastará con el nombre del mismo. Para obtener alguno de los archivos contenidos en el paquete bastará con pasar el nombre del archivo incluyendo su ruta relativa al paquete.

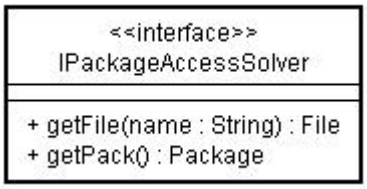


Diagrama 7.3-19 – Tipo de dato *IPackageAccessSolver*

Para poder cumplir con su objetivo, esta interfaz utiliza un objeto de tipo ***FileManager*** el cual es el encargado de manejar el filesystem.

Un resumen de las operaciones del subsistema aparece a continuación:

Operación	**GetFile**
Parámetros	Name – Nombre del archivo al que se quiere accede, incluyendo su path relativo al paquete.
Retorno	Archivo solicitado.
Descripción	Busca el archivo solicitado dentro del paquete, lo abre y lo devuelve.
Excepciones	
Precondiciones	El nombre de archivo pasado debe corresponderse con un archivo existente en el paquete.
Poscondiciones	Se retorna abierto el archivo solicitado.

Operación	**getPackage**
Parámetros	No tiene.
Retorno	Objeto de tipo Package con la información del paquete al cual se puede acceder en esta instancia de la clase.
Descripción	Devuelve la información relativa al paquete que se esta accediendo.
Excepciones	N/A
Precondiciones	N/A
Poscondiciones	• Retorna la información del paquete al que se esta accediendo.

7 Trazabilidad de Vista Lógica a la Vista de Implementación

En esta sección se muestra, mediante un diagrama, la trazabilidad entre los subsistemas más relevantes a la Arquitectura a nivel de diseño en la vista lógica, y aquellos correspondientes en la vista de implementación.

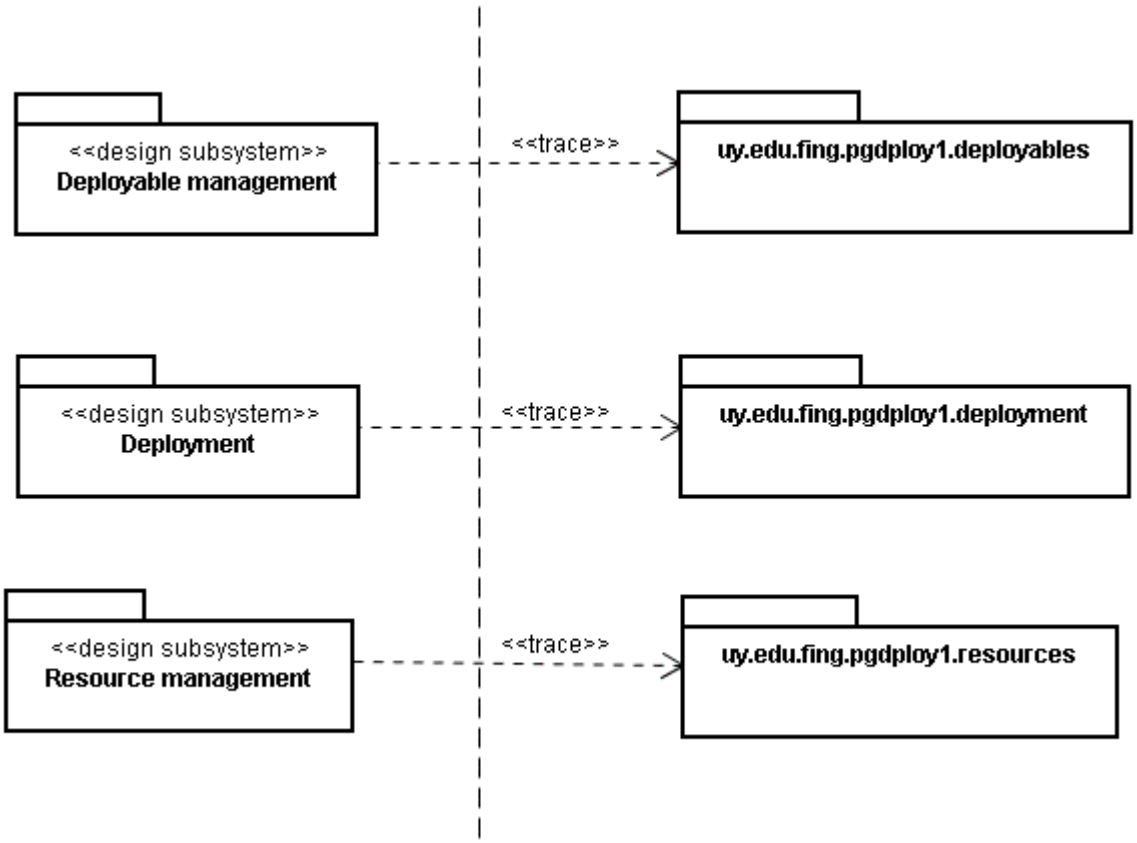


Diagrama 8-1 –Trazabilidad desde el modelo de casos de uso
al modelo de análisis

8 Vista de la Implementación

8.1 Subsistemas

8.1.1 Deployable management

Este subsistema se reside en el paquete ***uy.edu.fing.pgdploy1.deployables*** y contiene definición de las siguientes interfaces:

- ***ITargetDeployableAdmin***
- ***ISupportDeployableAdmin***
- ***IDeployablesRepository***
- ***INodeConnector***

Todas las interfaces son acompañadas de una respectiva implementación por defecto, excepto el caso de ***INodeConnector***, no se incluye implementación alguna, la cual corre por cuenta de quien haga uso del sistema.

8.1.2 Deployment

Este subsistema se reside en el paquete ***uy.edu.fing.pgdploy1.deployment*** y contiene definición de las siguientes interfaces:

- ***IDeployment***
- ***IDeployer***
- ***IDeploymentPlanner***
- ***IActionManager***
- ***IAction***
- ***IRevertible***
- ***IDeploymentPersistence***
- ***IPlatform***

Todas las interfaces son acompañadas de una respectiva implementación por defecto. Para el caso de ***IAction*** (y para ***IRevertible*** en caso de acciones revertibles), se incluye la implementación de acciones básicas.

8.1.3 Resource management

Este subsistema se reside en el paquete ***uy.edu.fing.pgdploy1.resources*** y contiene definición de las siguientes interfaces:

- ***IResourceManager***

Todas las interfaces son acompañadas de una respectiva implementación por defecto.

9 Vista de Deployment

9.1 Diagrama de Deployment

El siguiente diagrama muestra los nodos relevantes a la Arquitectura del Sistema y cómo son las conexiones entre ellos.

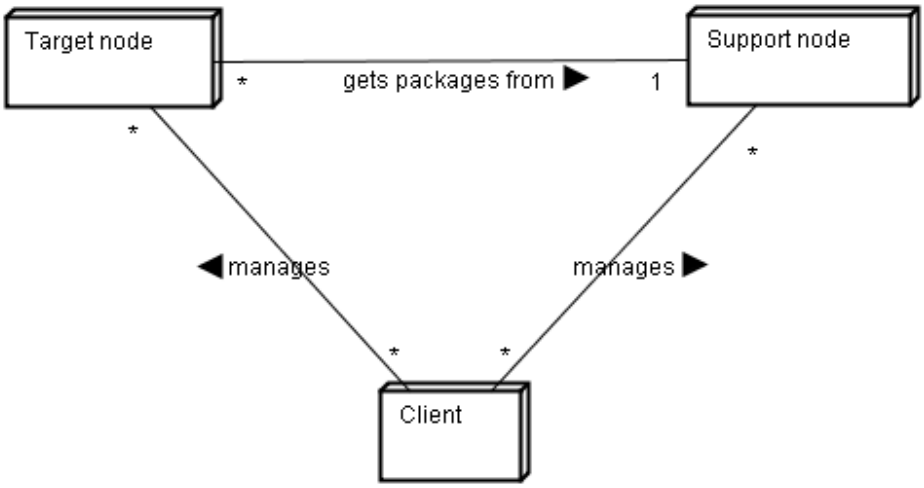


Diagrama 8.1-1 - Diagrama de distribución

9.2 Nodos

9.2.1 Nodo cliente (Client node)

El nodo cliente consiste simplemente en un PC de escritorio con capacidades que le permitan ejecutar un browser para ingresar a las aplicaciones web que ofician de punto de entrada a las funcionalidades provistas por los demás nodos del sistema.

9.2.2 Nodo soporte (Support node)

El nodo de soporte es donde reside el subsistema de ***Gestión de DP's*** el cual es responsable de administrar los DP's disponibles para ser bajados desde los nodos objetivo y posteriormente instalados en ellos. Consta de un servidor de aplicaciones J2EE que provee el ambiente de ejecución para el subsistema mencionado, un servidor web (o uno incorporado al servidor de aplicaciones) y un servidor FTP mediante el cual acceder y descargar de los DP's.

9.2.3 Nodo objetivo (Target node)

En este nodo se encuentra el servidor de aplicaciones J2EE objetivo para el deployment de DP's. Éstos son descargados desde el nodo soporte asociado y posteriormente son instalados. Contiene al subsistema de ***Descarga de paquetes***, es decir, el responsable de atender pedidos desde un cliente administrador del nodo y resolver la descarga de DP's desde el nodo soporte. En un nivel más bajo, reside el subsistema de ***Deployment*** el cual es responsable de realizar la instalación local de los DP's descargados por el subsistema de descarga de paquetes.

9.2.4 Nodo intermedio (Support/Target node)

Este nodo es un híbrido entre el nodo objetivo y el de soporte. Ejecuta sus propias aplicaciones (eventualmente obtenidas de un nodo soporte) y a su vez ofrece DP's para que otros nodos objetivo puedan descargar los mismos.

9.3 Conexiones

9.3.1 Nodo cliente al Nodo objetivo

El enlace entre el cliente y el nodo objetivo es mediante HTTP sobre TCP/IP. Esto es lógico ya que el acceso al nodo objetivo se realiza desde un browser. El ancho de banda no es de vital interés en esta conexión.

9.3.2 Nodo cliente al Nodo soporte

El enlace entre el cliente y el nodo soporte es mediante HTTP sobre TCP/IP. Del mismo modo que en el punto anterior, el acceso al nodo soporte se realiza desde un browser. Con respecto al ancho de banda, no se requiere que sea de alta capacidad ya que los DP's pueden ser agregados al nodo soporte de manera local, sin embargo, cuando por comodidad se requiere que la administración de dicho nodo y la subida de DP's se haga remotamente, es recomendable que así lo sea puesto que los paquetes pueden ser de tamaño importante.

9.3.3 Nodo objetivo al Nodo soporte

Esta conexión se realiza mediante HTTP o HTTPS sobre TCP/IP. El enlace es recomendable que sea de alta capacidad y/o velocidad debido a que posiblemente sean descargados DP's de gran tamaño desde el nodo soporte. Se diferencian dos tipos de conexión, una dedicada al intercambio de información entre componentes del sistema, en donde se utilizará Webservices y SOAP, y otra dedicada a la descarga de paquetes, la cual se realizará mediante la obtención directa de paquetes desde algún sitio web destinado a tales efectos y que reside en el nodo soporte.

9.4 Componentes

En la presente sección se ingresa en la interna de cada nodo detallando cuáles componentes forman parte de cada nodo desde el punto de vista del subsistema de deployment y que son relevantes para la arquitectura. Para cada caso, en que se considere necesario hacerlo, se incluye un diagrama y una breve descripción de cada componente en el nodo.

9.4.1 Nodo cliente

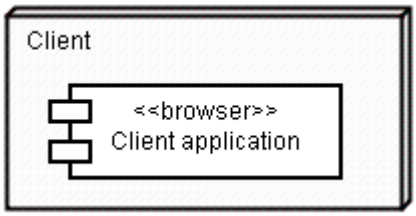


Diagrama 10.4-1 – Componentes del nodo cliente

En el nodo cliente no se exige nada más que un navegador o browser que permita acceder al sitio web del nodo objetivo o soporte, dependiendo de las características del usuario y qué desea hacer.

9.4.2 Nodo soporte

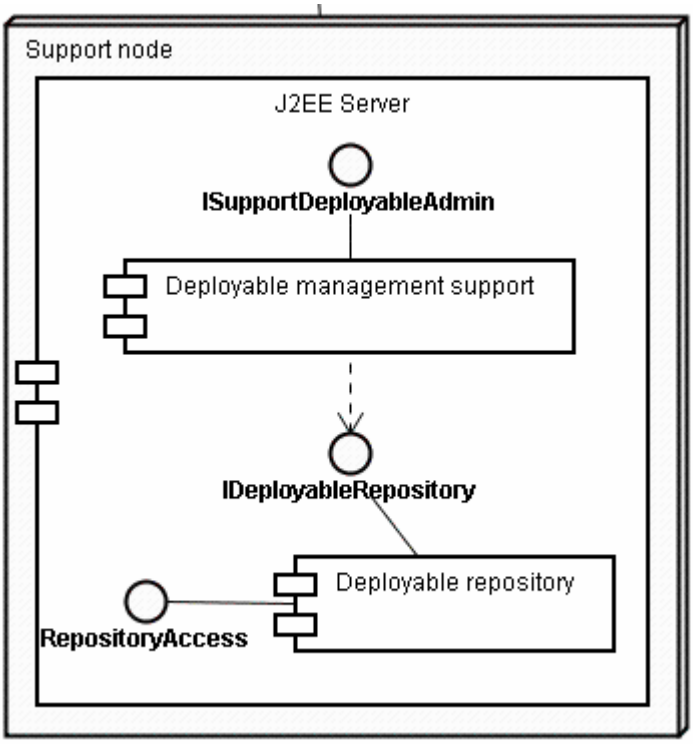


Diagrama 10.4-2 – Componentes del nodo soporte

El nodo soporte tiene como objetivo primordial exponer paquetes deployables de tal manera de que sea posible descargarlos desde nodos objetivo (o intermedio) para su posterior instalación. Desde el punto de vista de la arquitectura se destacan el componente ***Deployable management support*** que oficia de punto de acceso a la información del nodo desde los nodos objetivo y el componente ***Deployable repository*** el cual se encarga de administrar el repositorio de paquetes.

El primero de los componentes mencionados, ***Deployable management support***, es accedido por intermedio de la interfaz ***ISupportDeployableAdmin*** la cual es expuesta en principio vía un webservice SOAP o bien por otros mecanismos de RPC como RMI/IIOP.

El segundo, ***Deployable repository***, gestiona el repositorio local de paquetes y expone los mismos por una interfaz FTP (o SFTP) que denominamos ***RepositoryAccess***.

9.4.3 Nodo objetivo

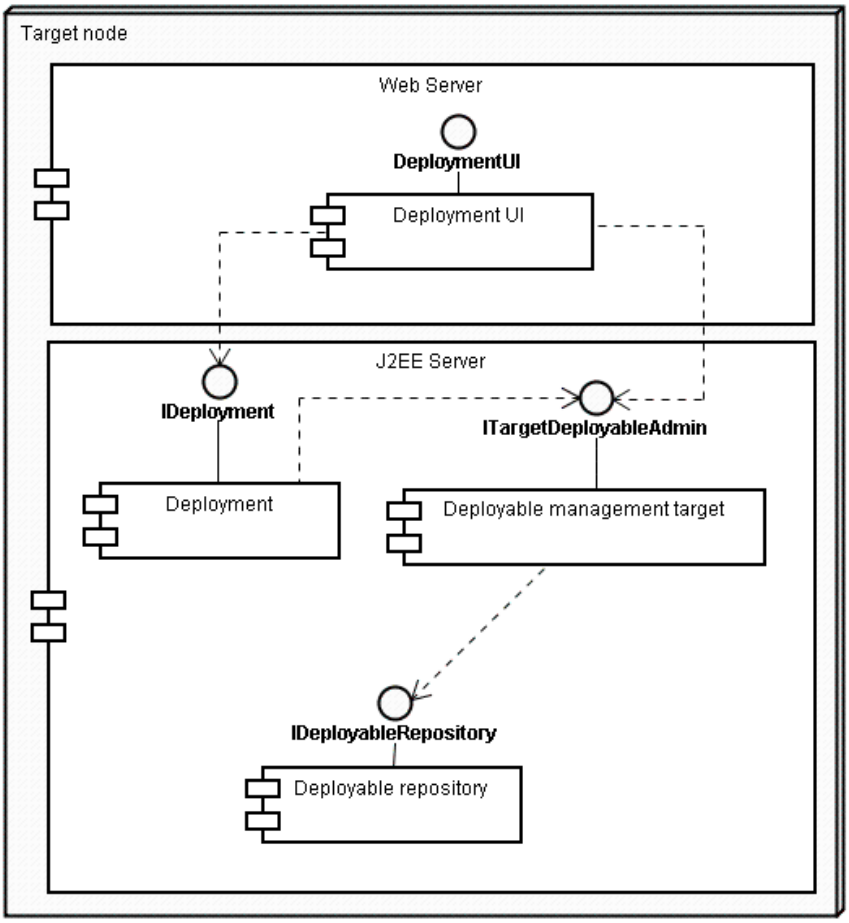


Diagrama 8.4-3 – Componentes del nodo objetivo

El nodo objetivo presenta mayor complejidad y desde el punto de vista de la arquitectura destacamos los siguientes componentes: ***DeploymentUI***, ***Deployment***, ***Deployable management target***, y ***Deployable repository***.

El primero consiste en la interfaz web del deployment incluyendo la administración de aplicaciones instaladas y la administración de paquetes en el repositorio local.

El segundo componente, ***Deployment***, es el responsable de llevar a cabo todas las tareas vinculadas con el deployment y undeployment de aplicaciones; para ello utiliza ***Deployable management target*** para obtener acceso a las aplicaciones contenidas en paquetes pertenecientes al repositorio local. Por otro lado, ***Deployable management target*** además es responsable de la comunicación con el nodo soporte para la descarga de paquetes desde dicho nodo, o para consultarlo acerca de cuáles son los paquetes que expone y cuáles son las aplicaciones contenidas en los mismos.

Finalmente, ***Deployable repository*** gestiona el repositorio local de paquetes el cual contiene aquellos descargados desde el nodo soporte (o intermedio) o bien aquellos incorporados directamente desde un CD o similar. Permite acceder de manera flexible a todos los archivos contenidos en cada uno de ellos.

9.4.4 Nodo intermedio

El nodo intermedio es un híbrido de nodo objetivo y nodo soporte, por lo tanto al nodo objetivo de le debe incluir el componente ***Deployable management support*** y realizar alguna configuración menor para que dicho nodo adquiera funcionalidades de nodo soporte.

UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

ANEXO D

PAQUETE DEPLOYABLE / ARCHIVOS DE CONFIGURACIÓN

Estudiantes
Nicolás Doroskevich
Sebastián Moreira

Tutores
Federico Piedrabuena
Raúl Ruggia

Proyecto de grado 2004
Facultad de Ingeniería - Universidad de la República

Índice general

1. Introducción

1.1 Propósito

El presente documento tiene como objetivo introducir al lector al contenido de los diversos archivos de configuración, obligatorios u opcionales, que pueden ser incluidos en un paquete deployable, es decir, en el empaquetamiento de una cierta versión de una aplicación instalable utilizando los servicios del framework de deployment.

1.2 Alcance

En este documento se detallará brevemente el contenido de los archivos de configuración de una aplicación, y que forman parte de un paquete deployable. Los archivos que se considerarán son:

- ***Descriptor de Aplicación (AD – Application Descriptor)***
  Corresponde al archivo XML ***descr.xml*** del paquete deployable. Este archivo es obligatorio puesto que representa la descripción básica de cada aplicación instalable.

- ***Especificación de Deployment (DS – Deployment specification)***
  Corresponde al archivo XML ***deploy.xml*** del paquete deployable. Este archivo es obligatorio puesto que indica las acciones de deployment y undeployment.

- ***Especificación de Recursos (RS – Resources specification)***
  Corresponde al archivo XML ***resources.xml*** del paquete deployable. Este archivo no es obligatorio ya que una aplicación no está obligada a utilizar los servicios del componente de gestión de recursos, o Manejador de Recursos.

Existe un concepto común a cada archivo listado, dicho concepto es el elemento ***parameter*** el cual permite la especificación de parámetros. Debido a que es compartido, se detallarán al final de este documento.

En este documento no se incluirá un DTD para cada XML, pero sí se incluirá su correspondiente schema XML. Esta aclaración se hace extensiva para el elemento ***parameter***.

1.3 Definiciones, Acrónimos y Abreviaciones

Se recomienda el uso del Glosario [10] como referencia en caso de utilizarse algún concepto que no sea conocido por el lector y que se desee acceder a su definición. A continuación se incluye la definición de algunos de los conceptos básicos que se requieren tener claro para encarar adecuadamente el presente documento.

1.3.1 Paquete deployable (DP - Deployable package)

Archivo que empaqueta/contiene todos los archivos que conforman una aplicación a instalar (deployar). Es análogo a los archivos RPM de Linux o MSI de Windows.

1.3.2 Descriptor de aplicación (AD - Application descriptor)

Archivo que describe el DP indicando nombre, autor, versión, dependencias, entre otros elementos descriptivos. Es utilizado para realizar registrar la información de las aplicaciones al momento de deployment y para determinar previamente si las dependencias son satisfechas.

1.3.3 Especificación de Deployment (DS - Deployment specification)

Archivo que describe todas las acciones a tomar al momento de deployment así como al momento de realizar el undeployment de aplicaciones o paquetes.

1.3.4 Especificación de Recursos (RS - Resources specification)

Archivo que describe cuáles son los recursos provistos por la aplicación a deployar y cuáles son aquellos recursos requeridos y que se desean enlazar para el correcto funcionamiento de la misma.

1.3.5 Recurso (Resource)

Un recurso es cualquier elemento requerido por una aplicación y que es provisto por otra aplicación instalada en un mismo nodo objetivo. De esta manera, una aplicación pude proveer acceso a una de sus funcionalidades mediante un Web service y considerarlo un recurso provisto. Otra aplicación podrá especificar que requiere dicho recurso y el sistema le brindará de los mecanismos que puedan resolver dicho enlace.

