Documento de Arquitectura de Software

Proyecto Batuta - Generador de Aplicaciones Orquestadoras

Facultad de Ingeniería - Universidad de la República

Pablo Alvez, Patricia Foti, Marco Scalone

Junio 2006.

Índice

1 Introducción....4
1.1 Propósito....4
1.2 Alcance....4
1.3 Definiciones, Acrónimos y Abreviaciones....4
1.4 Organización del Documento....4
2 Representación de la Arquitectura....5
3 Objetivos y Restricciones....6
3.1 Requerimientos Especiales....6
3.1.1 Interoperabilidad....6
4 Vista de Casos de Uso....7
4.1 Introducción....7
4.2 Identificación de los Casos de Uso relevantes para la arquitectura 7
4.3 Descripción de los Casos de Uso relevantes para la arquitectura....8
4.3.1 Diseño de Proceso de Negocio....8
4.3.2 Transformación e Instalación de Proceso de Negocio....9
4.3.3 Ejecución de Proceso de Negocio....9
5 Vista Lógica....10
5.1 Introducción....10
5.2 Descomposición en Subsistemas....10
5.3 Descripción de los Subsistemas....10
5.4 Diseño de Subsistemas....11
5.4.1 Definición de Procesos....11
5.4.2 Ejecución de Procesos....11
5.4.3 Resolución de Servicios....13
5.5 Realización de los Casos de Uso Relevantes para la Arquitectura....15
5.5.1 Diseño de Proceso de Negocio....15
5.5.2 Transformación e Instalación de Proceso de Negocio....15
5.5.3 Ejecución de Proceso de Negocio....16
6 Vista de Deployment....19
6.1 Introducción....19
6.2 Distribución y Deployment....19
7 Arquitectura del Sistema Batuta....21

7.1 Introducción 21
7.2 Vista Lógica 21
7.2.1 Subsistema de Definición de Procesos 21
7.2.2 Subsistema de Ejecución de Procesos 22
7.2.3 Subsistema de Resolución de Servicios 23
7.3 Vista de Deployment 24
7.3.1 Distribución y Deployment 24
7.4 Vista de Implementación 25
7.4.1 Estructura del Framework 25
7.4.2 Arquitectura de la Implementación 26
8 Referencias 29

1 Introducción

1.1 Propósito

El Documento de Arquitectura de Software presenta la arquitectura del Framework Batuta a través de diferentes vistas, cada una de las cuales ilustra un aspecto en particular del software desarrollado. Se pretende de esta forma que el documento brinde al lector una visión global y comprensible del diseño general del framework desarrollado.

Luego de describir en profundidad la arquitectura del Framework Batuta, se incluye una implementación concreta de dicho framework presentando la arquitectura del Sistema Batuta construido como prueba de concepto del modelo y framework desarrollados.

1.2 Alcance

El documento se centra en el desarrollo de la vista lógica del framework. Se incluyen los aspectos fundamentales del resto de las vistas y se omiten aquellas que no se consideren pertinentes como ser el caso de la vista de procesos.

En cuanto a los componentes externos que se mencionen, se incluye una descripción de los mismos en el nivel que se considere apropiado y se indican las referencias donde consultar más información sobre los mismos.

1.3 Definiciones, Acrónimos y Abreviaciones

Ver Glosario [1].

1.4 Organización del Documento

El documento se desarrolla y organiza en base a la plantilla elaborada para el artefacto *Software Architecture Document* del proceso de desarrollo de software elaborado por *RUP* [2], adaptada a las características particulares del tipo de proyecto en desarrollo.

La sección 2 realiza una introducción a la representación utilizada de la arquitectura de forma de asegurar una comprensión cabal del documento en tal sentido.

Las siguientes secciones se abocan a la descripción de la arquitectura del Framework Batuta. Luego de una descripción inicial de los objetivos y restricciones influyentes, se desarrolla cada una de las vistas y se cierra con algunas consideraciones finales importantes.

En las secciones finales, y sobre la base de lo desarrollado anteriormente, se incluye la descripción de la arquitectura del Sistema Batuta.

2 Representación de la Arquitectura

El modelo propuesto por *RUP* [2] para representar la arquitectura utiliza el siguiente conjunto de vistas:

- Vista de Casos de Uso: lista los casos de uso o escenarios del modelo de casos de uso que representen funcionalidades centrales del sistema final, que requieran una gran cobertura arquitectónica o aquellos que impliquen algún punto especialmente delicado de la arquitectura.
- Vista Lógica: describe las partes arquitectónicamente significativas del modelo de diseño, como ser la descomposición en capas, subsistemas o paquetes. Una vez presentadas estas unidades lógicas principales, se profundiza en ellas hasta el nivel que se considere adecuado.
- Vista de Procesos: describe la descomposición del sistema en threads y procesos pesados. Indica que procesos o grupos de procesos se comunican o interactúan entres sí y los modos en que estos se comunican.
- Vista de Deployment: describe uno o más escenarios de distribución física del sistema sobre los cuales se ejecutará y hará el deploy del mismo. Muestra la comunicación entre los diferentes nodos que componen los escenarios antes mencionados, así como el mapeo de los elementos de la Vista de Procesos en dichos nodos.
- Vista de Implementación: describe la estructura general del Modelo de Implementación y el mapeo de los subsistemas, paquetes y clases de la Vista Lógica a subsistemas y componentes de implementación.
- Vista de Datos: describe los elementos principales del Modelo de Datos, brindando un panorama general de dicho modelo en términos de tablas, vistas, índices, etc.

3 Objetivos y Restricciones

Framework Batuta cumple con el modelo propuesto en [3]. Dentro de esta base de desarrollo son propiedades esenciales para la arquitectura a definir:

- diseño basado en componentes de propósito claro y concreto y con alto grado de cohesión,
- desacoplamiento entre componentes que permita el fácil reemplazo de los mismos,
- componentes altamente reutilizables.

3.1 Requerimientos Especiales

3.1.1 Interoperabilidad

El framework debe soportar la capacidad de interoperar con sistemas externos a nivel de datos y procesos.

4 Vista de Casos de Uso

4.1 Introducción

La Vista de Casos de Uso presenta un subconjunto del Modelo de Caos de Uso. Describe los casos de uso o escenarios que representen funcionalidades centrales del sistema final, que requieran una gran cobertura arquitectónica o aquellos que impliquen algún punto especialmente delicado de la arquitectura. Estos casos de uso, en conjunto con los requerimientos no funcionales, permiten descubrir y diseñar la arquitectura del sistema.

4.2 Identificación de los Casos de Uso relevantes para la arquitectura

Para el diseño del Framework Batuta, se identifican como los casos de uso relevantes desde el punto de vista de la arquitectura, los abajo mencionados:

1. Diseño de Proceso de Negocio:
   - Representa una de las funcionalidades más importantes en la gestión de procesos de negocio, el diseño de los mismos.
   - Introduce la necesidad de contar con un lenguaje gráfico de representación de procesos de negocio así como un editor con el cual poder construir dichos procesos.
   - Este componente agrega la complejidad de manejar conceptos semánticos gráficamente de forma amigable e intuitiva para el usuario. Este usuario será en general un analista de negocio que posiblemente no tenga conocimientos profundos a nivel técnico informático.
2. Transformación e Instalación de un Proceso de Negocio
   - Esta funcionalidad es sumamente importante pues permite pasar de un nivel de abstracción a otro dentro del modelo propuesto.
   - Tanto este caso como el anterior genera la necesidad de buscar un formato de representación intermedia para la especificación del proceso de negocio.
   - Una vez fijado dicho formato se debe especificar un mecanismo de transformación de uno al otro. Más precisamente poder generar una representación ejecutable del proceso a partir de su representación intermedia.
3. Ejecución de Proceso de Negocio
   - Funcionalidad básica que debe proveer el framework para posibilitar la ejecución del proceso.
   - La ejecución de un proceso de negocio es iniciada por una aplicación cliente a través de una interfaz de servicio provista por el framework.
   - Como parte esencial de la ejecución de un proceso se debe resolver la invocación de servicios a partir de una descripción semántica de los mismos.

- La invocación de un servicio a partir de una descripción semántica implica tres procesos fundamentales:
    - Búsqueda de servicios que se correspondan semánticamente con la descripción del perfil de servicio requerido.
    - Contar con un mecanismo para la selección del mejor servicio, dentro de los hallados en el proceso anterior, en base a algún criterio preestablecido. Un ejemplo de esto es seleccionar el más adecuado en base a indicadores de la calidad de servicio de los mismos.
    - En la invocación del servicio propiamente dicha se deben manejar los parámetros teniendo en cuenta sus diferentes tipos de datos, incluyendo ontologías.
- Introduce además la necesidad de especificar un formato de mensaje para la invocación de servicios y para la devolución de resultados.