1.3.6 Enlace (Link)

Por enlace se entiende como todo aquel canal que permita a una aplicación conectarse con otra haciendo uso de un cierto recurso que la segunda provee y que ésta requiere. Ambas se consideran "enlazables" si forman parte de un mismo nodo objetivo.

1.4 Visión General

Este documento se estructura de la siguiente manera. En el capítulo 2 se describe el contenido del ***Descriptor de Aplicación***. En el capítulo 3 se procede similarmente para el caso de la ***Especificación de Deployment***, mientras que en el capítulo 4 se detalla la ***Especificación de Recursos***. En el capítulo 5 se describe el contenido de los parámetros, es decir, los detalles del elemento ***parameter***, utilizado en todos los archivos de configuración.

2 Descriptor de Aplicación

El Descriptor de Aplicación (de aquí en más AD) es un archivo XML que contiene elementos que permiten brindar información referente a la aplicación encapsulada en un paquete a instalar.

Este archivo, de nombre ***"descr.xml"*** deberá estar contenido en el directorio DEPLOY-INF del paquete deployable, el cual es un archivo ZIP de extención ***".deploy"***.

2.1 Estructura

A continuación aparece un ejemplo de un AD, en su forma más básica. La descripción de cada uno de los elementos (tags) del XML y sus atributos, serán detallados más adelante en la sección *1.1* (no se incluyen elementos vinculados a la temática del manejo de namespaces por un tema de legibilidad de los ejemplos XML, sin embargo se deberán especificar los mismos por lo que se recomienda la lectura del schema XML (ver 1.1).

```
01. <!-- Elemento root. Incluye el identificador (id) de la aplicación -->
02. <application-descr id="MiAplicacion">
03.
04.     <!-- Nombre del paquete -->
05.     <name>Nombre de la aplicación</name>
06.
07.     <!-- Versión en formato [a.b.c] -->
08.     <version>1.0.0</vesion>
09.
10.     <!-- Autor -->
11.     <author>nikodc</author>
12.
13.     <!-- e-mail de contacto -->
14.     <email>nikodc@adinet.com.uy</email>
15.
16.     <!-- Fecha del paquete -->
17.     <date>03-12-2004 20:30:40</date>
18.
19.     <!-- Descripción del paquete -->
20.     <description>
21.         Esta es una descripción de la aplicación.
22.     </description>
23.
24. </application-descr>
```

Listado 2-1 – Archivo desc.xml básico.

2.2 Parametrizaciones y Dependencias

El AD incluye, en una versión más compleja, incluye otros elementos como ser: **Parametrizaciones** (tag *parameterization*) y **Dependencias** (tag *dependency*).

Las parametrizaciones son de utilidad para ampliar la descripción de la aplicación más allá de los elementos básicos. En tal sentido, es posible definir una parametrización "MisParametrosPersonalizados" conteniendo datos descriptivos personalizados para el tipo de dominio en donde se utilice el framework de deployment.

Las dependencias declaran requerimientos necesarios para la correcta instalación y ejecución de la aplicación a deployar. Un ejemplo básico de esto es declarar la dependencia con alguna otra aplicación previamente instalada.

La herramienta de manejo de paquetes deployables provee mecanismos de extensión que no se detallarán en este documento. La extensión, en este caso, es aplicable para agregar nuevas **Dependencias** según el uso que se le quiera dar al sistema.

A continuación se incluye un ejemplo de un Descriptor más complejo, incluyendo los elementos mencionados:

```
01. <!-- Elemento root. Incluye el identificador (id) del paquete -->
02. <application-descr id="MiAplicacion">
03.
        ............
25.
26.     <!-- Parametrizaciones -->
27.     <parameterization type="MisParametrosPersonalizados">
28.         <parameter name="Parametro1">
29.             <simple type="string">Valor del parámetro</simple>
30.         </parameter>
31.         <parameter name="Parametro2">
32.             <simple type="string">Valor del otro parámetro</simple>
33.         </parameter>
34.     </parameterization>
35.
36.     <!-- Especificación de dependencias -->
37.     <dependency name="dep1" type="ApplicationVersion">
38.         <parameter name="app">
39.           <simple type="string">OtraAplicacion2</simple>
40.         </parameter>
41.         <parameter name="version">
42.             <simple type="string">1.2.0</simple>
43.         </parameter>
44.     </dependency>
45.
46.     <dependency name="dep2" type="DependenciaPersonalizada">
47.         <parameter name="Parametro1">
48.             <simple type="string">Valor del parámetro</simple>
49.         </parameter>
50.         <parameter name="Parametro2">
51.             <simple type="string">Valor del otro parámetro</simple>
52.         </parameter>
53.     </dependency>
54.
55. </application-descr>
```

Listado 2-2 – Archivo desc.xml con elementos personalizados.

2.3 Elementos y atributos - Referencia

En esta sección se presentan todos los elementos y atributos del XML ***"desc.xml"*** junto con una breve descripción acerca de qué representan. El formato es de guía rápida de referencia ya que no es la intención entrar a fondo sino comprender en gran medida el formato de este archivo de configuración.

Elemento (tag)	Descripción breve
`application-descr`	Elemento raíz del XML. Contiene todo los elementos que describen la aplicación.
`name`	Nombre de la aplicación.
`version`	Versión de la aplicación.
`author`	Nombre del autor (responsable) de la aplicación o del empaquetado.
`email`	E-mail del autor o e-mail de contacto.
`date`	Fecha de elaboración o de empaquetado de la aplicación.
`description`	Descripción de la aplicación.
`parameterization`	Parametrización.
`dependency`	Dependencia de la aplicación.
`parameter`	Establece un parámetro dentro una parametrización o dependencia personalizada. Los elementos contenidos en "parameter" se detallan en la sección X.

Tabla 2-1 – Elementos de *desc.xml*

2.3.1 application-descr

Este elemento es la raíz del documento XML. Contiene todos los restantes elementos que describen la aplicación a deployar, incluyendo parámetros y dependencias.

Atributo	Descripción
`id`	Identificador único de la aplicación. Token único asociado a todas las versiones de la aplicación.

Tabla 2-2 – Atributos del tag *application-descr*

2.3.2 name

Este elemento se corresponde con el nombre de la aplicación. La idea del mismo es dar una forma de referenciar a la aplicación de manera más descriptiva que el identificador.

2.3.3 version

Se corresponde con la versión de la aplicación. Deberá ser de la forma "a.b.c" ya que de no ser el formato de esta manera, el sistema no podrá determinar correctamente si una versión es anterior o posterior que otra.

2.3.4 author

Nombre del autor de la aplicación o del responsable de mantener el paquete.

2.3.5 email

Email del autor de la aplicación o del responsable del paquete.

2.3.6 date

Fecha de empaquetamiento de la aplicación, o fecha de elaboración de la misma.

2.3.7 description

Breve descripción de la aplicación contenida en el paquete.

2.3.8 parameterization

Este elemento contiene una serie de parámetros (tags *parameter*), de un cierto tipo, que se corresponden con un conjunto de parámetros personalizados incluidos en el sistema mediante los mecanismos de extensión que éste provee. Es de utilidad si el uso que se le quiere dar al sistema implica que los paquetes deban tener más parámetros descriptivos que los que por defecto provee, es decir, nombre, versión, autor, email, fecha y descripción.

Atributo	Descripción
type	Tipo de la parametrización (tipo asociado a una parametrización incluida mediante los mecanismos de extensión del sistema).

Tabla 2-3 – Atributos del tag *parameterization*

2.3.9 dependency

Declaración de una dependencia. Este elemento contiene una serie de parámetros (tags *parameter*), que se corresponden con los parámetros requeridos por el correspondiente tipo de dependencia incluida en el sistema ya sea por defecto o mediante los mecanismos de extensión que éste provee.

Atributo	Descripción
name	Nombre asociado a la dependencia. Tiene como cometido permitir el rastreo de las dependencias no cumplidas.
type	Tipo de dependencia (eventualmente el tipo asociado a una dependencia incluida mediante los mecanismos de extensión del sistema).

Tabla 2-4 – Atributos del tag *dependency*

2.4 Dependencias Básicas

En esta sección se dedican algunas líneas a describir los tipos de dependencias que el sistema provee por defecto y cuáles son sus correspondientes parámetros.

2.4.1 ApplicationVersion

Dependencia que indica el requerimiento hacia una cierta versión de una determinada aplicación. Dicha aplicación, en una versión igual o superior, deberá estar instalada en el nodo objetivo para que se pueda realizar el deployment. Sus parámetros son:

Nombre	Tipo	Descripción
app	string	Identificador de la aplicación requerida.
version	string	Versión requerida de la aplicación (igual o superior).

Tabla 2-5 – Parámetros de la dependencia *ApplicationVersion*

2.5 Schema XML

```
01. <xsd:schema targetNamespace="http://www.fing.edu.uy/~pgdploy1/descr"
02.             xmlns="http://www.fing.edu.uy/~pgdploy1/descr"
03.             xmlns:par="http://www.fing.edu.uy/~pgdploy1/params"
04.             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
05.             elementFormDefault="qualified"
06.             attributeFormDefault="unqualified">
07.
08.   <!-- Elemento raíz del descriptor de paquetes deployables -->
09.   <xsd:element name="application-descr" type="ApplicationDescr"/>
10.
11.   <!-- Tipo del elemento raiz del descriptor de paquetes deployables -->
12.   <xsd:complexType name="ApplicationDescr">
13.     <xsd:sequence>
14.       <xsd:element name="name" type="xsd:string"/>
15.       <xsd:element name="version" type="Version"/>
16.       <xsd:element name="author" type="xsd:string"/>
17.       <xsd:element name="email" type="Email"/>
18.       <xsd:element name="date" type="xsd:date"/>
19.       <xsd:element name="description" type="xsd:string"/>
20.       <xsd:element name="parametrization" type="Parametrization"
21.                    minOccurs="0" maxOccurs="unbounded"/>
22.       <xsd:element name="dependency" type="Dependency"
23.                    minOccurs="0" maxOccurs="unbounded"/>
24.     </xsd:sequence>
25.     <xsd:attribute name="id" type="xsd:token" use="required"/>
26.   </xsd:complexType>
27.
28.   <!-- Tipo de la información parametrizable -->
29.   <xsd:complexType name="Parametrization">
30.     <xsd:sequence>
31.       <xsd:element name="parameter" type="par:Parameter"
32.                    maxOccurs="unbounded"/>
33.     </xsd:sequence>
34.     <xsd:attribute name="type" type="xsd:token" use="required"/>
35.   </xsd:complexType>
36.
37.   <!-- Tipo de una declaración de dependencia -->
38.   <xsd:complexType name="Dependency">
39.     <xsd:sequence>
40.       <xsd:element name="parameter" type="par:Parameter"
41.                    minOccurs="0" maxOccurs="unbounded"/>
42.     </xsd:sequence>
43.     <xsd:attribute name="name" type="xsd:token" use="required"/>
44.     <xsd:attribute name="type" type="xsd:token" use="required"/>
45.   </xsd:complexType>
46.
```

Listado 2-3 – XML Schema del AD, parte 1, Elemento raíz y tipos complejos

```
47.    <!-- Tipo "version" que se corresponde con los números de versión de
48.         Las aplicaciones. Es de la forma "a.b.c", donde a, b y c son
49.         enteros -->
50.    <xsd:simpleType name="Version">
51.      <xsd:restriction base="xsd:token">
52.        <xsd:pattern value="(0|([1-9]\d*))\.(0|([1-9]\d*))\.(0|([1-
53.  9]\d*))"/>
54.      </xsd:restriction>
55.    </xsd:simpleType>
56.
57.    <!-- Tipo basico para la verificacion de emails -->
58.    <xsd:simpleType name="Email" >
59.      <xsd:restriction base="xsd:token">
60.        <xsd:pattern value="([\.a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+(([a-zA-Z0-
61.  9_-])*\.([a-zA-Z0-9_-])+)+"/>
62.      </xsd:restriction>
63.    </xsd:simpleType>
64.
65.  </xsd:schema>
```

Listado 2-4 – XML Schema del AD, parte 2, Tipos simples

3 Especificación de Deployment

La Especificación de Deployment (de aquí en más DS) es un archivo XML que contiene detallada una serie de acciones ejecutables al momento de realizar el deployment y al momento de realizar el undeployment de paquetes en un nodo objetivo.

Este archivo, de nombre ***"deploy.xml"*** deberá estar contenido en el directorio DEPLOY-INF del paquete deployable.

3.1 Estructura

A continuación aparece un ejemplo de una DS. La descripción de cada uno de los elementos (tags) del XML y sus atributos serán detallados más adelante en la sección 3.2.

```
01. <!-- Elemento raíz de la especificación -->
02. <deployment id="MiAplicacion">
03.
04.     <!-- Acciones de deployment -->
05.     <deployment-actions>
06.         <action name="dAct0" type="ManageResources"/>
07.         <action name="dAct1" type="JARInstaller">
08.             <parameter name="files">
09.                 <complex type="collection">
10.                     <item><simple type="string">app1.jar</simple></item>
11.                     <item><simple type="string">app2.jar</simple></item>
12.                     <item><simple type="string">app3.jar</simple></item>
13.                 </complex>
14.             </parameter>
15.         </action>
16.         <action name="dAct2" type="WARInstaller">
17.             <parameter name="files">
18.                 <complex type="collection">
19.                     <item>
20.                       <simple type="string">webapp1.jar</simple>
21.                     </item>
22.                     <item>
23.                       <simple type="string">webapp2.jar</simple>
24.                     </item>
25.                     <item>
26.                       <simple type="string">webapp3.jar</simple>
27.                     </item>
28.                 </complex>
29.             </parameter>
30.         </action>
31.     </deployment-actions>
32.
33.     <!-- Acciones de undeployment -->
34.     <undeployment-actions>
35.         <action name="udAct0" type="Revert">
36.             <parameter name="action">
37.                 <simple type="string">dAct2</simple>
38.             </parameter>
39.         </action>
40.         <action name="udAct1" type="Revert">
41.             <parameter name="action">
42.                 <simple type="string">dAct1</simple>
43.             </parameter>
44.         </action>
45.         <action name="udAct2" type="Revert">
46.             <parameter name="action">
47.                 <simple type="string">dAct0</simple>
48.             </parameter>
49.         </action>
50.     </undeployment-actions>
51. </deployment>
```

Listado 3-1 – Archivo deploy.xml.

3.2 Elementos y atributos - Referencia

En esta sección se presentan todos los elementos y atributos del XML ***"deploy.xml"*** junto con una breve descripción acerca de qué representan. El formato es de guía rápida de referencia ya que no es la intención entrar a fondo sino comprender en gran medida el formato de este archivo de configuración.

Elemento (tag)	Descripción breve
`deployment`	Elemento raíz del XML. Contiene todo los elementos de la especificación.
`deployment-actions`	Elemento que contiene todas las acciones de deployment.
`undeployment-actions`	Elemento que contiene todas las acciones de undeployment.
`action`	Elemento que se corresponde con una acción a tomar.
`parameter`	Establece un parámetro dentro una parametrización o dependencia personalizada. Los elementos contenidos en "parameter" se detallan en la sección X.

Tabla 3-1 – Elementos de *deploy.xml*

3.2.1 deployment

Este elemento es la raíz del documento XML. Contiene todos los restantes elementos que especifican las acciones a tomar al momento de deployment y undeployment.

Atributo	Descripción
`id`	Identificador único de la aplicación.

Tabla 3-2 – Atributos del tag *deployment*

3.2.2 deployment-actions

Elemento que contiene todas las acciones a ejecutar al momento de deployment. Cada acción se ejecuta en el orden en que aparece especificada por lo que este hecho deberá ser tomado en cuenta.

3.2.3 undeployment-actions

Elemento que contiene todas las acciones a ejecutar al momento de undeployment. Cada acción se ejecuta en el orden en que aparece especificada por lo que este hecho deberá ser tomado en cuenta.

3.2.4 action

Este elemento encapsula una acción a tomar. Tiene una serie de parámetros que la personalizan y que incidirán en el comportamiento de la misma al momento de ejecutarse. El sistema proveerá algunas acciones por defecto pero también provee un mecanismo de extensión que permite agregar nuevos tipos de acciones. No entraremos en detalle en este aspecto en el presente documento.

Atributo	Descripción
`name`	Nombre asignado a la acción para permitir ser referenciada desde las acciones (mediante programación).
`type`	Tipo de la acción. Deberá corresponderse con algún tipo de acción existente.

Tabla 3-3 – Atributos del tag *action*

3.3 Acciones básicas

En esta sección dedicaremos algunas líneas a describir las acciones que el sistema provee por defecto y cuáles son sus correspondientes parámetros.

3.3.1 JARInstaller

Acción que se encarga de hacer el deployment de un paquete de tipo JAR en el servidor de aplicaciones J2EE en el nodo objetivo. Sus parámetros son:

Nombre	Tipo	Descripción
files	collection	Colección de rutas completas dentro del paquete hacia los archivos JAR que se quieran deployar. Cada ruta es especificada con un parámetro de tipo string.

Tabla 3-4 – Parámetros de la acción *JARInstaller*

3.3.2 WARInstaller

Acción que se encarga de hacer el deployment de un paquete de tipo WAR en el servidor de aplicaciones J2EE en el nodo objetivo. Sus parámetros son:

Nombre	Tipo	Descripción
files	collection	Colección de rutas completas dentro del paquete hacia los archivos WAR que se quieran deployar. Cada ruta es especificada con un parámetro de tipo string.

Tabla 3-5 – Parámetros de la acción *WARInstaller*

3.3.3 EARInstaller

Acción que se encarga de hacer el deployment de un paquete de tipo EAR en el servidor de aplicaciones J2EE en el nodo objetivo. Sus parámetros son:

Nombre	Tipo	Descripción
files	collection	Colección de rutas completas dentro del paquete hacia los archivos EAR que se quieran deployar. Cada ruta es especificada con un parámetro de tipo string.

Tabla 3-6 – Parámetros de la acción *EARInstaller*

3.3.4 SARInstaller

Acción que se encarga de hacer el deployment de un paquete de tipo SAR en el servidor de aplicaciones J2EE en el nodo objetivo. Sus parámetros son:

Nombre	Tipo	Descripción
files	collection	Colección de rutas completas dentro del paquete hacia los archivos SAR que se quieran deployar. Cada ruta es especificada con un parámetro de tipo string.

Tabla 3-7 – Parámetros de la acción *SARInstaller*

3.3.5 ManageResources

Esta acción indica la ejecución del registro de recursos para el mecanismo de resolución de enlaces que el framework provee.

Nombre	Tipo	Descripción
`resources`	`string`	Nombre del archivo de especificación de recursos requeridos y provistos por el paquete que se intenta deployar. Es opcional y el valor por defecto es *"DEPLOY-INF/resources.xml"*.

Tabla 3-8 – Parámetros de la acción *ManageResources*

3.3.6 Revert

Esta acción se trata de una acción a tomar al momento de undeployment y tiene como funcionalidad revertir el efecto de una acción ejecutada al momento de deployment. Sus parámetros son:

Nombre	Tipo	Descripción
`action`	`string`	Nombre de la acción a ser revertida por esta acción. Se corresponde con el valor *name* de una acción de deployment.

Tabla 3-9 – Parámetros de la acción *Revert*

3.4 Schema XML

```
01. <xsd:schema targetNamespace="http://www.fing.edu.uy/~pgdploy1/deploy"
02.               xmlns="http://www.fing.edu.uy/~pgdploy1/deploy"
03.               xmlns:par="http://www.fing.edu.uy/~pgdploy1/params"
04.               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
05.               elementFormDefault="qualified"
06.               attributeFormDefault="unqualified">
07.
08.   <!-- Elemento raíz de la especificación -->
09.   <xsd:element name="deployment-spec" type="DeploymentSpec"/>
10.
11.   <!-- Tipo del elemento raíz -->
12.   <xsd:complexType name="DeploymentSpec">
13.     <xsd:sequence>
14.       <xsd:element name="deployment-actions" type="ActionList"/>
15.         <xsd:element name="undeployment-actions" type="ActionList"/>
16.     </xsd:sequence>
17.     <xsd:attribute name="id" type="xsd:token" use="required"/>
18.   </xsd:complexType>
19.
20.   <!-- Lista de acciones -->
21.   <xsd:complexType name="ActionList">
22.     <xsd:sequence>
23.        <xsd:element name="action" type="Action"
24.                     minOccurs="0" maxOccurs="unbounded"/>
25.     </xsd:sequence>
26.   </xsd:complexType>
27.
28.   <!-- Tipo de la acciones -->
29.   <xsd:complexType name="Action">
30.     <xsd:sequence>
31.        <xsd:element name="parameter" type="par:Parameter"
32.                     minOccurs="0" maxOccurs="unbounded"/>
33.        </xsd:sequence>
34.     <xsd:attribute name="name" type="xsd:token" use="required"/>
35.     <xsd:attribute name="type" type="xsd:token" use="required"/>
36.   </xsd:complexType>
37.
38. </xsd:schema>
```

Listado 3-2 – Schema de la DS

4 Especificación de Recursos

La Especificación de Recursos (de aquí en más RS) es un archivo XML que contiene una descripción de cuáles recursos son provistos por la aplicación a instalar y cuáles son los recursos requeridos por ella para su correcto funcionamiento. Este XML es procesado mediante la inclusión de la acción ***ManageResources*** en la Especificación de Deployment correspondiente al paquete que contiene la aplicación.

Este archivo, de no especificarse una ruta completa en la Especificación de Deployment, deberá tener el nombre ***"resources.xml"*** y deberá estar contenido en el directorio ***"DEPLOY-INF"*** del paquete deployable.

4.1 Estructura

A continuación aparece un ejemplo de RS. La descripción de cada uno de los elementos (tags) del XML y sus atributos serán detallados más adelante en la sección 4.2.

```
01. <!–- Elemento raíz. Contiene la especificación de recursos provistos
02.      y requeridos -->
03. <resources-spec id="MiAplicacion">
04.
05.     <!–- Especificación de los recursos requeridos -->
06.     <required-resources>
07.         <required name="APP1.api" app="OtraAplicación" type="SessionBean"
08.                 alias="ApiApp1"  view="LocalInterface" />
09.
10.         <required name="APP1.url" app="OtraAplicación" type=”WebInterface”
11.                 alias="WebApp1" />
12.
13.         <required name="APP2.ws" app="OtraAplicación"
14.                 type=”WebService”  />
15.
16.     </required-resources>
17.
18.     <!–- Especificación de los recursos provistos -->
19.     <provided-resources>
20.         <provided name="MiApl.api" type="SessionBean">
21.             <parameter name="jndiKey">
22.                 <simple type="string">/ejb/MiApl</simple>
23.             </parameter>
24.         </provided>
25.
26.         <provided name="MyAPP.web" type="WebInterface">
27.             <parameter name="uri">
28.                 <simple type=”string”>miApl/index.html</simple>
29.             </parameter>
30.         </provided>
31.     </provided-resources>
32.
33. </resources>
```

Listado 4-1 – Archivo resources.xml

4.2 Elementos y atributos - Referencia

En esta sección se presentan todos los elementos y atributos del XML ***"resources.xml"*** junto con una breve descripción acerca de qué representan. El formato es de guía rápida de referencia ya que no es la intención entrar a fondo sino comprender en gran medida el formato de este archivo de configuración.

Elemento (tag)	Descripción breve
`resources-spec`	Elemento raíz del XML. Contiene todo los elementos de la especificación.
`required-resources`	Elemento que contiene todas las declaraciones de recursos requeridos.
`provided-resources`	Elemento que contiene todas las declaraciones de recursos provistos.
`requiered`	Elemento que se corresponde con un recurso requerido.
`provided`	Elemento que se corresponde con un recurso provisto.
`parameter`	Establece un parámetro dentro del recurso provisto.

Tabla 4-1 – Elementos de *resources.xml*

4.2.1 resources-spec

Este elemento es la raíz del documento XML. Contiene todos los restantes elementos que especifican los recursos provistos y requeridos por la aplicación a deployar.

Atributo	Descripción
`id`	Identificador de la aplicación.

Tabla 4-2 – Atributos del tag *resources*

4.2.2 required-resources

Elemento que contiene todos los recursos requeridos por la aplicación contenida en el paquete. Cada recurso es registrado en el orden que aparece listados dentro de este tag.

4.2.3 provided-resources

Elemento que contiene todos los recursos provistos por la aplicación contenida en el paquete. Cada recurso es registrado en el orden que aparece listados dentro de este tag.

4.2.4 required

Este elemento indica que un recurso es requerido. Consta principalmente de 3 atributos, el primero es el nombre del recurso requerido (identificador), el segundo es el identificador de la aplicación que provee el recurso y el tercero el tipo de recurso. El sistema proveerá algunos tipos de recursos por defecto pero también provee un mecanismo de extensión que permite agregar nuevos tipos de recursos. No entraremos en detalle en este aspecto en el presente documento.

Atributo	Descripción
name	Nombre del recurso que es requerido.
app	Identificador de la aplicación que provee el recurso.
type	Tipo de recurso. Deberá corresponderse con algún tipo de recurso existente.
alias	Nombre que se le dará al recurso para ser accedido mediante el mecanismo de resolución de enlaces que el sistema provee. Con este nombre se referencia el recurso. Es de carácter opcional por lo que en caso de no especificarse se asumirá *alias = name*.
view	Vista del recurso, es decir, indica de qué manera deseamos ver obtener el recurso. A modo de ejemlo, un recurso Session Bean (EJB) puede verse como toda la información necesaria para acceder al bean, o bien, como una instancia de la clase EJBHome. No entraremos en profundidad en este aspecto dentro del presente documento.

Tabla 4-3 – Atributos del tag *required*

4.2.5 provided

Este elemento indica que un recurso es provisto. Consta principalmente de 2 atributos, el primero es el nombre del recurso provisto (identificador) y el segundo el tipo de recurso. El sistema proveerá algunos tipos de recursos por defecto pero también provee un mecanismo de extensión que permite agregar nuevos tipos de recursos. Como ya mencionamos en el punto anterior, no entraremos en detalle en este aspecto en el presente documento.

Atributo	Descripción
name	Nombre del recurso que es provisto.
type	Tipo de recurso. Deberá corresponderse con algún tipo de recurso existente.

Tabla 4-4 – Atributos del tag *provided*

4.3 Tipos básicos de recursos

En esta sección dedicaremos algunas líneas a describir los tipos de recursos que el sistema brinda por defecto y cuáles son sus correspondientes parámetros.

4.3.1 WebInterface

Este recurso se trata simplemente de una URI por donde ingresar a un elemento de la interfaz web de una aplicación instalada. Sus parámetros son:

Nombre	Tipo	Descripción
uri	string	URI de la interfaz web relativa al nodo objetivo

Tabla 4-5 – Parámetros del recurso *WebInterface*

4.3.2 SessionBean

Este recurso se trata de un punto de acceso mediante JNDI a un SessionBean (EJB), es decir, su Home Interface ya sea local o remota. Sus parámetros son:

Nombre	Tipo	Descripción
jndiHome	string	Clave JNDI en donde encontrar el recurso remoto.
jndiLocalHome	string	Clave JNDI en donde encontrar el recurso local.

Tabla 4-6 – Parámetros del recurso *SessionBean*

4.3.3 WebService

Este recurso se trata simplemente de una URI por donde acceder al WebService. Sus parámetros son:

Nombre	Tipo	Descripción
uri	string	URI del Web Service.

Tabla 4-7 – Parámetros del recurso *WebService*

4.4 Schema XML

```
01. <xsd:schema targetNamespace="http://www.fing.edu.uy/~pgdploy1/resources"
02.             xmlns="http://www.fing.edu.uy/~pgdploy1/resources"
03.             xmlns:par="http://www.fing.edu.uy/~pgdploy1/params"
04.             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
05.            elementFormDefault="qualified"
06.            attributeFormDefault="unqualified">
07.
08.   <!-- Elemento raíz de la especificación -->
09.   <xsd:element name="resources-spec" type="ResourcesSpec"/>
10.
11.   <!-- Tipo del elemento raíz -->
12.   <xsd:complexType name="ResourcesSpec">
13.     <xsd:sequence>
14.        <xsd:element name="provided-resources" type="ProvidedList"/>
15.        <xsd:element name="required-resources" type="RequiredList"/>
16.     </xsd:sequence>
17.     <xsd:attribute name="id" type="xsd:token" use="required"/>
18.   </xsd:complexType>
19.
20.   <!-- Lista de recursos provistos -->
21.   <xsd:complexType name="ProvidedList">
22.     <xsd:sequence>
23.       <xsd:element name="provided" type="Provided"
24.                    minOccurs="0" maxOccurs="unbounded"/>
25.     </xsd:sequence>
26.   </xsd:complexType>
27.
28.   <!-- Recurso provisto -->
29.   <xsd:complexType name="Provided">
30.     <xsd:sequence>
31.        <xsd:element name="parameter" type="par:Parameter"
32.                     minOccurs="0" maxOccurs="unbounded"/>
33.     </xsd:sequence>
34.     <xsd:attribute name="name" type="xsd:token" use="required"/>
35.     <xsd:attribute name="type" type="xsd:token" use="required"/>
36.   </xsd:complexType>
37.
38.   <!-- Lista de recursos requeridos -->
39.   <xsd:complexType name="RequiredList">
40.     <xsd:sequence>
41.       <xsd:element name="required" type="Required"
42.                    minOccurs="0" maxOccurs="unbounded"/>
43.     </xsd:sequence>
44.   </xsd:complexType>
45.
46.   <!-- Recurso requerido -->
47.   <xsd:complexType name="Required">
48.     <xsd:attribute name="name"  type="xsd:token" use="required"/>
49.     <xsd:attribute name="app"   type="xsd:token" use="required"/>
50.     <xsd:attribute name="type"  type="xsd:token"/>
51.     <xsd:attribute name="alias" type="xsd:token"/>
52.     <xsd:attribute name="view"  type="xsd:token"/>
53.   </xsd:complexType>
54.
55. </xsd:schema>
```

Listado 4-2 – Schema de la RS

5 Establecimiento de Parámetros

En este capítulo se describen todos los aspectos vinculados al elemento XML ***parameter***, utilizado para el establecimiento de parámetros. Este elemento puede formar parte de cualquiera de todos los diferentes archivos de configuración contenidos en un paquete deployable.

A modo de ejemplo, en el descriptor de paquetes deplorables es posible especificar grupos de parámetros personalizados. Cada uno de ellos puede contener diversos parámetros a establecer. Continuando con el ejemplo, uno tipo de grupo de parámetros podría llamarse “MyParams” en donde se desea establecer el valor del string “Param1” y el valor de la colección de enteros “Param2”; esto se especificaría como se ve en el primer ejemplo de estructura del elemento ***parameter*** en el Listado 5-1.

5.1 Estructura

A continuación aparece un ejemplo simple del elemento ***parameter***. La descripción de cada uno de los subelementos (tags) del mismo y sus atributos serán detallados más adelante en la sección 5.2.

```
<!-- Seteo de un parámetro simple -->
<parameter name="Param1">
    <simple type="string">
        Este es el valor de Param1
    </simple>
</parameter>

<!-- Seteo de un parámetro complejo -->
<parameter name="Param2">
    <complex type="collection">
        <item>
            <simple type="integer">
                1
            </simple>
        </item>
        <item>
            <simple type="integer">
                2
            </simple>
        </item>
    </complex>
</parameter>
```

Listado 5-1 – Ejemplo básico de estructura del elemento *parameter*

5.2 Elementos y atributos - Referencia

En esta sección se presentan todos los elementos y atributos que forman parte de una declaración de un parámetro junto con una breve descripción acerca de qué representan. El formato es de guía rápida de referencia ya que no es la intención entrar a fondo sino comprender en gran medida el formato de este archivo de configuración.

Elemento (tag)	Descripción breve
`parameter`	Elemento que especifica el establecimiento del valor de un parámetro.
`simple`	Elemento que representa el valor de un parámetro de contenido simple, es decir, tipos de datos simples (ver 5.2.2).
`complex`	Elemento que representa el valor de un parámetro de contenido complejo, es decir, colecciones, mapeos, y similares (ver *5.2.3*).
`item`	Elemento que se corresponde con un ítem contenido dentro de un parámetro complejo (ver 5.2.4).

Tabla 5-1 – Elementos participantes en la especificación de parámetros

5.2.1 parameter

Elemento que representa el establecimiento del valor de un parámetro Contiene un elemento ***simple*** o ***complex*** que representa el valor del parámetro.

Atributo	Descripción
`name`	Nombre del parámetro a establecer.

Tabla 5-2 – Atributos del tag *parameter*

5.2.2 simple

Elemento que representa un tipo de valor de parámetro. En este caso, se trata de un tipo de dato simple. Se consideran simples los siguientes tipos de datos:

- **boolean**
  Igual al tipo de dato xsd:boolean.
- **date**
  Igual al tipo de dato xsd:date.
- **double**
  Igual al tipo de dato xsd:double.
- **email**
  Formato “xxxxx@xxxxx”.
- **float**
  Igual al tipo de dato xsd:float.
- **integer**
  Igual al tipo de dato xsd:int.
- **string**
  Igual al tipo de dato xsd:string.
- **version**
  Formato “xx.xx.xx” (tres numero separados por puntos).

Su contenido es textual y se interpreta en función del atributo ***type***.

Atributo	Descripción
type	Tipo del parámetro simple. Alguno entre ***boolean, date, double, email, float, integer, string, version***.

Tabla 5-3 – Atributos del tag *simple*

5.2.3 complex

Elemento que representa un tipo de valor de parámetro. En este caso, se trata de un tipo de dato complejo. Se consideran complejos los siguientes tipos de datos:

- **collection**
  Colección de ítems donde cada uno puede ser de tipo simple o complejo.
- **map**
  Mapeo del tipo <clave, valor>. Puede (y conviene) verlo como una colección de ítems donde cada uno debe ser de tipo ***map.entry***.
- **map.entry**
  Ítem contenido en un ***map***. La clave se setea mediante el parámetro **"key"** mientras que el valor se setea mediante el parámetro **"value"**. Contiene en primer lugar una secuencia de tags ***item*** que se corresponden con ítems contenidos dentro del tipo complejo y a continuación contiene una secuencia de parámetros que eventualmente se deseen establecer.

Atributo	Descripción
type	Tipo del parámetro complejo. Alguno entre ***collection, map, map.entry***.

Tabla 5-4 – Atributos del tag *complex*

Ejemplos de tipos complejos pueden verse en la sección *5.3*.

5.2.4 item

Este elemento encapsula un ítem contenido dentro de un tipo complejo. Contiene un elemento ***simple*** o bien uno elemento ***complex***.

5.3 Ejemplo de parámetros complejos

En esta sección del anexo, se incluyen ejemplos de especificación de tipos complejos que permitan comprender un poco más la composición de los mismos.

5.3.1 Usando el tipo *collection*

A continuación se adjunta un ejemplo de una colección de email's.

```
<!-- Colección de emails -->
<complex type="collection">
    <item>
        <simple type="email">
            user1@pgdploy1.com.uy
        </simple>
    </item>
    <item>
        <simple type="email">
            user2@pgdploy2.com.uy
        </simple>
    </item>
</complex>
```

Listado 5-2 – Ejemplo de colección de emails

5.3.2 Usando el tipo *map*

A continuación se adjunta un ejemplo de un mapeo <string, email>.

```
<!-- Mapeo <string, email> -->
<complex type="map">
    <item>
        <complex type=”map.entry”>
            <parameter name=”key”>
                <simple type="string">user1</simple>
            </parameter>
            <parameter name=”value”>
                <simple type="email">user1@pgdploy1.com.uy</simple>
            </parameter>
        </complex>
    </item>
    <item>
        <complex type=”map.entry”>
            <parameter name=”key”>
                <simple type="string">user2</simple>
            </parameter>
            <parameter name=”value”>
                <simple type="email">user2@pgdploy1.com.uy</simple>
            </parameter>
        </complex>
    </item>
</complex>
```

Listado 5-3 – Ejemplo de mapeo <string, email>

5.3.3 Un ejemplo más sofisticado

A continuación se adjunta un ejemplo de un mapeo de <strings, colección de email>.

```
<!-- Mapeo <string, collection<email>> -->
<complex type="map">
  <item>
    <complex type=”map.entry”>
      <parameter name=”key”>
        <simple type="string">user1</simple>
      </parameter>
      <parameter name=”value”>
        <complex type="collection">
          <item>
            <simple type="email">user1@pgdploy1.com.uy</simple>
          </item>
          <item>
            <simple type="email">user1@fing.edu.uy</simple>
          </item>
        </complex>
      </parameter>
    </complex>
  </item>
  <item>
    <complex type=”map.entry”>
      <parameter name=”key”>
        <simple type="string">user2</simple>
      </parameter>
      <parameter name=”value”>
        <complex type="collection">
          <item>
            <simple type="email">user2@pgdploy1.com.uy</simple>
          </item>
          <item>
            <simple type="email">user2@fing.edu.uy</simple>
          </item>
        </complex>
      </parameter>
    </complex>
  </item>
</complex>
```

Listado 5-4– Ejemplo de mapeo <string, email>

5.4 Schema XML

```
01. <xsd:schema targetNamespace="http://www.fing.edu.uy/~pgdploy1/params"
02.             xmlns="http://www.fing.edu.uy/~pgdploy1/params"
03.             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
04.             elementFormDefault="qualified"
05.             attributeFormDefault="unqualified">
06.
07.   <!-- Tipo de los parametros -->
08.   <xsd:complexType name="Parameter">
09.     <xsd:choice>
10.         <xsd:element name="simple" type="SimpleElement"/>
11.         <xsd:element name="complex" type="ComplexElement"/>
12.     </xsd:choice>
13.     <xsd:attribute name="name" type="xsd:token" use="required"/>
14.   </xsd:complexType>
15.
16.   <!-- Tipo de los elementos simples -->
17.   <xsd:complexType name="SimpleElement">
18.     <xsd:simpleContent>
19.       <xsd:extension base="xsd:string">
20.         <xsd:attribute name="type" type="SimpleElementType"
21.                        use="required"/>
22.         </xsd:extension>
23.     </xsd:simpleContent>
24.   </xsd:complexType>
25.
26.   <!-- Tipo de los elementos complejos -->
27.   <xsd:complexType name="ComplexElement">
28.     <xsd:sequence>
29.         <xsd:element name="item" type="Item"
30.                      minOccurs="0" maxOccurs="unbounded"/>
31.         <xsd:element name="parameter" type="Parameter"
32.                      minOccurs="0" maxOccurs="unbounded"/>
33.     </xsd:sequence>
34.     <xsd:attribute name="type" type="ComplexElementType" use="required"/>
35.   </xsd:complexType>
36.
37.   <!-- Tipo de los items -->
38.   <xsd:complexType name="Item">
39.     <xsd:choice>
40.         <xsd:element name="simple" type="SimpleElement"/>
41.         <xsd:element name="complex" type="ComplexElement"/>
42.     </xsd:choice>
43.   </xsd:complexType>
44.
45.   <!-- Tipos validos para el atributo "type" de los elementos simples -->
46.   <xsd:simpleType name="SimpleElementType">
47.     <xsd:restriction base="xsd:anyURI">
48.         <xsd:enumeration value="string"/>
49.         <xsd:enumeration value="integer"/>
50.         <xsd:enumeration value="long"/>
51.         <xsd:enumeration value="double"/>
52.         <xsd:enumeration value="float"/>
53.         <xsd:enumeration value="date"/>
54.         <xsd:enumeration value="version"/>
55.         <xsd:enumeration value="email"/>
56.     </xsd:restriction>
57.   </xsd:simpleType>
58.
59.   <!-- Tipos validos del atributo "type" de los elementos complejos -->
60.   <xsd:simpleType name="ComplexElementType">
61.     <xsd:restriction base="xsd:anyURI">
62.         <xsd:enumeration value="map"/>
63.         <xsd:enumeration value="map.entry"/>
64.         <xsd:enumeration value="collection"/>
65.     </xsd:restriction>
66.   </xsd:simpleType>
67.
68. </xsd:schema>
```

Listado 5-6 – Schema de la especificación de parámetros

6 Referencias

[1] Sitio oficial del proyecto LinkAll
http://www.link-all.org

[2] Descripción del proyecto LinkAll
http://www.fing.edu.uy/inco/proyectos/linkall/Documentos/Descripcion-LinkAll.doc
INCO – Facultad de Ingeniería – UdelaR, 2003.

[3] Documento de descripción de la arquitectura LinkAll (v1.10)
INCO – Facultad de Ingeniería – UdelaR, 2005.

[4] Documento de requerimientos del sistema (v1.5)
Código: **RDR**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[5] Documento de casos de uso (v1.5)
Código: **RMC**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[6] Documento de descripción de la arquitectura (v1.3)
Código: **RDA**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[7] Doc. técnica – Especificación de Deployment (v1.2)
Código: **IDT-DEPSPEC**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[8] Doc. técnica – Especificación de Recursos (v1.2)
Código: **IDT-RESSPEC**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[9] Doc. técnica – Especificación de Parámetros en XML (v1.1)
Código: **IDT-PARAMS**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 – pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[10] Glosario (v1.1)
Código: **RG**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 – pgdploy1
INCO – Facultad de Ingeniería – UdelaR

UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

ANEXO E

MANUAL DE REALIZACIÓN DE PLUGINS

Estudiantes
Nicolás Doroskevich
Sebastián Moreira

Tutores
Federico Piedrabuena
Raúl Ruggia

Proyecto de grado 2004
Facultad de Ingeniería - Universidad de la República

Índice general

1. Introducción

1.1 Propósito

El presente documento tiene como objetivo describir el mecanismo de plugins incluido en el framework de deployment. A través del mencionado mecanismo, es posible extenderlo y personalizarlo para el uso en una cierta plataforma objetivo.

1.2 Alcance

En este documento se explicará cómo se debe proceder para incluir una nueva funcionalidad y cuáles son las exigencias al respecto. En tal sentido, se incluirá una descripción general del mecanismo de extensión, haciendo hincapié en la elaboración de plugins, para posteriormente explicar más detalladamente cómo se realizan los mismos en función del tipo de elemento a incorporar. Los elementos incorporables al framework son:

- ***Dependencias***
  Una dependencia se concibe como una restricción sobre una aplicación. Mediante el establecimiento de dependencias, es posible controlar que una aplicación pueda ser incorporada a la plataforma objetivo en base el cumplimiento de las mismas. Las dependencias pueden ser incluidas en el Descriptor de Aplicación [5].

- ***Acciones de deployment y undeployment***
  Este concepto se corresponde con una tarea a realizar al momento de efectuarse un deployment o un undeployment de una cierta aplicación. Las acciones son incluidas dentro de la Especificación de Deployment [7].

- ***Recursos***
  Un recurso es cualquier elemento provisto por una aplicación y que eventualmente puede ser requerido por otras. Una aplicación que desea exponer y/o utilizar un recurso, necesita notificar al sistema a través de la Especificación de Recursos [7]. Debido a que no se conoce de antemano cuáles son las características de los recursos existentes en la plataforma objetivo, los tipos de recursos puede ser incorporados en la forma de plugins.

En este documento no se pretende ver detalladamente la interna del framework de deployment para así comprender cómo funcionan los componentes encapsulados en la forma de plugins, sino que pretende brindar una referencia rápida mediante la cual entender cómo se especializa y extiende el mismo.

1.3 Definiciones, Acrónimos y Abreviaciones

Se recomienda el uso del Glosario [9] como referencia en caso de utilizarse algún concepto que no sea conocido por el lector y que se desee acceder a su definición.

1.4 Visión General

Este documento se estructura de la siguiente manera. En el capítulo 2 se incluye una descripción general del mecanismo de extensión, centrado en la elaboración de plugins. En los siguientes capítulos se ingresa más en detalle a los casos particulares. En el capítulo 3 se describe cómo incorporar una nueve dependencia, en el capítulo 4 se procede similarmente explicando cómo incluir una nueva acción, y finalmente, en el capítulo 5 se describe cómo incorporar un nuevo tipo de recurso.

2 Mecanismo de Extensión

El mecanismo de extensión fue desarrollado de tal manera de que el mismo sea homogéneo, es decir, que no se hagan grades diferencias entre lo que implica agregar una nueva dependencia, una nueva acción, o un nuevo recurso.