4.3 Descripción de los Casos de Uso relevantes para la arquitectura

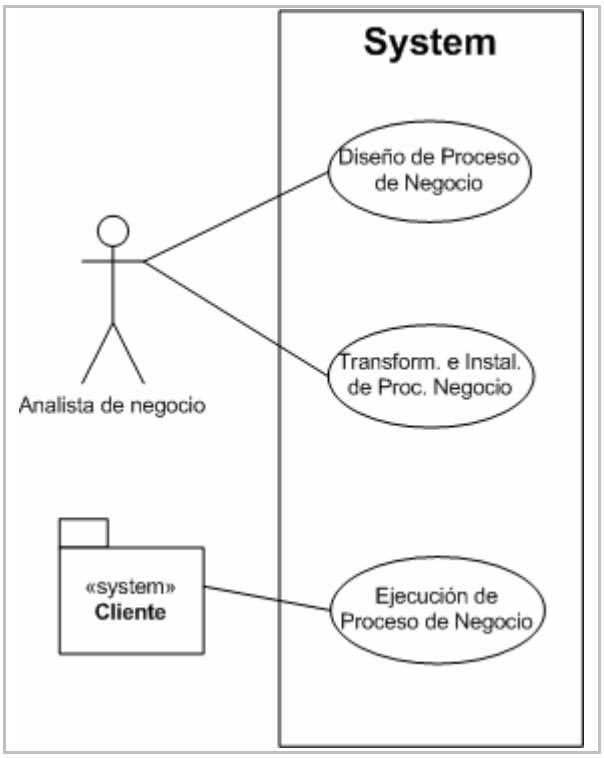

Figura 1 Diagrama de los Casos de Uso relevantes para la arquitectura.

4.3.1 Diseño de Proceso de Negocio

Nombre	Diseño de Proceso de Negocio
Actores	Analista de negocio

Sinopsis	El caso de uso comienza cuando el usuario analista de negocio indica al sistema que desea definir un proceso de negocios. El sistema proporciona a tales efectos, un lenguaje de alto nivel y una interfaz gráfica acorde que permiten definir el proceso. Durante la definición del proceso, el usuario puede especificar invocaciones directas: a la descripción semántica de un servicio o indirectas: mediante la especificación de un perfil de servicio deseado.

4.3.2 Transformación e Instalación de Proceso de Negocio

Nombre	Transformación e Instalación de Proceso de Negocio
Actores	Analista de negocio
Sinopsis	Una vez completada la definición de un proceso de negocio y obtenida la representación intermedia del mismo, el analista de negocio procede a la transformación e instalación del proceso. Para esto el sistema recibe dicha representación intermedia y un nombre para el proceso. Posteriormente, procede a realizar las transformaciones necesarias que dejen el proceso en un formato ejecutable. Finalmente, el proceso junto con los archivos necesarios para la ejecución son empaquetados y deployados en un motor de ejecución.

4.3.3 Ejecución de Proceso de Negocio

Nombre	Ejecución de Proceso de Negocio
Actores	**Aplicación Cliente**
Sinopsis	Una aplicación cliente invoca la ejecución de un proceso de negocio a través de la interfaz de servicio expuesta para tal fin. El sistema comienza la ejecución del proceso resolviendo cada una de las invocaciones a servicios especificadas. Dependiendo del tipo de invocación a resolver, y en caso de ser necesario, el sistema busca un servicio concreto que cumpla con un perfil de servicio especificado.

5 Vista Lógica

5.1 Introducción

Se presentan en este punto los sucesivos refinamientos que definen las diferentes unidades lógicas que componen la arquitectura del Framework Batuta.

El primer refinamiento realizado consiste en la descomposición en subsistemas. Los subsistemas representan cortes verticales al diseño del sistema. Cada subsistema consiste en el agrupamiento de diferentes funcionalidades relacionadas entre sí y posee la capacidad de funcionar como un sistema en sí mismo.

Posteriormente se explora la composición de cada uno de los subsistemas.

Finalmente se incluye la realización de los casos de uso descriptos en la sección anterior mediante los componentes arquitectónicos definidos.

5.2 Descomposición en Subsistemas

La descomposición propuesta, basada en el modelo *Peer to Peer*, organiza la arquitectura en un conjunto de subsistemas funcionalmente cohesivos que interactúan entre sí para cumplir sus funciones.