En primer lugar es necesario definir qué tipo de elemento se pretende incorporar (dependencia, acción o recurso). Una vez definido el elemento, se debe proceder a dar implementación al mismo mediante la realización de la interfaces apropiadas que el framework provee. La forma de implementación, y las interfaces involucradas, se detallarán en los siguientes capítulos, distinguiendo según el tipo de elemento a incorporar.

Una vez implementada la extensión, es momento de empaquetar la misma en la forma de plugin. Un plugin no es más que una aplicación común y corriente. Como toda aplicación, incluye un Descriptor de Aplicación, una Especificación de Deployment, y eventualmente puede contener una Especificación de Recursos. Por otro lado, también incluye un contenido binario, en este caso, la implementación de la extensión empaquetada en una librería Java (archivo JAR).

El punto distinto, el cual le otorga a esta aplicación las cualidades de un plugin, es la inclusión de acciones específicas para el registro de nuevos componentes en la Especificación de Deployment. Veamos un ejemplo concreto:

```
01. <deployment-actions>
02.   <action name="dAct1" type="JARInstaller">
03.     <parameter name="files">
04.       <complex type="collection">
05.         <item>
06.           <simple type="string">MiNuevaAccion.jar</par:simple>
07.         </item>
08.       </complex>
09.     </parameter>
10.   </action>
11.   <action name="dAct2" type="PlugActionType">
12.     <parameter name="type">
13.       <simple type="string">MiAccion</simple>
14.     </parameter>
15.     <parameter name="className">
16.       <simple type="string">package.my.MiAccion</simple>
17.     </parameter>
18.   </action>
19. </deployment-actions>
```

Listado 2-1 – Ejemplo de DS, registrando un plugin

En el listado anterior, se aprecia un fragmento de la Especificación de Deployment en donde se indica la ejecución de dos acciones al momento de deployment. En la línea **02** se indica la instalación de una librería JAR en la plataforma objetivo. Luego de ejecutada la acción, la misma queda incorporada al ambiente objetivo y entonces ya es posible realizar el registro de la extensión, en este caso, una nueva acción. En la línea **11**, se indica, mediante la ejecución de la acción ***PlugActionType***, el registro de una nueva acción, pasando el nombre del nuevo tipo de acción y su respectiva clase Java que realiza la implementación, como parámetros. De esta manera, con posterioridad es posible referenciar la nueva acción mediante el nombre de tipo, como se muestra en el siguiente fragmento de otra Especificación de Deployment:

```
01. <deployment-actions>
02.   <action name="dAct1" type="MiAccion">
03.     <parameter name="miParametro">
04.       <simple type="string">Valor de miParametro</par:simple>
05.     </parameter>
06.   </action>
07. </deployment-actions>
```

Listado 2-2 – Ejemplo de uso de la nueva acción registrada

Como puede apreciarse, el nombre del tipo es asociado a la clase Java indicada en el Listado 2-1. Cuando en la línea **02** se indica la ejecución de la acción, el framework delega el trabajo al nuevo componente para que realice las tareas que le corresponda realizar.

Con este ejemplo sencillo, se pretende haber ilustrado en gran medida el mecanismo de registro de plugins, por lo que en los siguientes capítulos estudiaremos cada caso particular. Vale aclarar, que existe una acción de registro, por tipo de plugin, es decir:

- ***PlugDependencyType***: registro de depdendencias.
- ***PlugActionType***: registro de acciones.
- ***PlugResourceType***: registro de tipos recursos.

3 Incorporando nuevas dependencias

Para incorporar una nueva dependencia, es necesario realizar la implementación de la interfaz ***IDependency*** definida en el framework.

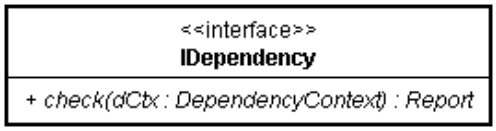


Fig. 3-1 – Interfaz *IDependency*

Se deberá dar implementación a las siguientes operaciones:

Operación	**Check**
Parámetros	`dCtx` – Información contextual utilizada para obtener información que puede ser de utilidad para la resolución del chequeo de la dependencia. Contiene información acerca de cuáles aplicaciones se pretenden incorporar o eliminar, permitiendo a la dependencia consultar toda información anexada, es decir, sus Descriptores de Aplicación y sus Especificaciones de Deployment.
Retorno	Reporte conteniendo un resumen de todo hecho acontecido durante el procesamiento de esta operación, mediante un conjunto de notas, advertencias y errores. En caso de no cumplirse la dependencia, debe incorporarse al menos un error.
Descripción	Realiza el control del cumplimiento de la dependencia en la plataforma objetivo, eventualmente considerando la información contextual pasada como parámetro.
Excepciones	No tiene.

Consideremos un caso de ejemplo. Supongamos que se desea incorporar una nueva dependencia capaz de controlar que la plataforma objetivo posea una memoria RAM de cierta capacidad. Para ello, se deberá realizar la interfaz ***IDependency***, implementando la clase ***MemoryRequeriment***, tal como se aprecia en la figura 3-2.

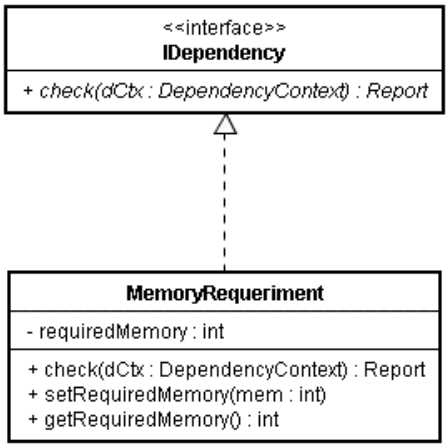


Fig. 3-2 – Implemenación de *IDependency*

La clase *MemoryRequeriment*, puede ser parametrizada incluyendo, por supuesto, la capacidad requerida. El campo ***requiredMemory*** almacena dicha información, utilizada en el método ***check*** para comparar contra los recursos que el sistema de base posee. En caso de cumplirse el requerimiento establecido, se retornará un reporte sin errores, en caso contrario, se retornará un reporte conteniendo al menos un error.

Una vez implementada la dependencia, ésta puede ser encapsulada en una aplicación (plugin) estableciendo en la Especificación de Deployment la ejecución de la acción ***PlugDependencyType***, tal como se muestra en el siguiente listado.

```
01. <action name="regPlugin" type="PlugDependencyType">
02.   <parameter name="type">
03.     <simple type="string">RequiredMemoryControl</simple>
04.   </parameter>
05.   <parameter name="className">
06.     <simple type="string">my.dependecies.MemoryRequeriment</simple>
07.   </parameter>
08. </action>
```

Listado 3-1 – Registro de una nueva dependencia

La dependencia, una vez registrada, puede ser utilizada para el control del requerimiento de cierta capacidad de memoria. En el siguiente listado, se incluye un fragmento de un Descriptor de Aplicación correspondiente a una aplicación que requiere 256 MB de RAM para su correcta ejecución.

```
01. <dependency type="RequiredMemoryControl" name="memReq">
02.   <parameter name="requiredMemory">
03.     <simple type="integer">256</simple>
04.   </parameter>
05. </dependency>
```

Listado 3-2 – Utilizando la dependencia *MemoryRequeriment*

Es importante destacar dos elementos. En primer lugar, la dependencia se referencia por su nombre de tipo y no por el nombre de clase. En segundo lugar, cada campo se establece usando la notación de parámetros definida en [8].

Sobre el segundo punto, notar que existe una correspondencia entre el nombre y tipo del campo expresado en XML, y el nombre y campo definido en la clase Java. El valor establecido para el campo se establece utilizando JavaBeans por lo que es necesario la correspondencia entre el nombre de la propiedad en XML y en Java. Para una mejor comprensión de esta característica, puede ser necesario recurrir a [10].

4 Incorporando nuevas acciones

Para incorporar una nueva acción, debe realizarse una implementación de la interfaz ***IAction*** definida en el framework.

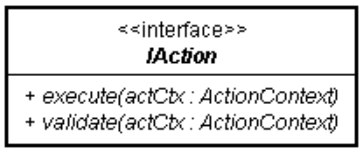


Fig. 4-1 – Interfaz *IAction*

Se deberá dar implementación a las siguientes operaciones:

Operación	**`validate`**
Parámetros	`actCtx` – Información contextual. Desde este objeto, la acción puede acceder a información relevante para la resolución de las funcionalidades provistas por la acción.
Retorno	No tiene.
Descripción	Realiza la validación de los parámetros establecidos de la acción. En caso de encontrarse errores, deberán incluirse en el reporte contenido en `actCtx`.
Excepciones	No tiene.

Operación	**`execute`**
Parámetros	`actCtx` – Información contextual. Desde este objeto, la acción puede acceder a información relevante para la resolución de las funcionalidades provistas por la acción.
Retorno	No tiene.
Descripción	Ejecuta la acción en base a los parámetros establecidos. En caso de encontrarse errores, deberán incluirse en el reporte contenido en `actCtx`.
Excepciones	No tiene.

Consideremos un caso de ejemplo. Supongamos que se desea incorporar una nueva acción capaz de ejecutar un script en una base de datos. Para ello, se debe implementar la interfaz ***IAction***; sea dicha implementación la clase ***SQLScriptExecution***.

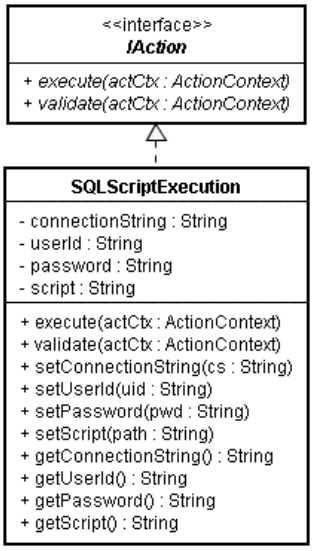


Fig. 4-2 – Implemenación de *IAction*

La clase *SQLScriptExecution*, puede ser parametrizada incluyendo el script de conexión, el usuario, el password y la ruta al script SQL. El método ***execute*** realiza la conexión a la base y ejecuta el script en ella. Los métodos ***validate*** valida que todos sus parámetros sean correctos.

Una vez implementada la acción, ésta puede ser encapsulada en una aplicación (plugin), estableciendo en la Especificación de Deployment la ejecución de la acción ***PlugActionType***, tal como se muestra en el siguiente listado.

```
01. <action name="regPlugin" type="PlugActionType">
02.   <parameter name="type">
03.     <simple type="string">RunSQL</simple>
04.   </parameter>
05.   <parameter name="className">
06.     <simple type="string">my.actions.SQLScriptExecution</simple>
07.   </parameter>
08. </action>
```

Listado 4-1 – Registro de una nueva acción

La acción, luego del registro, puede ser referenciada desde la Especificación de Deployment de cualquier otra aplicación, tal como se aprecia en el Listado 4-2.

```
01. <action name="runSQL" type="RunSQL">
02.   <parameter name="connectionString">
03.     <simple type="string"> jdbc:hsqldb:hsql://localhost:1234/</simple>
04.   </parameter>
05.   <parameter name="userId">
06.     <simple type="string">myself</simple>
07.   </parameter>
08.   <parameter name="pasword">
09.     <simple type="string">mypass</simple>
10.   </parameter>
11.   <parameter name="script">
12.     <simple type="string">SQL/myscript.sql</simple>
13.   </parameter>
14. </action>
```

Listado 4-2 – Utilizando la acción *SQLScriptExecution*

Algunas acciones puede ser reversibles, es decir, capaces de ejecutar las mismas tareas pero en sentido contrario. Esta característica puede ser aprovechada al momento de undeployment. Supongamos que una acción ***InstallGame*** permite instalar un juego en la plataforma objetivo. El juego podría consistir en una librería Java (JAR), pero no es importante discutir este aspecto para comprender el ejemplo. La acción puede ser reversible si además de ***IAction*** implementa la interfaz ***IRevertible***. El diseño de *InstallGame* se aprecia en la siguiente figura:

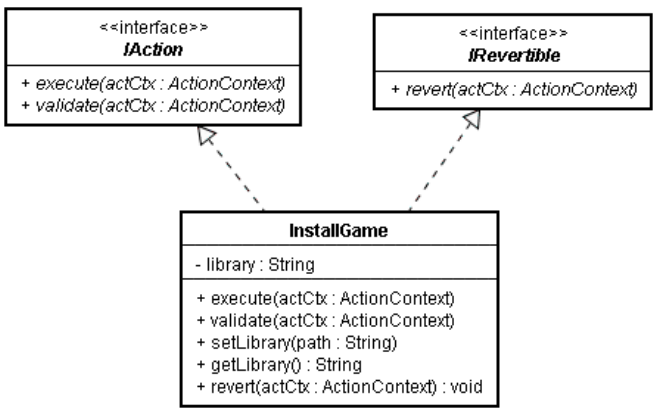


Fig. 4-3 – Implemenación de *IAction* e *IRevertible*

Adicionalmente a las operaciones de *IAction*, las siguientes operaciones definidas en *IRevertible* deben ser implementadas:

Operación	**`revert`**
Parámetros	`actCtx` – Información contextual. Desde este objeto, la acción puede acceder a información relevante para la resolución de las funcionalidades provistas por la acción.
Retorno	No tiene.
Descripción	Ejecuta la acción en base a los parámetros establecidos, pero en sentido inverso, revirtiendo cambio realizado en el otros sentido. En caso de encontrarse errores, deberán incluirse en el reporte contenido en `actCtx`.
Excepciones	No tiene.

El método ***revert***, para el caso del ejemplo, deberá eliminar el juego de la plataforma objetivo. Para ello, podrá valerse de la información establecida al momento de deployment. A continuación un fragmento de la Especificación de Deployment de un juego desarrollado para la plataforma objetivo del ejemplo, mostrando la acción de deployment y el posterior uso de la acción ***Revert*** encargada de ejecutar la operación de reversión correspondiente a una acción de deployment.

```
01. <deployment-actions>
02.   <action name="dAct1" type="InstallGame">
03.     <parameter name="library">
04.       <simple type="string">bin/game.jar</simple>
05.     </parameter>
06. </deployment-actions>
07.
08. <undeployment-actions>
09.   <action name="uAct2" type="Revert">
10.     <parameter name="action">
11.       <simple type="string">dAct2</ simple>
12.     </parameter>
13. </undeployment-actions>
```

Listado 4-4 – Revirtiendo una acción que implementa *IRevertible*

La acción ***Revert*** recibe como parámetro el nombre de la acción de deployment que se desea revertir. Como *InstallGame* es una acción revertible, se ejecuta la operación *revert* siendo establecidos los mismos parámetros utilizados al momento de deployment.

5 Incorporando nuevos recursos

6 Referencias

[1] Documento de descripción de la arquitectura LinkAll (v1.10)
INCO – Facultad de Ingeniería – UdelaR, 2005.

[2] Documento de requerimientos del sistema (v1.5)
Código: **RDR**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[3] Documento de casos de uso (v1.5)
Código: **RMC**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[4] Documento de descripción de la arquitectura (v1.3)
Código: **RDA**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[5] Doc. técnica – Descriptor de Aplicación (v1.2)
Código: **IDT-DESCR**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[6] Doc. técnica – Especificación de Deployment (v1.2)
Código: **IDT-DEPSPEC**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 - pgdploy1
INCO – Facultad de Ingeniería - UdelaR

[7] Doc. técnica – Especificación de Recursos (v1.2)
Código: **IDT-RESSPEC**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 - pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[8] Doc. técnica – Especificación de Parámetros en XML (v1.1)
Código: **IDT-PARAMS**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2005 – pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[9] Glosario (v1.1)
Código: **RG**
Nicolás Doroskevich, Sebastián Moreira
Proyecto de grado 2004 – pgdploy1
INCO – Facultad de Ingeniería – UdelaR

[10] JavaBeans
http://java.sun.com/products/javabeans/
Sun Microsystems

UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

ANEXO F

PLAN DE PROYECTO

Estudiantes
Nicolás Doroskevich
Sebastián Moreira

Tutores
Federico Piedrabuena
Raúl Ruggia

Proyecto de grado 2004
Facultad de Ingeniería - Universidad de la República

Índice general

1. Introducción

Este documento se corresponde con el plan del proyecto de grado "**Utilitarios para deployment de aplicaciones en un servidor basado en J2EE"** o **PGDPLOY1** ([2]) que se encuentra enmarcado en el contexto del proyecto LinkAll ([1]).

Tiene como objetivo desarrollar utilitarios y componentes para el ambiente LinkAll, que permita la inclusión de nuevas aplicaciones (deployment), las cuales se van a ejecutar en un contexto de un servidor de aplicaciones J2EE, utilizando una misma interfaz al exterior (html, web-services y flash) y una base de datos común.

1.1 Alcance del Proyecto

El alcance del proyecto se definió en función de la priorización de los requerimientos y de cuáles funcionalidades son necesarias y cuáles otras son discutibles en función del tiempo según lo discutido en sucesivas regiones con el cliente (docentes supervisores). Las funcionalidades de "Deployment" y "Undeployment" se consideran necesarias y las funcionalidades relacionadas con la "Descarga de paquetes" y "Obtención de paquetes" como secundarias, por lo tanto, las primeras se desea que sean resueltas "elegantemente" ya que son el punto fuerte del proyecto de grado, mientras que las segundas pueden ser implementadas de manera más rudimentaria. En este punto se destaca nuevamente que las funcionalidades secundarias serán más o menos rudimentarias en función del tiempo y el cliente prefirió ajustar el alcance de las mismas a lo largo del proyecto.

1.2 Prinicipales funciones del software y su prioridad

Las principales funcionalidades del software a desarrollar son las que aparecen listadas a continuación. Para conocer más en profundidad los detalles de dichas funcionalidades es posible consultar el Documento de requerimientos del sistema ([3]). Las mismas se listan en orden de prioridad.

Prioridad	Funcionalidades
1	➢ Deployment/Undeployment/Upgrades ➢ Seguimiento de problemas durante el deployment ➢ Verificación de autenticidad de paquetes
2	➢ Descarga de paquetes ➢ Administración de paquetes ➢ Browser de paquetes en el servidor de descargas
3	➢ Log del sistema ➢ Manejo de eventos ➢ Extensibilidad del sistema
4	➢ Instalación desde el nodo soporte ➢ Obtención de paquetes en forma local

Tabla 1.2-1 – Principales funcionalidades priorizadas

Las funcionalidades de prioridad **1** son obligatorias. Las de prioridad **2** no dejan de ser obligatorias pero podrán ser más o menos rudimentarias en función de la disponibilidad de tiempo y los ajustes en el alcance a lo largo del proyecto. Las funcionalidades de prioridad **3** son funcionalidades importantes para una mayor calidad del producto final pero son propuestas por el equipo y aceptadas por el cliente, por lo tanto, el equipo determinará sus cualidades y cuán rudimentariamente se podrán implementar. Por último, las funcionalidades de prioridad **4** son en principio opcionales y, posiblemente, puedan ser implementadas en forma básica.

1.3 Ítems de comportamiento y performance

Un elemento importante a considerar es que el software deberá ser fácil de mantener y modificar. Por otro lado, se deberá tener en cuenta que los usuarios finales del sistema serán personas con poco conocimiento de computación por lo que el sistema deberá ser lo más amigable y fácil de usar posible.

1.4 Restricciones técnicas y de Gestión

La principal restricción técnica se relaciona con la interfaz web del proyecto puesto que deberá adaptarse al estándar definido para la interfaz LinkAll. Otra restricción a considerar se relaciona con el software de desarrollo a utilizar puesto que deberá ser el mismo utilizado para el desarrollo del proyecto LinkAll (no obligatoriamente, pero se recomienda) y la plataforma sobre la cual deberá correr el sistema, la cual es Linux.

Con respecto a la Gestión, la principal restricción es que se deberá aplicar RUP como modelo de proceso. Es decir, un modelo iterativo e incremental centrado en la arquitectura y guiado en los casos de uso. Las fases e iteraciones del presente proyecto serán descriptas más adelante en este mismo documento.

2. Recursos del proyecto

En esta sección se detallan los recursos humanos, de hardware, software, herramientas y otros que son necesarios para construir el software.

El grupo de trabajo está formado por dos integrantes: Nicolás Doroskevich y Sebastián Moreira. Ambos integrantes tienen limitada su disponibilidad horaria debido a que trabajan de becarios/pasantes y cumplen un horario de 6 horas desde la mañana y hasta las 15:30 hrs. de Lunes a Viernes. Por otro lado, el primero desarrolla actividades laborales de 15 hrs. semanales dentro del propio proyecto LinkAll como miembro del equipo de desarrollo. Además, asisten a clase en Facultad de Ingeniería (UdelaR) por lo que no pueden dedicar 100% de su tiempo a actividades del proyecto. Debido a esto es que los picos de trabajo se podrán apreciar los fines de semana principalmente.

Con respecto al hardware y software disponibles, dichos elementos serán los que cada miembro del equipo tenga disponible en sus respectivos hogares:

- **Nicolás Doroskevich**
  Hardware: AMD Athlon XP 2600+ con 512 MB RAM (DDR).
  Software: Windows XP (Pro) y Linux Fedora Core 2. MS Office, Visio, Jude UML, JBoss.

- **Sebastián Moreira**
  Hardware: Intel Celeron 500 Mhz. con 320 MB RAM.
  Software: Windows 98 y Linux Fedora Core 1. MS Office, Visio, Jude UML, JBoss.

3. Estimaciones del proyecto

El equipo de trabajo se enfrenta por primera vez a un proyecto de las características del actual proyecto. Debido a esto, las estimaciones se realizaron en base a la experiencia de los clientes (docentes supervisores) ya que no son clientes ajenos a la materia informática, sino que por el contrario, son docentes del Instituto de Computación, Facultad de Ingeniería (UdelaR) con mayor experiencia en el área que cualquiera de los miembros del grupo.

Sin embargo, el equipo, tras ingresar más a fondo en la realidad del proyecto en el cual se ve envuelto comprendió que a pesar de no tener puntos propios de referencia, de todas maneras le era posible realizar sus propias estimaciones en función del análisis realizado del problema, y aplicando sus conocimientos y experiencias adquiridas en otros proyectos anteriores a pesar de no ser “tan” similares al actual.

Realizadas las puntualizaciones anteriores, se concluyó que el proyecto requerirá al menos hasta Junio de 2005 para su finalización. Si bien esta estimación fue realizada en concordancia con todos los interesados en el proyecto, el equipo presentó un plan de trabajo personalizado en donde se definen fases e iteraciones, cada una con sus propios objetivos y actividades. Dicho cronograma fue estudiado y aprobado en reuniones realizadas durante el mes de Septiembre de 2004 y será ajustado a lo largo del proyecto. La versión actual del cronograma se detalla en la sección 6.

También debido a lo mencionado al comienzo de esta sección, no se realzaron estimaciones de líneas de código. Típicamente esto es realizado mediante el uso de Puntos de función pero con variantes que permitan aplicar la técnica en un proyecto no relacionado con un Sistema de Información. El equipo desconoce en profundidad tales técnicas y considera suficiente hacer uso de las estimaciones realizadas en base a la experiencia ya que comenzar un estudio en éstos temas llevaría a un gasto de tiempo innecesario desde su punto de vista.

4. Gestión de riesgos

En esta sección se presenta una lista de riesgos encontrados por el equipo, así también se cuenta con una clasificación de los mismos según impacto en el proyecto, probabilidad de ocurrencia y una descripción de cómo mitigarlos.

La clasificación de riesgos según impacto será la siguiente:

- **Muy alto**
  Se considerará a aquellos riesgos tales que, de darse, el proyecto se vea seriamente comprometido.
- **Alto**
  Se considerará a aquellos riesgos tales que, de darse, el proyecto se vea retrasado teniendo que volver a ejecutar actividades pertenecientes a fases anteriores del proyecto.

- **Medio**
  Se considerará a aquellos riesgos tales que, de darse, el proyecto se vea retrasado teniendo que volver a ejecutar actividades pertenecientes a iteraciones anteriores dentro de una misma fase del proyecto.

- **Bajo**
  Se considerará a aquellos riesgos tales que, de darse, el proyecto se vea retrasado pero no teniendo necesidad de volver a atrás y volver a llevar a cabo actividades de fases o iteraciones anteriores.

Con respecto a la probabilidad de ocurrencia:

- **Altamente probable**
  El grupo considera que hay una probabilidad de entre 75% y 100% de ocurrencia.

- **Medianamente probable**
  El grupo considera que hay una probabilidad de entre 40% y 75% de ocurrencia.

- **Poco probable**
  El grupo considera que hay una probabilidad de entre 0% y 40% de ocurrencia.

4.1 Tabla de Riesgos

A continuación, se presenta la tabla de riesgos del proyecto completa, la cual incluye el nombre del riesgo detectado, la probabilidad de ocurrencia asociada y el impacto en el proyecto de la ocurrencia del mismo. Además incluye cómo mitigarlos y monitorearlos. Los riesgos aparecen ordenados según impacto sobre el proyecto.

4.1.1 Problemas de comunicación con el cliente.

4.1.1.1 Descripción

Debido a problemáticas en la comunicación con el cliente (docentes supervisores) es posible que algún requerimiento sea mal comprendido, o que se realicen tareas innecesarias o de mal forma. Este elemento generará sin duda retrasos y su impacto dependerá de las características del problema de comunicación.

4.1.1.2 Impacto

Alto.

4.1.1.3 Probabilidad de ocurrencia

Altamente probable.

4.1.1.4 Mitigación

Como mecanismo de mitigación se realizarán resúmenes de reuniones con los docentes y presentaciones mediante diapositivas. Con respecto a los resúmenes, tanto los docentes como los miembros del grupo deberán recurrir a esta documentación para revisar que realmente se hayan comprendido los requerimientos discutidos en las reuniones. Con respecto a las diapositivas, su principal interés es poder trasmitir de una manera ágil y simplificada los avances y las propuestas que el grupo genere desde una reunión anterior a una reunión siguiente en función de poder expresar oralmente y visualmente las mismas.

4.1.1.5 Monitoreo

En las reuniones grupales se estudiarán las reacciones de los clientes ante las propuestas realizadas, las respuestas a las dudas que se le presentaron y las tareas que nos hayan asignado con la intención de detectar situaciones de mala comunicación.

4.1.2 No comprensión de requerimientos

4.1.2.1 Descripción

En caso de falta de comprensión de un requerimiento, se generarán retrasos y su impacto dependerá de las características de cada uno de los requerimientos mal comprendidos.

4.1.2.2 Impacto

Alto.

4.1.2.3 Probabilidad de ocurrencia

Medianamente probable.

4.1.2.4 Mitigación

Se utilizarán similares mecanismos de mitigación que los definidos en la sección 4.1.1.4 para el caso de problemas de comunicación. Se agregará otro mecanismo de mitigación, la realización de prototipos para validar con el cliente (docentes supervisores).

4.1.2.5 Monitoreo

Se intentará mantener un diálogo fluido con el cliente. Con él se monitoreará que los requerimientos hayan sido debidamente comprendidos y posteriormente implementados de manera correcta.

4.1.3 Requerimientos cambiantes

4.1.3.1 Descripción

Un requerimiento no estable resulta problemático en función de cuánto repercuta principalmente en la arquitectura. También puede derivar en rediseño y/o en re-implementación.

4.1.3.2 Impacto

Alto.

4.1.3.3 Probabilidad de ocurrencia

Medianamente probable.

4.1.3.4 Mitigación

Se utilizarán similares mecanismos de mitigación que los definidos en la sección 4.1.2.4 para el caso de no comprensión de requerimientos.

4.1.3.5 Monitoreo

Se intentará mantener un diálogo fluido con el cliente. Con él se discutirán los cambios, se negociarán los mismos y se validarán.

4.1.4 Tiempos acotados

4.1.4.1 Descripción

El equipo de trabajo tiene otras actividades de importancia por lo que el tiempo para dedicarle al proyecto deberá ser tal que permita desempeñar esas otras actividades. Lo anterior, sumando a los retrasos típicos que puedan surgir durante el desarrollo conforman un riesgo de gran importancia y requiere ser debidamente controlado y monitoreado.

4.1.4.2 Impacto

Medio.

4.1.4.3 Probabilidad de ocurrencia

Medianamente probable.

4.1.4.4 Mitigación

Se llevará un calendario/cronograma actualizado en el sitio web del grupo (ver [2]) cuyas fechas tope se intentará respetar para evitar retrasos pero a su vez dichas fechas deberán ser realistas. También se mitigará el tema “tiempo” mediante la realización de informes que resuman las reuniones tanto con los docentes como las grupales; de esta forma se evitará la pérdida de tiempo ocasionado debido al no recordar hecho, tareas asignadas, dichos, etc.

4.1.4.5 Monitoreo

Se revisará el calendario de trabajo inmediato (mensual) en cada reunión grupal o con los docentes. El calendario de trabajo general, con sus fases e iteraciones, deberá ser discutido necesariamente con los docentes.

4.1.5 Arquitectura inadecuada

4.1.5.1 Descripción

Una mala arquitectura puede ser fatal para el proyecto. El trabajo necesario para rediseñar la arquitectura puede implicar la no finalización del proyecto principalmente si se detecta el problema en etapas avanzadas del desarrollo.

4.1.5.2 Impacto

Muy alto.

4.1.5.3 Probabilidad de ocurrencia

Medianamente probable.

4.1.5.4 Mitigación

Para mitigar este problema se considera vital tener una buena comunicación con el cliente y prototipar la arquitectura en una etapa temprana del proyecto. Se utilizarán los mecanismos definidos en la sección 4.1.2.4.

4.1.5.5 Monitoreo

Se monitoreará el riesgo del mismo modo que el descrito en la sección 4.1.2.5 con el agregado de que los prototipos deberán hacer hincapié en la arquitectura como punto fuerte.

4.1.6 Mala verificación

4.1.6.1 Descripción

Realizar una correcta verificación, bien planeada e implementada, es fundamental para aumentar la calidad del producto y no sufrir retrasos.

4.1.6.2 Impacto

Medio.

4.1.6.3 Probabilidad de ocurrencia

Medianamente probable.

4.1.6.4 Mitigación

Para mitigar el riesgo se realizarán ajustes al "Plan de verificación" (VPP) en acuerdo con los docentes y con el asesor en temas de calidad (Diego Vallespir).

4.1.6.5 Monitoreo

Se monitoreará como consecuencia de las reuniones con el asesor en temas de calidad (Diego Vallespir).

4.1.7 Confusión y pérdida de versiones

4.1.7.1 Descripción

Confundir y perder versiones de entregables puede ser fatal en caso de que no se tengan los recursos de tiempo como para volver a generar las versiones faltantes.

4.1.7.2 Impacto

Medio.

4.1.7.3 Probabilidad de ocurrencia

Medianamente probable.

4.1.7.4 Mitigación

Se realizará una gestión de configuración básica y se utilizará un repositorio CVS para almacenar los entregables y tener la posibilidad de acceder a cualquier versión generada de los mismos.

4.1.7.5 Monitoreo

Se auditarán las líneas base previamente a ser marcadas en el CVS. Se revisará periódicamente las últimas versiones (tag HEAD de CVS) de todos los elementos en el repositorio para determinar que no falte ninguno o que haya algún otro tipo de error o problema.

4.1.8 Problemas de hardware

4.1.8.1 Descripción

Las principales herramientas durante el desarrollo son aplicaciones corriendo sobre el hardware disponible por los miembros del grupo. Una problemática en este sentido puede repercutir en el retraso del proyecto.

4.1.8.2 Impacto

Medio.

4.1.8.3 Probabilidad de ocurrencia

Poco probable.

4.1.8.4 Mitigación

Se respaldarán periódicamente todos los documentos generados así también como ejecutables y código fuente.

4.1.8.5 Monitoreo

Se revisarán periódicamente todos los respaldos realizados, o al menos el último realizado.

5. Agenda del proyecto

En esta sección se detalla el cronograma a seguir para el desarrollo del proyecto. Se definen fases e iteraciones las cuales se corresponden con el modelo de proceso de referencia a utilizar, el cual en este caso se trata del modelo RUP. Debido a las características de proyecto y el número pequeño de integrantes del grupo, la instanciación del modelo se presenta bastante simplificada y no se realizan aquellas actividades (y entregables) que el grupo considerada prescindibles o poco importantes.

5.1 Diagramas de planificación

En esta sección se incluyen dos diagramas. El primero (*Diagrama 6.1-1*) muestra las fases e iteraciones ubicadas cronológicamente y el segundo (*Diagrama 6.1-2*) muestra las actividades a realizar también en orden cronológico. En la siguiente sección se detallarán cada uno de los elementos que aparecen en estos diagramas.

Fases e Iteraciones	*SET*	*OCT*	*NOV*	*DIC*	*ENE*	*FEB*	*MAR*	*ABR*
Elaboración	E1							
Construcción			C1		C2		C3	
Transición								T1 | T2

Diagrama 5.1-1 – Fases e iteraciones del proyecto

En el anterior diagrama se puede apreciar con claridad la ausencia de la definición de la fase inicial (Inception en RUP). Esto se debe a que al momento de la realización de la primera versión de este Plan del proyecto el grupo consideró haber pasado dicha etapa de acuerdo a las exigencias y objetivos establecidos en el modelo de referencia.

Con respecto a las fases se definieron 3: Elaboración, Construcción y Transición. Dentro de cada una de ellas se realizarán iteraciones en donde cada una de ellas tiene sus propias metas y actividades, del mismo modo y en concordancia con las metas y actividades de la fase que las contiene. Una descripción más fina de estas fases e iteraciones aparecerá en la siguiente sección.

Cronograma	SET	OCT	NOV	DIC	ENE	FEB	MAR	ABR	Area
Requerimientos y CU's									Req
Arquitectura									Dis
Diseño									
Enganches con APIs									
Enganche con UI Link All									
Enganche con BD y ACL									
Enganche con el federador									
Implementar Prototipo									Imp
Implementar Sistema									
Prueba de funcionalidades									Test
Pruebas de componentes									
Pruebas unitarias de métodos									
Documentación del sistema									Doc
Manual de usuario									
Documentación de instalación									
Instalación									Ins

Diagrama 5.1-2 – Cronograma

En este último diagrama se aprecian las actividades, a grandes rasgos, que se van a realizar y en qué períodos de tiempo. Se categorizan en función del área a la que las actividades mencionadas pertenecen.

Es importante destacar que el anterior cronograma está alineado con el cronograma general definido por el grupo de trabajo del proyecto LinkAll por lo que se extiende hasta el mes de Abril tal como aparece definido en el mencionado cronograma.

Los períodos de tiempo para las actividades podrán ser ajustados previo acuerdo con los docentes supervisores del proyecto, dado el caso de necesidad debido a atrasos o inconvenientes, o bien, debido a la terminación temprana de alguna actividad anterior.

5.2 Plan de trabajo

En este punto se detallan las características de cada fase e iteración en función del modelo de referencia RUP. Se incluyen objetivos y principales actividades.

5.2.1 Elaboración

5.2.1.1 Objetivos

A continuación se detallan los objetivos de la fase.

- Analizar, diseñar e implementar la arquitectura del Sistema a construir, llegando al final de la fase con el ejecutable de la línea base de la arquitectura, el cual es la base para las funcionalidades que se irán sumando en la fase de Construcción para llegar al ejecutable final del sistema.
- Los riesgos identificados del proyecto están siendo mitigados y monitoreados, a la vez que se buscan nuevos riesgos.
- La gestión del proyecto es controlada: se realiza el seguimiento y control del proyecto.
- La calidad del proyecto es controlada: se revisa la calidad de los productos y el ajuste al proceso.
- La gestión de configuración es controlada: se controlan las versiones.

5.2.1.2 Principales actividades

Las principales actividades a desarrollar son:

- Relevar y validar requerimientos con el cliente
  - Priorizarlos
  - Detallarlos
  - Casos de prueba
- Identificar y validar casos de uso con el cliente
  - Priorizarlos
  - Detallarlos
  - Casos de prueba
- Análisis de casos de uso, paquetes y clases
  - Definir vista de análisis
- Diseñar casos de uso, subsistema y principales clases
- Diseñar la base de datos
- Definir una arquitectura estable
  - Vista del modelo de análisis

- Vista del modelo de casos de uso
- Vista del modelo de distribución

➢ Implementar prototipo a nivel de arquitectura

➢ Planificar verificación

- Definir plan de pruebas para la próxima fase/iteración

➢ Verificar documentos

➢ Pruebas del sistema (prototipo)

5.2.2 Construcción

5.2.2.1 Objetivos

A continuación se detallan los objetivos de la fase.

➢ Completar el Análisis, Diseño, Implementación y Verificación de todas las funcionalidades del Sistema.

➢ Alcanzar paralelismo en la Implementación, ya que las distintas funcionalidades pueden ser implementadas en paralelo.

➢ Desarrollar Manuales de Usuario y Documentación Técnica.

➢ La Gestión del proyecto es controlada: se realiza el seguimiento y control del proyecto.

➢ La Calidad del proyecto es controlada: se revisa la calidad de los productos y el ajuste al proceso, se realizan Revisiones técnicas.

➢ La Gestión de configuración es controlada: se controlan las versiones y los cambios, se audita la línea base.

5.2.2.2 Principales actividades

Las principales actividades a desarrollar son:

➢ Diseñar casos de uso, subsistemas y clases

➢ Diseñar la base de datos

➢ Ajustar arquitectura

- Agregar vista del modelo de diseño
- Agregar vista del modelo de implementación
- Agregar otras vistas de ser necesario

➢ Implementar el sistema

- Iteración 1: funcionalidades básicas del sistema

- Iteración 2: funcionalidades de nivel de calidad 5
- Iteración 3: todas las funcionalidades dentro de alcance

➢ Verificar documentos

➢ Pruebas del sistema, casos de uso y componentes

➢ Planificar implantación

5.2.2.3 Objetivos/actividades por iteración

5.2.2.3.1 Iteración 1

➢ Diseñar casos de uso, subsistemas y clases (funcionalidades básicas)

➢ Diseñar la base de datos

➢ Ajustar arquitectura

- Agregar vista del modelo de diseño (funcionalidades básicas)

➢ Implementar funcionalidades básicas del sistema

➢ Verificar documentos

➢ Pruebas del sistema, casos de uso y componentes (funcionalidades básicas)

5.2.2.3.2 Iteración 2

➢ Finalizar y verificar todos los modelos

➢ Diseñar casos de uso, subsistemas y clases (funcionalidades de nivel 5)

➢ Finalizar arquitectura

- Agregar vista del modelo de diseño (funcionalidades de nivel 5)
- Agregar vista de implementación

➢ Implementar funcionalidades de nivel 5

➢ Verificar documentos

➢ Pruebas del sistema, casos de uso y componentes (funcionalidades de nivel 5)

➢ Pruebas de regresión

➢ Revisar la calidad de los productos

5.2.2.3.3 Iteración 3

➢ Finalizar diseño del sistema

➢ Implementar resto de las funcionalidades

➢ Verificar documentos

➢ Pruebas del sistema, casos de uso y componentes

- Pruebas de regresión
- Revisar la calidad de los productos
- Realizar documentación técnica y del sistema
- Realizar manuales
- Planificar implantación

5.2.3 Transición

5.2.3.1 Objetivos

A continuación se detallan los objetivos de la fase.

- Asegurar que el software esté disponible para los usuarios finales.
- Ajustar los errores y defectos encontrados.
- Capacitar a los usuarios y proveer el soporte técnico necesario.
- Verificar que el producto cumpla con las especificaciones.
- Dejar disponible el software para el uso por parte de los usuarios.

5.2.3.2 Principales actividades

Las principales actividades a desarrollar son:

- Pruebas del sistema, casos de uso y componentes
- Implantar el sistema
- Ajustar los errores y defectos encontrados
    - Ajustes mínimos en implementación
- Finalizar documentación técnica y manuales
    - Capacitar usuarios
- Verificar documentos

6. Mecanismos de control y ajuste

En este punto se detallan las técnicas que serán utilizadas para realizar el control y ajuste del proyecto.

6.1 Mecanismos de SQA

Los mecanismos de verificación se encuentran alineados con las exigencias definidas para los niveles de calidad 3 y 5 según corresponda

6.2 Mecanismos de SCM

Se utilizará un servidor CVS para gestionar las versiones de código y documentación. La nomenclatura a utilizar es la utilizada para los entregables del proceso Java del curso Proyecto de Ingeniería de software (ver [4]). El grupo es pequeño por lo que no considera necesario realizar un Plan de SCM para desempeñar correctamente las actividades de SCM. Sin embargo, si podrá ser viable realizar alguna documentación de guía para la nomenclatura de entregables, versionado, uso del CVS, etc.

6.3 Mecanismos de Verificación

Los mecanismos de verificación se encuentran alineados con las exigencias definidas para los niveles de calidad 3 y 5 según corresponda. Para entrar más en profundidad se deberá recurrir al Plan de verificación y validación (VPP).

6.4 Mecanismos de Gestión del Proyecto

Este plan se ajustará al final de cada iteración en caso de que el equipo lo considere necesario.

7. Referencias

[1] Sitio oficial del proyecto LinkAll

http://www.link-all.org

[2] Sitio del proyecto pgdploy1

http://www.fing.edu.uy/~pgdploy1

[3] Documento de requerimientos del sistema (RDR) v1.5 – Octubre 2004

Nicolás Doroksevich, Sebastián Moreira

INCO, Facultad de Ingeniería, UdelaR

[4] Sitio del modelo de proceso Java (2003)

http://www.fing.edu.uy/inco/cursos/ingsoft/pis/sitioweb/experiencia2003/Java03/index.htm

Proyecto de ingeniería se software

INCO, Facultad de Ingeniería, UdelaR

[5] Sitio del modelo de proceso RUP – versión de evaluación

Racional, IBM

UTILITARIOS PARA DEPLOYMENT DE APLICACIONES EN UNA PLATAFORMA BASADA EN SERVICIOS SOBRE J2EE

ANEXO G

INFORME DE VERIFICACIÓN Y VALIDACIÓN

Estudiantes
Nicolás Doroskevich
Sebastián Moreira

Tutores
Federico Piedrabuena
Raúl Ruggia

Proyecto de grado 2004
Facultad de Ingeniería - Universidad de la República

Índice general

1. Introducción

En el presente documento se tratarán los temas relacionados con la verificación y validación llevada a cabo por el grupo durante el transcurso del proyecto. Éste tema tomó relevancia en el proyecto debido a los niveles de calidad exigidos para el proyecto por parte de los tutores del mismo.

Los niveles de calidad fueron definidos por el Ing. Diego Vallespir en su tesis de maestría ([8]), y seleccionados por los tutores para el proyecto. Vallespir también oficio de auditor en los temas relacionados con la calidad del proyecto. Se seleccionó más de un nivel de calidad, se definieron dos niveles, nivel de calidad tres y cinco. Este ultimo nivel se aplicó a las funcionalidades consideradas críticas, mientras que el nivel tres fue aplicado a el resto de las funcionalidades.

En este anexo se comenzará viendo el plan de verificación y validación elaborado por el grupo y validado por Vallespir al inicio del proyecto. En este plan se define el propósito de la verificación y la validación, las actividades a realizar durante el proceso, los cubrimientos seleccionados, el ciclo de vida de la actividad y los reportes a realizar.

Luego se verán los casos y procedimientos de prueba realizados para el sistema. Se especificará como fueron validados los requerimientos funcionales y los casos de uso para el sistema. También se mostrarán los casos de prueba utilizados para la verificación y validación de la versión final del sistema, así como también los resultados obtenidos.

Por último se verán las pruebas planificadas para realizar sobre los principales componentes del sistema así como los resultados obtenidos para las mismas.

2. Plan de verificación y validación

En esta sección se detallará el plan de verificación y validación realizado por el grupo para cumplir con las exigencias de calidad definidas.

2.1 Propósito

El proyecto de software al que se le aplicará este plan consiste, en un software que provea herramientas de deployment de aplicaciones en servidores Link All [6]. El software podrá obtener las aplicaciones desde un servidor de descargas (nodo soporte o nodo intermedio) ubicado en un nodo administrativo de la red Link All y las instalará en el nodo de aplicaciones (nodo objetivo). El deployment deberá ser los más automatizado posible. Todo el software que necesite la aplicación a ser deployada para poder ser ejecutada, deberá estar previamente instalado en el servidor.