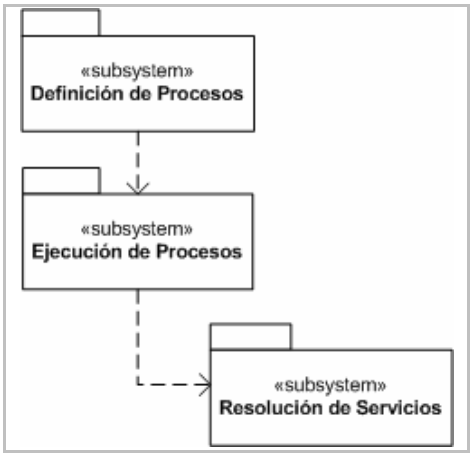

Figura 2 Principales componentes del Framework Batuta.

5.3 Descripción de los Subsistemas

- **Definición de Procesos**: Este subsistema es el encargado de proporcionar las herramientas adecuadas que le permitan al analista de negocio, diseñar de forma amigable e intuitiva el proceso de negocio. Debe permitir manipular la semántica asociada a los servicios. Finalmente, debe ser capaz de transformar la representación gráfica del proceso a una especificación intermedia del mismo que sirva de entrada para el subsistema de ejecución de procesos de negocio.

- **Ejecución de Procesos:** Tomando como entrada la representación intermedia de un proceso de negocio, realiza una serie de transformaciones que la llevan a una representación ejecutable. Este subsistema se encarga de la transformación instalación y ejecución del proceso de negocio y utiliza los servicios del subsistema *Resolución de Servicios* para la invocación de los servicios requeridos por el proceso.
- **Resolución de Servicios:** Se encarga de la resolución de un servicio especificado mediante una descripción semántica. La resolución implica la búsqueda y selección del servicio concreto que mejor se adapte al perfil de servicio requerido, y la ejecución del mismo utilizando los parámetros proporcionados.

5.4 Diseño de Subsistemas

5.4.1 Definición de Procesos

El diseño de este subsistema quedó fuera del alcance del proyecto. Se prescinde del mismo recurriendo a una herramienta externa para la implementación del framework. La salida del trabajo de este subsistema es una representación intermedia del proceso de negocio como se detalla en [3]; dicha representación intermedia es entrada del subsistema *Ejecución de Procesos* que se describe a continuación.

5.4.2 Ejecución de Procesos

El subsistema *Ejecución de Procesos* tiene dos grandes componentes como puede verse en la figura 3.

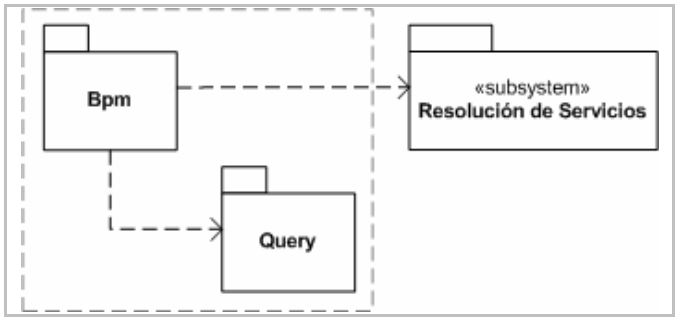

Figura 3 Componentes del Subsistema Ejecución de Procesos.

Bpm es el componente principal del subsistema. Se encarga de dos tareas fundamentales:

- la transformación de la entrada: una representación intermedia del proceso de negocio a una representación ejecutable del mismo, y
- la ejecución del proceso de negocio a partir de la representación ejecutable obtenida en el paso anterior.

Durante la ejecución del proceso, el componente recurre al subsistema *Resolución de Servicios* para resolver las invocaciones a servicios incluidas en el proceso.

El resultado de la resolución de un servicio puede ser una ontología; dentro de esta ontología pueden estar contenidos elementos a ser utilizados como resultados intermedios del proceso con los que se precise hacer algún cálculo o ser entrada de una nueva invocación a un servicio. El componente *Query* es utilizado por *Bpm* precisamente para consultar una

ontología en busca de un elemento (o conjunto de elementos) en particular. Por tal motivo, debe utilizarse un lenguaje de consulta desarrollado para tales fines. En la sección 7.2.2.2 se presenta un caso concreto que ejemplifica el funcionamiento y cometido del componente.

5.4.2.1 Diseño detallado del subsistema.

Componente Bpm.