El propósito de este documento es el definir un plan, que guíe las actividades de verificación a realizar durante el transcurso del proyecto. En este se especificaran los siguientes ítems:

- Organización de la verificación.
- Determinación de la agenda de actividades.
- Descripción de los recursos disponibles.
- Asignación del trabajo de verificación para cada fase.
- Herramientas, técnicas y metodologías a utilizar para llevar a cabo la verificación.

Las actividades enmarcadas en este plan de Verificación y Validación serán llevadas a cabo sobre los artefactos (componentes) del producto en construcción. Esto determina que se verificarán:

- Los documentos generados en las líneas de trabajo básicas, es decir los referidos al producto en sí mismo y no aquellos puramente administrativos.
- Los componentes del sistema en forma individual e integrados, el ejecutable de cada iteración, la versión implantable.

Los objetivos que plantea la actividad de verificación detallada por este plan es la de obtener un producto que satisfaga las siguientes características:

- Posea todas las cualidades requeridas por el cliente, tanto funcionales como no funcionales.
- Asegure la trazabilidad de los distintos componentes del mismo (análisis- diseño, diseño – implementación).
- Haya pasado por un exhaustivo control de defectos, que minimice la probabilidad de ocurrencia de fallas y reduzca los costos de mantenimiento en la fase de explotación.

2.2 Definiciones

- Trazabilidad - Cualidad de un producto de ser compatible y no ambiguo respecto de sus conceptos.
- Hito - determina el comienzo/ fin de una actividad.
- Anomalía - Hecho observado en la documentación o funcionamiento del software, que se desvía de las expectativas previas del producto o el documento.
- Caso de prueba - Secuencia de acciones donde se tomarán ciertas entradas de datos, se las ingresará al sistema y se buscará la confirmación de un flujo de acciones por parte del sistema y una salida acorde a la esperada.
- Prueba de integración - Pruebas realizadas al integrar componentes unitarios, o subsistemas.
- Prueba de aceptación - Pruebas que se realizarán en el ambiente real en conjunto con el cliente para su validación.

2.3 Descripción de la Verificación y Validación

En esta sección se describen los recursos necesarios para realizar la verificación en el marco del proyecto. Se especificarán organización, agenda, recursos, responsabilidades, y herramientas, técnicas y metodologías.

2.3.1 Organización

Las tareas del área de verificación, que incluyen actividades durante todo el proceso de desarrollo del producto, las llevarán a cabo los integrantes del grupo (dos miembros). Por ser un grupo pequeño, no se realizaron subdivisiones dentro del mismo para dedicar un subgrupo a la verificación y validación.

2.3.1.1 Interacción del equipo con los usuarios

Relacionamiento del grupo con los clientes:
Clientes: brindarán aportes sobre los puntos más importantes a verificar, en base a sus conocimientos de los requerimientos y la realidad donde será implantado el producto final. Para ello se intentará fijar reuniones de consulta y validación. Además participará de las pruebas de aceptación y pruebas beta del producto.

2.3.1.2 Consideraciones generales

Se intentarán verificar las últimas versiones de los documentos generados y presentados a los clientes, luego de las reuniones con los mismos, con el objetivo de acelerar el proceso de corrección de errores.

En caso de detectar fallas en el producto, estas serán corregidas a la brevedad posible.

2.3.2 Agenda Principal

Ver documento de plan de proyecto ([1]).

A continuación se especifica qué actividades deben ser realizadas con anterioridad a las actividades de verificación y cómo estas influyen en su realización:

- **Elaboración del Plan de Verificación**
  No existen actividades previas definidas.

- **Elaboración del Plan de Pruebas del Prototipo**
  Para poder realizar el Plan de Pruebas del Prototipo es necesaria la especificación del mismo. Al momento de escribir este documento, la especificación del prototipo aún no está finalizada.

- **Elaboración del Plan de Pruebas**
  Esta actividad tiene como entradas los Modelos de Casos de Uso, Análisis (si corresponde), Diseño e Implementación, los cuales se van refinando con el transcurso de las iteraciones. El plan de pruebas asociado a cada iteración se realizará en base a las versiones más recientes de estos documentos que se encuentren disponibles.

- **Verificar documentos**
  La verificación de un documento se realizará inmediatamente después de que una versión del mismo sea discutida con los clientes, con el objetivo de que las correcciones a los errores encontrados sean incorporadas a la siguiente versión del mismo.

- **Diseñar Casos de Pruebas**
  Esta actividad tiene como entradas los Modelos de Casos de Uso, siempre se utilizará la última versión disponible para diseñar los casos de pruebas.

- **Prueba del Prototipo**
  Esta actividad tiene dos entradas: el prototipo y los casos de pruebas para el prototipo.

- **Prueba de Integración**
  Esta actividad tiene como entradas el plan de pruebas, los casos de pruebas de integración y los componentes ya probados. El objetivo es verificar que los componentes desarrollados interactúen de la forma esperada, y en caso negativo generar reportes que registren las anomalías encontradas.

- **Prueba del Sistema**
  Esta actividad se realiza una vez completada la prueba de integración. El objetivo de la misma es verificar que el sistema se comporta de la forma esperada y si cumple con los requerimientos especificados de forma satisfactoria. En caso de detectarse algún error, éste se incluirá en el reporte correspondiente.

2.3.3 Resumen de Recursos

Los recursos para esta tarea son todos los integrantes del grupo (dos personas).

2.3.4 Responsabilidades

Ambos integrantes del grupo son responsables de realizar todas las tareas de planificación para el área, definiendo en cada caso, cuando, como y que productos serán verificados, cuando, como y que pruebas serán realizadas sobre el software en desarrollo.

2.3.5 Herramientas, Técnicas y metodologías

Las herramientas con las que se cuenta para la verificación son:

- Entorno de desarrollo Eclipse
- Lenguaje de programación Java, versión 1.4.2
- Servidor de aplicaciones J2EE JBoss
- Editores de texto
- Planillas de Excel
- JUnit

Metodologías:

- **Inspecciones**

  El objetivo es revisar el componente y la documentación del mismo desde un enfoque formal, para asegurar que realice lo que marca el diseño, que su codificación siga el estándar de implementación y que su documentación sea consistente y completa. Cada integrante del equipo revisará el componente y su documentación en forma individual y luego se tendrá una reunión del grupo para discutir los errores que identificó cada uno y establecer su relevancia. Esta metodología en principio será aplicada a los componentes críticos (Nivel 5).

- **Enfoque de caja Negra**

  Se considera el componente como una caja negra que de acuerdo a ciertas entradas especificadas en la interfaz del componente produce una salida especificada en el diseño. Los casos de prueba a aplicar se realizarán considerando las diferentes posibilidades de datos de entrada. Los resultados obtenidos por la serie de pruebas se compararán con las salidas esperadas de acuerdo a la especificación del componente.

2.4 Criterios de Cubrimientos

2.4.1 Cubrimiento de especificación de requerimientos

Se tendrá al menos un caso de prueba para cada requerimiento especificado.

2.4.2 Cubrimiento de casos de uso

Para los casos de uso que cubran funcionalidades de nivel de calidad 3, se realizara un cubrimiento del escenario del flujo normal. Para los casos de uso que cubran funcionalidades de nivel 5, se realizara un cubrimiento de todos los escenarios posibles del caso de uso.

2.4.3 Cubrimiento de componentes

Se cubrirán todas las funcionalidades que la componente brinda.

2.5 Ciclo de Vida de la Verificación & Validación

Cuando se realiza la actividad de verificación y validación de documentos se identifican los posibles errores y se generan los reportes correspondientes. La meta de estos reportes es que en las próximas versiones de los documentos verificados asociados a dichos reportes se incorporen las correcciones marcadas en los mismos. Por lo tanto, se ha definido como política a seguir que los documentos serán verificados inmediatamente después de que sean presentados a los clientes, con el fin de acelerar el feedback, minimizar las anomalías y cumplir con el objetivo anteriormente mencionado.

2.5.1 Línea de Trabajo de Requerimientos

Describe como se realizará la verificación en esta área durante el desarrollo del proyecto.

2.5.1.1 Tareas

- Revisión de los documentos producidos durante la etapa de requerimientos, los principales documentos a revisar son:
    - Documento de Requerimientos.
    - Alcance del Sistema.
    - Modelo de Casos de Uso.
    - Descripción de la Arquitectura.
    - Glosario.

2.5.1.2 Métodos y Criterios

Los documentos se verificarán manualmente y se buscará que sean completos, correctos, consistentes, no ambiguos, realistas, verificables y trazables.

En las nuevas versiones de los documentos se deberán considerar los reportes de verificación anteriores.

2.5.1.3 Entradas / salidas

A continuación se especifican las entradas necesarias y salidas previstas para la verificación de documentos.

Entradas:

- Documento a verificar.

Salidas:

- Reportes de verificación.

2.5.1.4 Recursos

Todos los integrantes del grupo.

2.5.2 Línea de Trabajo de Análisis

Describe como se realizará la verificación en esta área durante el desarrollo del proyecto.

2.5.2.1 Tareas

- Revisión de los documentos producidos durante la etapa de requerimientos, los principales documentos a revisar son:
    - Modelo de Análisis.

2.5.2.2 Métodos y Criterios

Se revisará la correctitud, completitud, consistencia y no-ambigüedad del análisis, dándole un énfasis mayor a la trazabilidad y consistencia con respecto a los documentos de requerimientos.

2.5.2.3 Entradas / salidas

A continuación se especifican las entradas necesarias y salidas previstas para la verificación de documentos esta sección.

Entradas:

- Documentos de Requerimientos y Análisis (documento de requerimientos, modelo de análisis, etc.)

Salidas:

- Reportes de Verificación.

2.5.2.4 Recursos

Todos los integrantes del grupo.

2.5.3 Línea de Trabajo de Diseño

Describe como se realizará la verificación en esta área durante el desarrollo del proyecto.

2.5.3.1 Tareas

- Revisión de los documentos producidos durante la etapa de diseño, los principales documentos a revisar son:
    - Modelo de Diseño.
    - Modelo de Datos.

2.5.3.2 Métodos y Criterios

Se revisará la correctitud, completitud, consistencia y no-ambigüedad del diseño, poniendo mayor énfasis en la trazabilidad y consistencia con respecto a los documentos producidos en las etapas de requerimientos y análisis.

2.5.3.3 Entradas / salidas

A continuación se especifican las entradas necesarias y salidas previstas para la verificación de documentos de esta sección.

Entradas:

- Documentos de requerimientos, análisis y diseño (modelo de datos, modelo de diseño, modelo de análisis, documento de requerimientos, etc.)

Salidas:

- Reportes de verificación.

2.5.3.4 Recursos

Todos los integrantes del grupo.

2.5.4 Línea de Trabajo de Implementación

Describe como se realizara la verificación en esta área durante el desarrollo del proyecto.

2.5.4.1 Tareas

- Revisión de los documentos producidos durante la etapa de diseño, los principales documentos a revisar son:
    - Estándar de Implementación.
    - Modelo de Implementación.
    - Plan de Integración.
    - Documentación Técnica.
- Planificar las Pruebas.
- Diseñar los Casos de Prueba.

2.5.4.2 Métodos y Criterios

- Verificación de Documentos.

  En relación a la verificación de los estándares se evaluará que sean útiles y fáciles de seguir. El resto de los documentos se verificarán de idéntica forma a los documentos resultantes de las demás líneas de trabajo.

- Planificar las Pruebas.

  El plan de pruebas deberá ser diseñado según los productos que serán generados. El objetivo de este plan debe ser encontrar las posibles faltas que contengan los productos generados y coordinar la corrección de las mismas.

- Diseñar los Casos de Prueba.

  El documento de casos de prueba generado debe cumplir con las siguientes características:

  - Debe contener casos de prueba que tengan la mayor probabilidad de detectar el mayor número posible de errores dados los recursos disponibles, las limitaciones de tiempo, etc., dado que es imposible la realización de pruebas exhaustivas que verifiquen todas las posibles secuencias de ejecución del producto.
  - Los casos de prueba deben estar expresados de forma clara y concisa.
  - Los casos de prueba deben ser reproducibles.
  - La salida esperada (correcta) debe estar claramente definida, de manera que la generación del reporte de pruebas que surgirá del testeo sea útil.
  - Se crearán casos de prueba que cubran las combinaciones más relevantes de estados de entidades, de condiciones iniciales, actores, entradas y que generen escenarios de interés.

2.5.4.3 Entradas / salidas

A continuación se especifican las entradas necesarias y salidas previstas para la verificación de documentos de esta sección.

- Verificación de Documentos

  Entradas:

  - Documento a verificar.

  Salidas:

  - Reportes de verificación.

- Planificar las Pruebas

  Entradas:

  - Modelo de Casos de uso.
  - Modelo de Análisis (si corresponde).

- Modelo de Diseño.
- Modelo de Implementación.

Salidas:

- Plan de Pruebas.

- Diseñar Casos de Prueba

  Entradas:

  - Modelo de Casos de uso.

  Salidas:

  - Casos de Prueba.

2.5.4.4 Recursos

Todos los integrantes del grupo.

2.5.5 Línea de Trabajo de Testeo

2.5.5.1 Tareas

- Realizar las Pruebas de Integración.
- Realizar las Pruebas del Sistema.

2.5.5.2 Métodos y Criterios

- Realizar las Pruebas de Integración

  El objetivo de estas pruebas será verificar las interacciones entre los distintos subsistemas. Se desarrollaran casos de prueba que verifiquen el correcto funcionamiento de los componentes integrados, sobretodo en lo referente a las interfaces. Se tendrá mayor énfasis en la verificación de las partes de nivel de calidad 5.

- Realizar las Pruebas del Sistema

  Esta prueba se desarrollará en el ambiente de producción procurando simular una situación de operación lo más cercana a la realidad. Aquí se verificaran las funcionalidades principales y se pondrá especial atención en los requerimientos no funcionales y en la opinión del cliente.

2.5.5.3 Entradas / salidas

A continuación se especifican las entradas necesarias y salidas previstas para la verificación de documentos de esta sección.

- Realizar las Pruebas de Integración

  Entradas:

  - Plan de Pruebas.
  - Casos de Pruebas.
  - Componentes Integrados.

  Salidas:

  - Reporte de pruebas.

- Planificar las Pruebas del Sistema

  Entradas:

  - Plan de Pruebas.
  - Casos de Pruebas.
  - Ejecutable.

  Salidas:

  - Reporte de Pruebas.

2.5.5.4 Recursos

Todos los integrantes del grupo.

2.6 Reportes de Verificación y Validación

2.6.1 Reportes requeridos

- **Reporte de pruebas de integración**

  En este documento se registran las pruebas de los componentes integrados, en este se documentan los resultados obtenidos de la ejecución de los caso de prueba de integración y su comparación con los resultados esperados de las mismas.

- **Reporte de pruebas del sistema**

  Se documentan las pruebas realizadas sobre el ejecutable, se ejecutan los casos de prueba, se compara los resultados obtenidos con los esperados.

  Existen otros documentos, estos son; el Plan de pruebas y Casos de Prueba (y procedimientos).

2.6.2 Reportes opcionales

Por el momento no se tienen elementos que indiquen la necesidad de agregar otros reportes a los ya existentes, pero no se descarta su existencia en un futuro.

3. Casos y procedimientos de prueba

En este capítulo se detallarán los casos y procedimientos de prueba planificados para verificar el sistema, en función de los cubrimientos definidos. También se incluirá en esta sección los resultados obtenidos para las pruebas funcionales.

3.1 Versión del sistema:

Se utiliza la versión final del sistema.

3.2 Requerimientos Funcionales

En esta sección se mostrará cuales fueron los procedimientos de prueba planificados para validar y verificar los requerimientos del sistema.

3.2.1 Administración de paquetes.

Cubierto por los puntos 3.2.1 y 3.2.2.

3.2.2 Browser de paquetes en el servidor de descargas.

Cubierto por el punto 3.2.3.

3.2.3 Descarga de paquetes/aplicaciones.

Cubierto por el punto 3.2.4.

3.2.4 Obtención de paquetes en forma local.

Se copiarán los paquetes manualmente desde un sistema de archivos predefinido, a la carpeta que contiene los paquetes en el nodo objetivo.

3.2.5 Autenticación en el servidor de descargas.

Se probará dando de alta en el nodo soporte un paquete que se sepa pasa la autenticación y otro que no lo haga.

3.2.6 Verificación de la autenticidad del paquete.

Se probará descargando en el nodo objetivo un paquete que se sepa pasa la verificación y otro que no lo haga.

3.2.7 Deployment.

Cubierto por el punto 3.2.5.

3.2.8 Configuración de una aplicación.

Cubierto por el punto 3.2.5.

3.2.9 Undeployment.

Cubierto por el punto 3.2.6.

3.2.10 Upgrades.

Cubierto por el punto 3.2.7.

3.2.11 Seguimiento de problemas durante el deployment.

Se probará con el punto 3.2.5 verificando los reportes que retorna.

3.2.12 Configuración del sistema.

Cubierto por los puntos 3.2.8 y 3.2.9.

3.2.13 Manejo de eventos.

Se probará lanzando eventos y verificando que sean captados por el sistema.

3.2.14 Log del sistema.

Se probará creando errores y verificando que todos sean registrados en el log.

3.2.15 Extensibilidad del sistema.

Se probará agregando módulos al sistema y verificando que estos funcionen correctamente.

3.2.16 Instalación desde el nodo soporte.

Se probará ejecutando remotamente los puntos 3.2.5, 3.2.6 y 3.2.7.

3.3 Casos de uso

3.3.1 Alta de paquetes en el nodo soporte

El ASS selecciona la opción alta de paquetes, el sistema le pide que ingrese el camino al paquete y el nombre del mismo y lo da de alta en el nodo soporte.

3.3.1.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Alta exitosa	Flujo Normal	N/A	
Alta no exitosa por falta de datos	Flujo Normal	3.A	
Alta no exitosa por falla de verificación	Flujo Normal	5.A	
Alta exitosa con falta de aplicaciones dependientes	Flujo Normal	6.A	
Alta no exitosa por no poder registrar el paquete en el nodo soporte	Flujo Normal	6.A	
Alta abortada	Flujo Normal	N/A	

3.3.1.2 Planilla de Condiciones

Escenario – Condición	Tipo de Resultado Esperado	Casos
1. Alta exitosa	El sistema copia el paquete en el nodo soporte, lo registre en el mismo y despliegue un mensaje indicando el éxito de la operación.	1,3,4,5,6, 7,8
2. Alta no exitosa por falta de datos	El sistema despliega el mensaje correspondiente, comunicando al usuario que debe ingresar los datos requeridos.	
3. Alta no exitosa por falla de verificación	Falla la verificación de autenticidad realizada por el sistema. El sistema despliega el mensaje correspondiente y se cancela el caso de uso.	
4. Alta exitosa con falta de aplicaciones dependientes	El sistema despliega el mensaje correspondiente. El sistema copia el paquete en el nodo soporte, lo registre en el mismo y despliegue un mensaje indicando el éxito de la operación.	2
5. Alta no exitosa por no poder registrar el paquete en el nodo soporte	El sistema detecta que no pudo registrar el paquete en el nodo soporte, el sistema despliega el mensaje correspondiente y se cancela el caso de uso.	
6. Alta abortada	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso	

3.3.2 Baja de paquetes en el nodo soporte

El ASS selecciona los paquetes que quiere dar de baja del nodo soporte de una lista de paquetes que el sistema le despliega. El sistema da de baja el paquete seleccionado.

3.3.2.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Baja exitosa	Flujo Normal	N/A	
El usuario no confirma la operación	Flujo Normal	6.A	
No se puede borrar el paquete del nodo soporte	Flujo Normal	7.A	
Baja abortada	Flujo Normal	N/A	

3.3.2.2 Planilla de Condiciones

Escenario – Condición	Tipo de Resultado Esperado	Casos
7. Baja exitosa	El sistema borra los paquetes seleccionados del nodo soporte y los borra del registro.	
8. El usuario no confirma la operación	El sistema despliega el mensaje correspondiente, cancela el caso de uso.	
9. No se puede borrar algún paquete del nodo soporte	El sistema borra y quita del registro los paquetes seleccionados que pueda. Para los que no pueda despliega un mensaje indicando cuales fueron los paquetes que no se pudieron dar de baja.	
10. Baja abortada	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso	

3.3.3 Navegación entre paquetes

El sistema le muestra las aplicaciones que hay disponibles para la descarga, junto con una descripción de la funcionalidad de la aplicación y la información de la misma (autor, fecha de creación, tamaño, etc.). También se muestra para cada aplicación la lista de aplicaciones que necesita para ejecutarse correctamente.

3.3.3.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Navegación exitosa	Flujo Normal	N/A	
Navegación abortada	Flujo Normal	N/A	

3.3.3.2 Planilla de Condiciones

Escenario – Condición	Tipo de Resultado Esperado	Casos
11. Navegación exitosa	El sistema despliega las paquetes solicitados indicando sus datos y sus dependencias.	2,3,4,5,6, 7,8
12. Navegación abortada	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso	

3.3.4 Descarga de paquetes

El ASA se conecta al servidor de descargas utilizando la opción de descargas y el sistema le muestra las aplicaciones que puede descargar junto con una descripción de la funcionalidad de la aplicación. También se muestra para cada aplicación la lista de aplicaciones que necesita para ejecutarse correctamente. El usuario selecciona las aplicaciones que desee descargar y el sistema descarga los paquetes que contienen las aplicaciones al nodo objetivo Link All.

3.3.4.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Descarga exitosa	Flujo Normal	N/A	
Falla conexión al servidor de descargas	Flujo Normal	2.A	
Falla obtención de los descriptores	Flujo Normal	3.A	
Falla verificación de dependencias	Flujo Normal	6.A	
Falla descarga del paquete	Flujo Normal	7.A	
Falla autenticación del paquete	Flujo Normal	7.B	
Descarga abortada	Flujo Normal	N/A	

3.3.4.2 Planilla de Condiciones

Escenario – Condición	Tipo de Resultado Esperado	Casos
13. Descarga exitosa	EL paquete queda descargado en el nodo objetivo.	4,7,8
14. Falla conexión al servidor de descargas	El sistema despliega el mensaje correspondiente, cancela el caso de uso.	
15. Falla obtención de descriptores	El sistema despliega el mensaje correspondiente, cancela el caso de uso.	
16. Falla verificación de dependencias	El sistema despliega el mensaje correspondiente, cancela el caso de uso.	
17. Falla descarga del paquete	El sistema despliega el mensaje correspondiente, cancela el caso de uso.	4
18. Falla autenticación del paquete	El sistema despliega el mensaje correspondiente, elimina el paquete descargado, cancela el caso de uso.	6
19. Descarga abortada	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso.	

3.3.5 Deplyment

El ASA selecciona la opción “Deployment”, selecciona las aplicaciones a instalar y el sistema instala la aplicación retornándole al ASA un reporte del proceso.

3.3.5.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Deploy exitoso	Flujo Normal	N/A	
No hay paquetes para deployar o están todos instalados	Flujo Normal	2.A	
Faltan seleccionar aplicaciones dependientes	Flujo Normal	5.A	
Fallo alguna de las acciones de instalación	Flujo Normal	6.A	
Deploy abortado	Flujo Normal	N/A	

3.3.5.2 Planilla de Condiciones

Escenario – Condición	Tipo de Resultado Esperado	Casos
20. Deploy exitoso	El sistema despliega un mensaje indicando que la operación se realizo con éxito y muestra un reporte del proceso.	7
21. No hay paquetes para deployar o están todos instalados	El sistema despliega el mensaje correspondiente, cancela el caso de uso.	
22. Faltan seleccionar aplicaciones dependientes	El sistema se da cuenta que para alguna aplicación no se han seleccionado todas las aplicaciones de las cuales depende que aun no han sido deploradas. Despliega el mensaje correspondiente y da la opción de seleccionar nuevamente las aplicaciones a deployar	
23. Fallo alguna de las acciones de instalación	El sistema no pudo, para alguna aplicación, copiar los archivos en el lugar que correspondía u ocurrió algún error inesperado. Registra el error en el reporte del proceso, no instala esa aplicación ni las aplicaciones seleccionadas que dependían de ella, continua con la instalación de las otras aplicaciones seleccionadas no dependientes de ella. Al culminar despliega un mensaje indicando que alguna aplicación no pudo ser instalada y muestra el reporte donde se indica que aplicación no pudo ser instalada y cuales no fueron instaladas porque dependían de esta.	8
24. Deploy abortado	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso	

3.3.6 Undeployment

El ASA selecciona la opción "Undeployment", selecciona las aplicaciones a desinstalar y el sistema desinstala las aplicaciones retornándole al ASA un reporte del proceso. El ASA tiene la opción de desinstalar todas las aplicaciones que dependen de esta aplicación ya que no funcionarán correctamente luego del undeployment; lo mismo se hará extensivo para aquellas aplicaciones que dependen transitivamente.

3.3.6.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Undeployment exitoso	Flujo Normal	N/A	
Undeployment abortado	Flujo Normal	N/A	

3.3.6.2 Planilla de Condiciones

Escenario - Condición	Tipo de Resultado Esperado	Casos
25. Undeployment exitoso	El sistema desinstala todas las aplicaciones seleccionadas, elimina sus enganches y despliega un mensaje indicando que la operación termino con éxito.	
26. Undeployment abortado	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso	

3.3.7 Upgrade de aplicaciones

El ASA selecciona la opción upgrade, selecciona la aplicación a la que le va a hacer el upgrade, selecciona la ubicación del paquete que contiene el upgrade y el sistema instala la nueva versión retornándole al ASA un reporte del proceso.

3.3.7.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Upgrade exitoso	Flujo Normal	N/A	
Upgrade abortado	Flujo Normal	N/A	

3.3.7.2 Planilla de Condiciones

Escenario – Condición	Tipo de Resultado Esperado	Casos
27. Upgrade exitoso	La aplicación seleccionada para realizarle el upgrade queda actualizada.	
28. Upgrade abortado	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso.	

3.3.8 Configuración del nodo objetivo

El ASA selecciona la opción de "Configurar", cambia los datos que desee y acepta los cambios, el sistema aplica el cambio en la configuración. Lo que se puede configurar es:

- dirección de acceso al servidor de descargas
- dirección de almacenamiento local de paquetes
- dirección de almacenamiento de especificaciones de deployment
- dirección del almacenamiento temporal
- usuario, password y string de conexión a la base de datos que contiene la información sobre los recursos y los links entre ellos.
- mantenimiento de enlaces entre recursos

3.3.8.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Configuración exitosa	Flujo Normal	N/A	
Configuración abortada	Flujo Normal	N/A	

3.3.8.2 Planilla de Condiciones

Escenario – Condición	Tipo de Resultado Esperado	Casos
29. Configuración exitosa	El sistema aplica los cambios en la configuración realizados por el usuario.	
30. Configuración abortada	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso.	

3.3.9 Configuración del nodo soporte

El ASS selecciona la opción de "Configurar dirección de almacenamiento", ingresa la nueva dirección para la carpeta de almacenamiento de paquetes y acepta los cambios, el sistema toma esta dirección como la dirección por defecto para el almacenamiento de los paquetes.

3.3.9.1 Escenarios

Nombre de escenario	Flujo inicial	Flujo alternativo	Flujo alternativo
Configuración en nodo soporte exitosa	Flujo Normal	N/A	
Configuración en nodo soporte abortada	Flujo Normal	N/A	

3.3.9.2 Planilla de Condiciones

Escenario - Condición	Tipo de Resultado Esperado	Casos
31. Configuración en nodo soporte exitosa	El sistema aplica los cambios en la configuración del nodo soporte realizados por el usuario.	
32. Configuración en nodo soporte abortada	El sistema permite la opción cancelar en cualquier momento del caso de uso, se cancela el caso de uso.	

3.4 Casos de prueba

Prueba	Datos de prueba			Resultado Esperado
1	**Caso de uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete.**
	Alta de paquetes en el nodo soporte	Alta exitosa	Paquete / Valido	
2	**Caso de uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete pero indicando que fallaron las dependencias. Muestra los paquetes existentes en el nodo.**
	Alta de paquetes en el nodo soporte	Alta exitosa	Dependencias / No Verificables	
	Navegación entre paquetes	Navegación exitosa	Paquetes/ Válidos	
3	**Caso de Uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete. Muestra los paquetes existentes en el nodo.**
	Alta de paquetes en el nodo soporte	Alta exitosa	Paquete / Válido	
	Navegación entre paquetes	Navegación exitosa	Paquetes/ Válidos	
4	**Caso de Uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete. Muestra los paquetes existentes en el nodo. Descarga al nodo objetivo de los paquetes seleccionados en la navegación.**
	Alta de paquetes en el nodo soporte	Alta exitosa	Paquete / Válido	
	Navegación entre paquetes	Navegación exitosa	Paquetes / Válidos	
	Descarga de paquetes	Descarga exitosa	Paquetes / Válidos Dependencias / Verificables Firmas / Autenticables	

5	**Caso de Uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete. Muestra los paquetes existentes en el nodo. Mensaje de error indicando cuales fueron las dependencias que fallaron. Descarga al nodo objetivo de los paquetes (nuevamente seleccionados) seleccionados en la navegación.**
	Alta de paquetes en el nodo soporte	Alta exitosa	Paquete / Válido	
	Navegación entre paquetes	Navegación exitosa	Paquetes/ Válidos	
	Descarga de paquetes	Falla descarga del paquete	Paquetes/ Válidos Dependencias / No Verificables Firmas / Autenticables	
	Descarga de paquetes	Descarga exitosa	Paquetes / Válidos Dependencias / Verificables Firmas / Autenticables	
6	**Caso de Uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete. Muestra los paquetes existentes en el nodo. Mensaje de error indicando que fallo la verificación de firmas. Descarga al nodo objetivo de los paquetes (nuevamente seleccionados) seleccionados en la navegación**
	Alta de paquetes en el nodo soporte	Alta exitosa	Paquete / Válido	
	Navegación entre paquetes	Navegación exitosa	Paquetes / Válidos	
	Descarga de paquetes	Falla autenticación del paquete	Paquetes / Válidos Dependencias / Verificables Firmas / No Autenticables	
	Descarga de paquetes	Descarga exitosa	Paquetes / Válidos Dependencias / Verificables Firmas / Autenticables	
7	**Caso de Uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete. Muestra los paquetes existentes en el nodo. Mensaje de error indicando que fallo la verificación de firmas. Descarga al nodo objetivo de los paquetes (nuevamente seleccionados) seleccionados en la navegación**
	Alta de paquetes en el nodo soporte	Alta exitosa	Paquete / Válido	
	Navegación entre paquetes	Navegación exitosa	Paquetes / Válidos	
	Descarga de paquetes	Descarga exitosa	Paquetes / Válidos Dependencias / Verificables Firmas / Autenticables	
	Deployment	Deploy exitoso	Paquete / Válido Acciones / Válidas	

8	**Caso de Uso**	**Escenario / Condición**	**Entidades / Estados**	**Mensaje indicando el alta exitosa del paquete. Muestra los paquetes existentes en el nodo. Descarga al nodo objetivo de los paquetes seleccionados en la navegación. Mensaje indicando el error por el cual no se pudo deployar la aplicación.**
	Alta de paquetes en el nodo soporte	Alta exitosa	Paquete / Válido	
	Navegación entre paquetes	Navegación exitosa	Paquetes/ Válidos	
	Descarga de paquetes	Descarga exitosa	Paquetes/ Válidos Dependencias / Verificables Firmas / Autenticables	
	Deployment	Deploy no exitoso por falla en la acción	Paquete / Válido Acciones / Invalidas	

3.5 Resultados

En esta sección detallaremos brevemente los resultados obtenidos durante el testing del producto final a nivel funcional. Los requerimientos de calidad exigían un índice de fallas (cantidad de fallas / casos de prueba) menor o igual a 0,1.

Del testeo funcional realizado se obtuvo un indicador de 0,086, este índice satisface el requerido para el nivel de calidad 5. En este caso las fallas fueron variadas pero en lo que respecta a casos de uso y requerimiento el resultado obtenido fue menor al expuesto anteriormente. Las fallas radicaron principalmente en problemas de configuración del sistema e interfaz de usuario, principalmente detalles en la misma.

La metodología seguida al testear el sistema fue la siguiente, para los casos de prueba propuestos en este documento se crearon diferentes juegos de datos para realizar el testing. Las funcionalidades fueron todas testeadas a través de la UI provista por el sistema. No fueron creadas pruebas automáticas ya que el punto de entrada es la UI y el grupo considero que esta era la mejor forma de realizar el testing funcional.

El grupo considera que las fallas encontradas son mínimas y sin mayor relevancia, las fallas encontradas fueron corregidas en su totalidad. El test realizado cumple con los cubrimientos propuestos, si se tomaran otros criterios más exigentes, cabe la posibilidad de detectar fallas que no fueron detectadas en el testing realizado por el grupo. Cabe aclarar que los cubrimiento satisfacen lo solicitado para el nivel de calidad exigido para el proyecto.

4. Pruebas de componentes

En este capítulo se definirán pruebas para los principales componentes del sistema. Se mostrarán los componentes con sus funcionalidades y pruebas para las mismas. Al final de este capítulo serán expuestos los resultados obtenidos durante las pruebas de componentes.

4.1 Deployable Management Support

4.1.1 Carga de paquetes

Esta funcionalidad recorre el repositorio del nodo soporte en busca de paquetes de deployment, e intenta registrar los archivos de este tipo.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Utilizar la funcionalidad con un repositorio vacío	No ocurren errores, se obtiene un reporte con solamente una nota indicando la finalización de la carga.
2	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados)	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los existentes en el repositorio.
3	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados) y archivos de otros tipos	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los paquetes de deployment validos existentes en el repositorio.
4	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados) y paquetes inválidos (que no puedan ser registrados, sin AD y sin firma)	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los paquetes de deployment validos existentes en el repositorio. También en el reporte esta la información de que paquetes no fueron registrados y deben coincidir con los paquetes invalidos existentes en el repositorio.
5	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados), archivos de otros tipos y paquetes de deployment inválidos (que no puedan ser registrados)	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los paquetes de deployment validos existentes en el repositorio. También en el reporte esta la información de que paquetes no fueron registrados y deben coincidir con los paquetes inválidos existentes en el repositorio.
6	Utilizar la funcionalidad con un repositorio con paquetes de deployment inválidos (que no puedan ser registrados)	No ocurren errores, se obtiene un reporte conteniendo la información de que paquetes no fueron registrados y deben coincidir con los paquetes inválidos existentes en el repositorio.

7	Utilizar la funcionalidad con un repositorio con paquetes de deployment inválidos y archivos de otros tipos	No ocurren errores, se obtiene un reporte conteniendo la información de que paquetes no fueron registrados y deben coincidir con los paquetes inválidos existentes en el repositorio.
8	Utilizar la funcionalidad con un repositorio con archivos de otros tipos	No ocurren errores, se obtiene un reporte con solamente una nota indicando la finalización de la carga.

4.1.2 Registro de paquetes

Esta funcionalidad recibe el nombre de un paquete de deployment y lo registra en el nodo soporte.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasarle como parámetro el nombre de un paquete inexistente.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.
2	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD, pase la verificación de firmas –requerido en el sistema- y el chequeo de dependencias)	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete.
3	Pasarle como parámetro el nombre de un paquete existente y que exista el archivo AD, pase la verificación de firmas –requerido en el sistema- y falle el chequeo de dependencias.	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete pero con errores en el chequeo de dependencias.
4	Pasarle como parámetro el nombre de un paquete existente y que no exista el archivo AD.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.
5	Pasarle como parámetro el nombre de un paquete existente y que exista el archivo AD, falle la verificación de firmas –requerido en el sistema- y pase el chequeo de dependencias.	No ocurren errores, se obtiene un reporte indicando el error en la verificación de la firma y diciendo que no se registro el paquete.
6	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD y el chequeo de dependencias) El chequeo de firmas no es requerido.	No ocurren errores, se obtiene un reporte indicando el registro correcto del.
7	Pasarle como parámetro el nombre de un paquete existente y que exista el archivo AD y falle el chequeo de dependencias. El chequeo de firmas no es requerido.	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete pero con warnings en el chequeo de dependencias.

8	Pasarle como parámetro el nombre de un paquete existente y que no exista el archivo AD. El chequeo de firmas no es requerido.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.
9	Pasarle como parámetro el nombre de un paquete existente y que no exista el archivo de firma. El chequeo de firmas es requerido.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.
10	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD, pase la verificación de firmas –requerido en el sistema- y el chequeo de dependencias). Que la el paquete dependa de otro.	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete.
11	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD, pase la verificación de firmas – no requerido en el sistema- y el chequeo de dependencias). Que la el paquete dependa de otro.	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete.

4.1.3 Eliminación de paquetes

Esta funcionalidad recibe el identificador de un paquete y lo elimina del nodo soporte.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasarle como parámetro el identificador de un paquete registrado en el nodo, que exista el archivo físico en el repositorio y que nadie dependa de él.	No ocurren errores, el paquete no esta mas registrado en el nodo, el archivo físico no existe mas en el repositorio y se obtiene un reporte indicando que se removió el paquete del nodo.
2	Pasarle como parámetro el identificador de un paquete que no este registrado en el nodo.	No ocurren errores, se obtiene un reporte indicando que el paquete no existe en el nodo.
3	Pasarle como parámetro el identificador de un paquete registrado en el nodo pero que no exista el archivo físico en el repositorio.	No ocurren errores, el paquete no esta mas registrado en el nodo y se obtiene un reporte indicando que se removió el paquete del nodo.
4	Pasarle como parámetro el identificador de un paquete registrado en el nodo, que exista el archivo físico y alguien dependa de él.	No ocurren errores, el paquete no esta mas registrado en el nodo, el archivo físico no existe mas en el repositorio y se obtiene un reporte indicando que se removió el paquete del nodo.

4.1.4 Consulta de los paquetes disponibles en el nodo

Esta funcionalidad retorna los descriptores de los paquetes registrados en el nodo soporte.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Utilizar la funcionalidad cuando no exista ningún paquete registrado en el nodo.	No ocurren errores, se obtiene una estructura vacia.
2	Utilizar la funcionalidad cuando existan paquetes registrados en el nodo.	No ocurren errores, se obtiene una estructura con los descriptores de paquetes registrados

4.2 Deployable Management Target

4.2.1 Carga de paquetes

Esta funcionalidad recorre el repositorio del nodo objetivo en busca de paquetes de deployment, e intenta registrar los archivos de este tipo.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Utilizar la funcionalidad con un repositorio vacío	No ocurren errores, se obtiene un reporte con solamente una nota indicando la finalización de la carga.
2	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados)	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los existentes en el repositorio.
3	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados) y archivos de otros tipos	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los paquetes de deployment validos existentes en el repositorio.
4	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados) y paquetes inválidos (que no puedan ser registrados)	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los paquetes de deployment validos existentes en el repositorio. También en el reporte esta la información de que paquetes no fueron registrados y deben coincidir con los paquetes invalidos existentes en el repositorio.

5	Utilizar la funcionalidad con un repositorio con paquetes de deployment validos (que puedan ser registrados), archivos de otros tipos y paquetes de deployment inválidos (que no puedan ser registrados)	No ocurren errores, se obtiene un reporte conteniendo notas que indican que paquetes se registraron, y estos deben coincidir con los paquetes de deployment validos existentes en el repositorio. También en el reporte esta la información de que paquetes no fueron registrados y deben coincidir con los paquetes inválidos existentes en el repositorio.
6	Utilizar la funcionalidad con un repositorio con paquetes de deployment inválidos (que no puedan ser registrados)	No ocurren errores, se obtiene un reporte conteniendo la información de que paquetes no fueron registrados y deben coincidir con los paquetes inválidos existentes en el repositorio.
7	Utilizar la funcionalidad con un repositorio con paquetes de deployment inválidos y archivos de otros tipos	No ocurren errores, se obtiene un reporte conteniendo la información de que paquetes no fueron registrados y deben coincidir con los paquetes inválidos existentes en el repositorio.
8	Utilizar la funcionalidad con un repositorio con archivos de otros tipos	No ocurren errores, se obtiene un reporte con solamente una nota indicando la finalización de la carga.

4.2.2 Registro de paquetes

Esta funcionalidad recibe el nombre de un paquete de deployment y lo registra en el nodo objetivo.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasarle como parámetro el nombre de un paquete inexistente.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.
2	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD, pase la verificación de firmas –requerido en el sistema- y el chequeo de dependencias)	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete y que el mismo no fue registrado.
3	Pasarle como parámetro el nombre de un paquete existente y que exista el archivo AD, pase la verificación de firmas –requerido en el sistema- y falle el chequeo de dependencias.	No ocurren errores, se obtiene un reporte con errores indicando que no se registro el paquete e indica cuales son las aplicaciones de las que depende y que no se encuentran registradas en el nodo objetivo.
4	Pasarle como parámetro el nombre de un paquete existente y que no exista el archivo AD.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.

5	Pasarle como parámetro el nombre de un paquete existente y que exista el archivo AD, falle la verificación de firmas –requerido en el sistema- y pase el chequeo de dependencias.	No ocurren errores, se obtiene un reporte indicando el error en la verificación de la firma y diciendo que no se registro el paquete.
6	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD y el chequeo de dependencias) El chequeo de firmas no es requerido.	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete y que el mismo no fue registrado.
7	Pasarle como parámetro el nombre de un paquete existente y que exista el archivo AD y falle el chequeo de dependencias. El chequeo de firmas no es requerido.	No ocurren errores, se obtiene un reporte con errores indicando que no se registro el paquete e indica cuales son las aplicaciones de las que depende y que no se encuentran registradas en el nodo objetivo.
8	Pasarle como parámetro el nombre de un paquete existente y que no exista el archivo AD. El chequeo de firmas no es requerido.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.
9	Pasarle como parámetro el nombre de un paquete existente y que no exista el archivo de firma. El chequeo de firmas es requerido.	No ocurren errores, se obtiene un reporte indicando el error al registrar el paquete y que el mismo no fue registrado.
10	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD, pase la verificación de firmas –requerido en el sistema- y el chequeo de dependencias). Que la el paquete dependa de otro.	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete.
11	Pasarle como parámetro el nombre de un paquete existente y que pueda ser registrado correctamente. (exista el archivo AD, pase la verificación de firmas – no requerido en el sistema- y el chequeo de dependencias). Que el paquete dependa.	No ocurren errores, se obtiene un reporte indicando el registro correcto del paquete.

4.2.3 Eliminación de paquetes

Esta funcionalidad recibe el identificador de un paquete y lo elimina del nodo objetivo.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasarle como parámetro el identificador de un paquete registrado en el nodo, que exista el archivo físico en el repositorio y que nadie dependa de él.	No ocurren errores, el paquete no esta mas registrado en el nodo, el archivo físico no existe mas en el repositorio y se obtiene un reporte indicando que se removió el paquete del nodo.

2	Pasarle como parámetro el identificador de un paquete que no este registrado en el nodo.	No ocurren errores, se obtiene un reporte indicando que el paquete no existe en el nodo.
3	Pasarle como parámetro el identificador de un paquete registrado en el nodo pero que no exista el archivo físico en el repositorio.	No ocurren errores, el paquete no esta mas registrado en el nodo y se obtiene un reporte indicando que se removió el paquete del nodo.
4	Pasarle como parámetro el identificador de un paquete registrado en el nodo, que exista el archivo físico y alguien dependa de él.	No ocurren errores, el paquete no es eliminado del repositorio y se obtiene un reporte indicando que no se removió el paquete y errores indicando quienes dependen de él, por lo que no se puede eliminar.

4.2.4 Consulta de los paquetes disponibles en el nodo

Esta funcionalidad retorna los descriptores de los paquetes registrados en el nodo objetivo.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Utilizar la funcionalidad cuando no exista ningún paquete registrado en el nodo.	No ocurren errores, se obtiene una estructura vacia.
2	Utilizar la funcionalidad cuando existan paquetes registrados en el nodo.	No ocurren errores, se obtiene una estructura con los descriptores de paquetes registrados

4.2.5 Consulta de los paquetes disponibles en otros nodos

Esta funcionalidad retorna los descriptores de los paquetes registrados en el nodo cuya string de conexión es pasado como parámetro.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Utilizar la funcionalidad con un string de conexión valido cuando no exista ningún paquete registrado en el nodo que se esta consultando.	No ocurren errores, se obtiene una estructura vacia.
2	Utilizar la funcionalidad con un string de conexión valido cuando existan paquetes registrados en el nodo que se esta consultando.	No ocurren errores, se obtiene una estructura con los descriptores de paquetes registrados
3	Utilizar la funcionalidad con un string de conexión invalido.	No ocurren errores, se obtiene una estructura vacia.

4.2.6 Obtener acceso a un paquete dado

Esta funcionalidad retorna un objeto que provee el acceso a un paquete dado, recibe como parámetro el identificador de un paquete.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasar como parámetro el identificador de un paquete registrado en el nodo.	No ocurren errores, se obtiene el objeto que accede al paquete físico.
2	Pasar como parámetro el identificador de un paquete que no este registrado en el nodo.	No ocurren errores, se retorna null.

4.2.7 Descarga de paquetes desde otro nodo

Esta funcionalidad recibe una lista de identificadores de paquetes existentes en el nodo desde el cual se va a descargar y los descarga al repositorio del nodo objetivo.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasar como parámetro una lista vacía	No ocurren errores, se obtiene un reporte indicando la descarga exitosa.
2	Pasar como parámetro una lista con identificadores de paquetes existentes en el nodo de descarga. Sus dependencias se cumplen. Chequeo de firmas no requerido.	No ocurren errores, se obtiene un reporte indicando que los paquetes fueron descargados en forma exitosa. Los paquetes quedan registrados en el nodo objetivo.
3	Pasar como parámetro una lista con identificadores de paquetes inexistentes en el nodo de descarga.	No ocurren errores, se obtiene un reporte indicando que los paquetes no fueron descargados y cuál fue la razón. Los paquetes no quedan registrados en el nodo objetivo.
4	Pasar como parámetro una lista con identificadores de paquetes existentes en el nodo de descarga. Uno/s de los paquetes a descargar depende de un paquete que no esta en el nodo objetivo y tampoco esta para en la lista de paquetes a descargar. Chequeo de firmas no requerido	No ocurren errores, se obtiene un reporte indicando que paquetes no fueron descargados y que la causa fue el chequeo de dependencias . Los paquetes no quedan registrados en el nodo objetivo.
5	Pasar como parámetro una lista con identificadores de paquetes existentes e inexistentes en el nodo de descarga. Chequeo de firmas no requerido. Para los existentes, sus dependencias se cumplen.	No ocurren errores, se obtiene un reporte indicando que paquetes no fueron descargados y cuál fue la razón. Los paquetes existentes quedan registrados en el nodo objetivo. Los paquetes insistentes no quedan registrados en el nodo objetivo.

6	Pasar como parámetro una lista con identificadores de paquetes existentes en el nodo de descarga. Uno/s de los paquetes a descargar depende de un paquete que no esta en el nodo objetivo y tampoco esta para en la lista de paquetes a descargar. Chequeo de firmas requerido	No ocurren errores, se obtiene un reporte indicando que paquetes no fueron descargados y que la causa fue el chequeo de dependencias . Los paquetes no quedan registrados en el nodo objetivo.
7	Pasar como parámetro una lista con identificadores de paquetes existentes e inexistentes en el nodo de descarga. Chequeo de firmas requerido. Para los existentes, sus dependencias se cumplen.	No ocurren errores, se obtiene un reporte indicando que paquetes no fueron descargados y cuál fue la razón. Los paquetes existentes quedan registrados en el nodo objetivo. Los paquetes inexistentes no quedan registrados en el nodo objetivo.
8	Pasar como parámetro una lista con identificadores de paquetes existentes en el nodo de descarga. Sus dependencias se cumplen. Chequeo de firmas requerido.	No ocurren errores, se obtiene un reporte indicando que los paquetes fueron descargados en forma exitosa. Los paquetes quedan registrados en el nodo objetivo.
9	Pasar como parámetro una lista con identificadores de paquetes existentes en el nodo de descarga. Sus dependencias se cumplen. Alguno/s de los paquetes no verifican las firmas. Chequeo de firmas requerido.	No ocurren errores, se obtiene un reporte indicando que los paquetes fueron descargados en forma exitosa y indicando que paquetes no fueron registrados por falla en la verificación de la firma. Los paquetes que fallan el chequeo de firmas no quedan registrados en el nodo objetivo, los otros si.

4.3 Resource Management

4.3.1 Carga de recursos persistidos

Esta funcionalidad carga los recursos persistidos.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	No existe el archivo de persistencia.	No ocurren errores, se obtiene un reporte indicando el error en la carga.
2	El archivo de persistencia no es valido	No ocurren errores, se obtiene un reporte indicando el error en la carga.
3	El archivo de persistencia es valido	Los recursos existentes quedan registrados en el resource mangement y se obtiene un reporte exitoso de la operación.

4.3.2 Registrar todos los recursos de una aplicación

Esta funcionalidad carga los recursos especificados en el archivo RS de una aplicación dada.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	No existe el archivo RS.	No ocurren errores, se obtiene un reporte indicando el error.
2	El archivo RS no es valido	No ocurren errores, se obtiene un reporte indicando el error.
3	El archivo RS es valido	Los recursos provistos y requeridos por la aplicación quedan registrados en el resource mangement y se obtiene un reporte exitoso de la operación.

4.3.3 Remover los recursos de una aplicación

Esta funcionalidad elimina los recursos provistos y requeridos, registrados por una aplicación dada.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Se pasa como parámetro el identificador de una aplicación que no tiene recursos registrados.	No ocurren errores, se obtiene un reporte indicando que se eliminaron los recursos de la aplicación.
2	Se pasa como parámetro el identificador de una aplicación que tiene recursos registrados.	No ocurren errores, se obtiene un reporte indicando que se eliminaron los recursos de la aplicación.

4.3.4 Obtener las especificaciones de todos los recursos registrados

Esta funcionalidad retorna las especificaciones de los recursos registrados en resource management.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Utilizar la funcionalidad cuando no exista ningún recurso registrado en el nodo.	No ocurren errores, se obtiene una estructura vacia.
2	Utilizar la funcionalidad cuando existan recursos registrados en el nodo.	No ocurren errores, se obtiene una estructura con las especificaciones de los recursos registrados

4.3.5 Obtener el acceso a los recursos requeridos por una aplicación

Esta funcionalidad retorna un objeto que provee el acceso a los recursos requeridos por una aplicación dada, recibe como parámetro el identificador de la aplicación que quiere enlazarse con recursos provistos por otra aplicación.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasar como parámetro el identificador de una aplicación que no tenga recursos requeridos	No ocurren errores, se obtiene un objeto que no provee acceso a ningún recurso.
2	Pasar como parámetro el identificador de una aplicación que tenga recursos requeridos.	No ocurren errores, se obtiene un objeto que provee acceso a los recursos requeridos por otra aplicación.
3	Pasar como parámetro el identificador de una aplicación que no este registrada	No ocurren errores, se obtiene un objeto que no provee acceso a ningún recurso.

4.4 Deployment

4.4.1 Deploy

Esta funcionalidad recibe una colección con los identificadores de las aplicaciones a deployar.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasarle como parámetro una colección de vacia.	No ocurren errores, se obtiene un reporte indicando resultado exitoso.
2	Pasarle como parámetro un colección de identificadores de aplicaciones que satisfagan las siguientes condiciones: el paquete conteniendo la aplicación existe en el nodo objetivo, el paquete esta correcto y las aplicaciones de las cuales depende la aplicación a deployar estan deployadas.	No ocurren errores, se obtiene un reporte indicando resultado exitoso de la operación y un detalle de las acciones de deployment ejecutadas. Las aplicaciones quedan deployadas.
3	Pasarle como parámetro un colección de identificadores de aplicaciones que satisfagan las siguientes condiciones: el paquete conteniendo la aplicación existe en el nodo objetivo, el paquete esta correcto y las aplicaciones de las cuales depende la aplicación a deployar estan deployadas o en la colección.	No ocurren errores, se obtiene un reporte indicando resultado exitoso de la operación y un detalle de las acciones de deployment ejecutadas. Las aplicaciones quedan deployadas.

4	Pasarle como parámetro un colección de identificadores de aplicaciones que satisfagan las siguientes condiciones: el paquete conteniendo la aplicación existe en el nodo objetivo, el paquete esta correcto y las aplicaciones de las cuales depende la aplicación a deployar estan en la colección.	No ocurren errores, se obtiene un reporte indicando resultado exitoso de la operación y un detalle de las acciones de deployment ejecutadas. Las aplicaciones quedan deployadas.
5	Pasarle como parámetro un colección de identificadores de aplicaciones que satisfagan las siguientes condiciones: el paquete conteniendo la aplicación existe en el nodo objetivo, el paquete esta correcto y las aplicaciones de las cuales depende la aplicación a deployar no están deployadas y no están en la colección.	No ocurren errores, se obtiene un reporte indicando los errores de dependencias e indicando que la operación no termino exitosamente. Las aplicaciones no quedan deployadas.
6	Pasarle como parámetro un colección de identificadores de aplicaciones de las cuales algunas no satisfacen sus dependencias.	No ocurren errores, se obtiene un reporte indicando los errores de dependencias e indicando que la operación no termino exitosamente. Las aplicaciones no quedan deployadas.
7	Pasarle como parámetro un colección de identificadores de aplicaciones de las cuales algunas no tienen un paquete asociado en el nodo objetivo.	No ocurren errores, se obtiene un reporte indicando los errores e indicando que la operación no termino exitosamente. Las aplicaciones no quedan deployadas.
8	Pasarle como parámetro un colección de identificadores de aplicaciones de las cuales algunas tienen un paquete asociado al cual le falta el DS .	No ocurren errores, se obtiene un reporte indicando los errores e indicando que la operación no termino exitosamente. Las aplicaciones no quedan deployadas.

4.4.2 Undeploy

Esta funcionalidad recibe una colección de identificadores de aplicaciones deployadas y les realiza el undeploy.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasarle como parámetro una colección de vacia.	No ocurren errores, se obtiene un reporte indicando resultado exitoso.
2	Pasarle como parámetro una colección de identificadores de aplicaciones que estén deployadas y no exista ninguna aplicación que dependa de ellas.	No ocurren errores, se obtiene un reporte indicando resultado exitoso de la operación y un detalle de las acciones de undeployment ejecutadas. Las aplicaciones quedan undeployadas.

3	Pasarle como parámetro una colección de identificadores de aplicaciones que estén deployadas y hay aplicaciones que dependen de ellas pero las mismas tambien estan ne la colección de aplicaciones a undeployar.	No ocurren errores, se obtiene un reporte indicando resultado exitoso de la operación y un detalle de las acciones de undeployment ejecutadas. Las aplicaciones quedan undeployadas.
4	Pasarle como parámetro una colección de identificadores de aplicaciones de las cuales algunas no esten deployadas.	No ocurren errores, se obtiene un reporte indicando los errores e indicando que la operación no termino exitosamente. Las aplicaciones no quedan undeployadas.
5	Pasarle como parámetro una colección de identificadores de aplicaciones de las cuales algunas son necesarias por otras que no seran undeployadas.	No ocurren errores, se obtiene un reporte indicando los errores e indicando que la operación no termino exitosamente. Las aplicaciones no quedan undeployadas.

4.4.3 Obtener las aplicaciones deploradas

Esta funcionalidad retorna los descriptores de las aplicaciones deployadas.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Utilizar la funcionalidad cuando no exista ninguna aplicación deployada.	No ocurren errores, se obtiene una estructura vacia.
2	Utilizar la funcionalidad cuando existan aplicaciones deployadas.	No ocurren errores, se obtiene una estructura con los descriptores de deployment pertenecientes a las aplicaciones deployadas.

4.4.4 Obtener los descriptores de las aplicaciones deploradas

Esta funcionalidad retorna el descriptor de la aplicación deployada cuyo identificador es pasado como parametro.

A continuación se listan las pruebas a realizar sobre esta funcionalidad:

Prueba		Resultado esperado
1	Pasar como parámetro el identificador de una aplicación que no este deployada	No ocurren errores, se obtiene null
2	Pasar como parámetro el identificador de una aplicación que este deployada	No ocurren errores, se obtiene el descriptor de la aplicación deplorada cuyo identificador fue pasado como parametro.

4.5 Resultados obtenidos

En esta sección detallaremos brevemente los resultados obtenidos durante el testing del producto final a nivel de componentes. Dado el nivel de calidad exigido para el proyecto, el indice de fallas (cantidad de fallas / casos de prueba) debia ser menor o igual a 0,05.

Para testear los componentes se crearon tests JUnit basados en los casos propuestos en este documento. Los resultados obtenidos en el testing arrojaron un índice de 0,044 para el total de casos de prueba dados.

Este índice satisface el requerido para el nivel de calidad 5. Las fallas se dieron principalmente en el control de errores dentro de los componentes. Los errores fueron corregidos y los tests ejecutados nuevamente. Esta nueva corrida no arrojo errores.

Discriminando los resultados obtenidos, tenemos que los componentes que arrojaron errores fueron el ResourceManager y el DeployableManagementeTarget. En lo que concierne al primero de los componentes, los errores se dieron en las funcionalidades de registro y eliminación de recursos. Los errores ocurrieron en los reportes retornados, en el caso de que fallara el registro de recursos, la falla no era indicada correctamente en el reporte retornado. Con respecto a la eliminación de recursos, el error se daba al pasarle como parámetro el identificador de una aplicación sin recursos registrados, el reporte que se retornaba contenía elementos null.

La funcionalidad que fallo en el componente DeployableManagementeTarget fue la descarga. Las fallas se debieron a la falta de control de errores en la funcionalidad. Los errores se producían cuando se pasaban como parámetro identificadores de paquetes inexistentes en el nodo de descarga, no se controlaba si los paquetes efectivamente existían.

5. Conclusiones generales

Del proceso de verificación y validación se puede decir que a pesar de que los integrantes del grupo no tenían experiencia en el tema, las actividades fueron realizadas en forma satisfactoria. Pese a esto, se considera que hubo actividades que pudieron haberse hecho de mejor manera.

Desde el punto de vista del testing, de acuerdo al Plan de Proyecto [1], las actividades de testing debían realizarse en los meses de diciembre, febrero y abril; con respecto a estas fechas existió un desfasaje de aproximadamente un mes. Además de esto, la forma en que se realizó la actividad no fue lo suficientemente formal como se pretendía y había planificado. Los test aplicados no fueron realizados con ninguna herramienta de testing formal, excepto el testeo de la versión final del sistema que fue realizada con JUnit. La causa del desfasaje en la verificación para las primeras dos instancias se debió a que el sistema estaba en un estado inestable en esos momentos por lo que se considero que realizar pruebas mientras estuviera inestable no aportaría nada. En la última instancia el retraso se debió a que el grupo prefirió tener terminada la última versión del sistema para realizar el testing del mismo.

Otra diferencia con la planificación fue que se iba testeando a medida que se iban programando las diferentes funcionalidades, es decir, cuando se tenía pronta una funcionalidad, ésta se testeaba y no se esperaba al momento planificado. Por esta razón se piensa que la cantidad de fallas detectadas en las distintas instancias de verificación son menores a las que se hubieran detectado si se hubiera realizado el testing de acuerdo a la planificación. Esto sucedió a causa, principalmente, del tiempo con el que se contaba, y a que los integrantes del grupo no estaban familiarizados con esta forma de trabajo y metodología.

A continuación se presentarán dos gráficos conteniendo los resultados (cantidad de fallas / casos de prueba) por instancia de verificación.

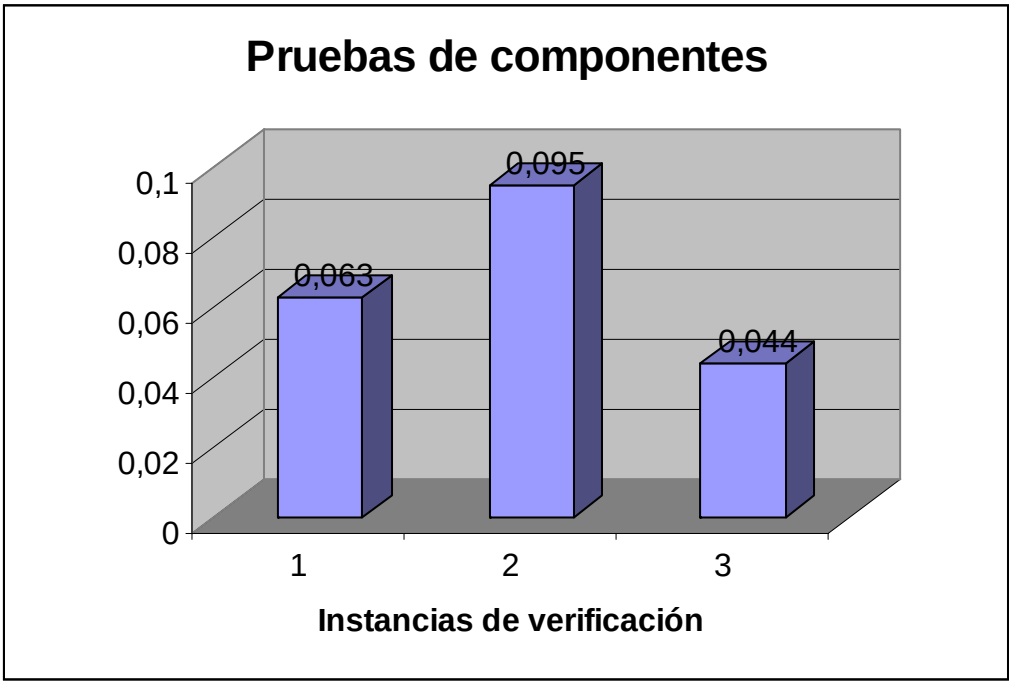


Figura 5-1 Resultados de las pruebas de componentes

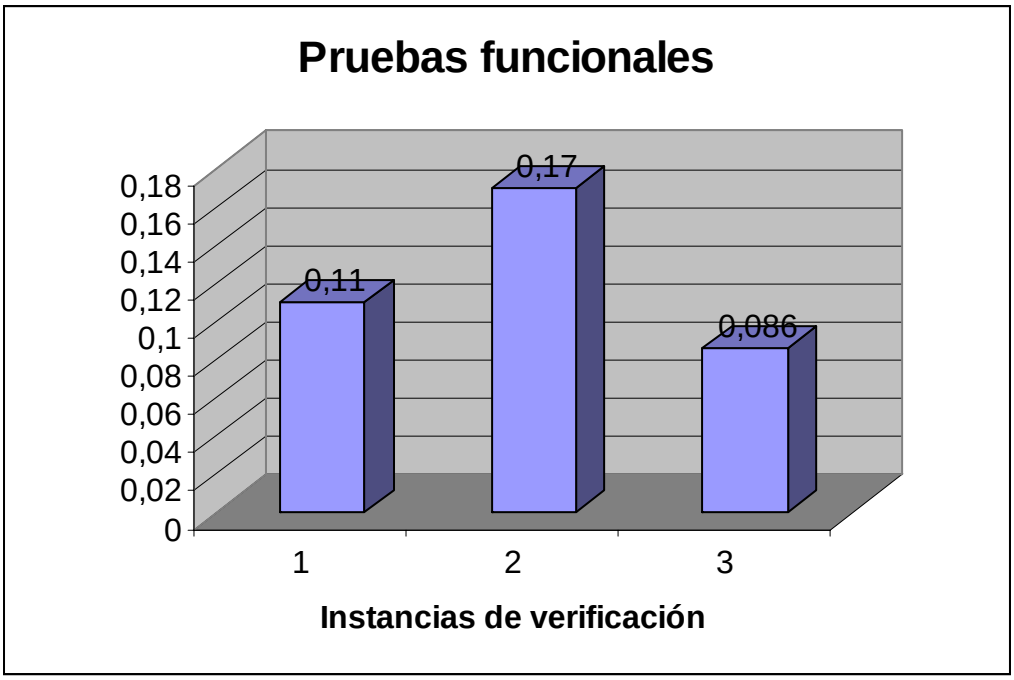


Figura 5-2 Resultados de las pruebas funcionales

Con respecto a los resultados podemos decir, tanto para las pruebas funcionales como para las pruebas de componentes, que los valores obtenidos para la versión final sistema están dentro de los valores esperados para los niveles de calidad definidos. Pese a que en las primeras dos instancias de verificación no se contó con tests formales, y de que la actividad no fue realizada de acuerdo a lo planificado, se piensa que igualmente los resultados son válidos ya que el contexto en el cuál se realizo la actividad fue similar en las tres instancias.

6. Referencias

[1] Plan de proyecto (GPP), Nicolás Doroskevich, Sebastián Moreira, Proyecto de grado 2004 - pgdploy1, INCO – Facultad de Ingeniería – UdelaR

[2] Documento de requerimientos (RDR), Nicolás Doroskevich, Sebastián Moreira, Proyecto de grado 2004 - pgdploy1, INCO – Facultad de Ingeniería – UdelaR

[3] Documento de casos de uso (RDC), Nicolás Doroskevich, Sebastián Moreira, Proyecto de grado 2004 - pgdploy1, INCO – Facultad de Ingeniería – UdelaR

[4] Glosario (RDG), Nicolás Doroskevich, Sebastián Moreira, Proyecto de grado 2004 - pgdploy1, INCO – Facultad de Ingeniería - UdelaR

[5] Guía del plan de verificación y validación(SVVP), Sitio del curso de Ingeniería de software, INCO–FACULTAD DE INGENIERÍA – UDELAR

[6] Sitio oficial del proyecto Link All, http://www.link-all.org

[7] Descripción del proyecto Link All (2003), http://www.fing.edu.uy/inco/proyectos/linkall/Documentos/Descripcion-LinkAll.doc , INCO–Facultad de Ingeniería – UdelaR

[8] "Actividades tendientes a mejorar la calidad de los productos de software", Diego Vallespir, http://www.fing.edu.uy/~dvallesp/Tesis/webActividades/index.htm.