Bpm está integrado por tres módulos tal cual se muestra en la figura 4.

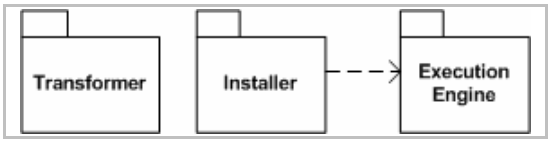

Figura 4 Diseño del componente Bpm.

- *Transformer*: recibe la representación intermedia de un proceso de negocio y genera un archivo con la representación ejecutable del mismo.
- *Installer*: se encarga de tomar la representación ejecutable de un proceso de negocio, resultado del trabajo del módulo *Transformer*, y dejarla disponible en un motor de ejecución de procesos de negocio de forma que pueda ser ejecutada.
- *Execution Engine*: este módulo representa el motor de ejecución de procesos de negocio; los procesos son deployados dentro de este componente y se hacen accesibles para ser ejecutados generando una interfaz para cada uno de ellos; en general son expuestos en forma de web service.

La figura 5 presenta la interfaz del componente *Bpm* y sus subcomponentes.

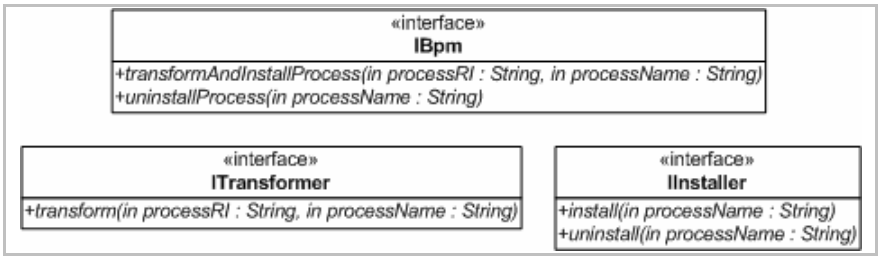

Figura 5 Interfaces – componente Bpm.

- *IBpm*
  - *transformAndInstall*: a partir de la representación intermedia de un proceso y el nombre del mismo se encarga de dejarlo disponible para ser ejecutado; para ello utiliza las operaciones *transform* de *ITransformer* e *install* de *IInstaller*.
  - *uninstallProcess*: a través de esta operación se dan de baja procesos disponibles en el motor de ejecución; utiliza la operación *uninstall* de *IInstaller*.

Como se menciona anteriormente la ejecución de un proceso de negocio se realiza accediendo a la interfaz del mismo expuesta por el motor de ejecución. Por este motivo es que no se encuentra una operación de ejecución en las interfaces presentadas en la figura 5.

5.4.3 Resolución de Servicios

La composición del subsistema *Resolución de Servicios* se muestra en la figura 6.

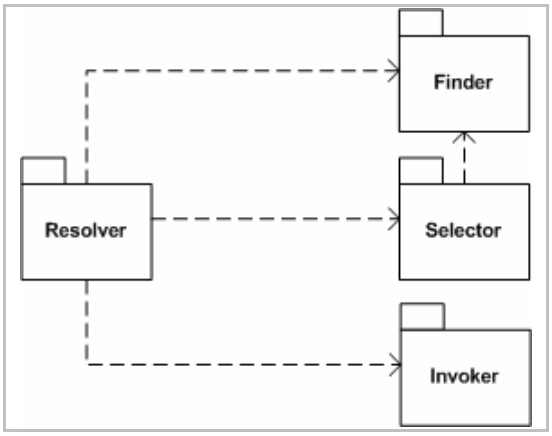

Figura 6 Componentes del Subsistema Resolución de Servicios.

El componente *Resolver* recibe las solicitudes de resolución de servicios y utiliza los componentes *Finder*, *Selector* e *Invoker* para resolver dichos pedidos.

- *Finder* recibe como entrada una descripción semántica de las características del servicio buscado y se encarga de hallar servicios concretos que cumplan con tales características. Para ello debe poder acceder a un repositorio de servicios sobre los cuales consultar y contar con una herramienta que resuelva el macheo semántico deseado: especificación semántica del servicio general buscado vs. descripción semántica de las características de cada uno de los servicios concretos. El resultado del trabajo de este componente es una lista de servicios que machean la especificación semántica indicada.

  La determinación de si un servicio machea una determinada descripción semántica se resuelve mediante un algoritmo que evalúa de acuerdo a una serie de criterios si existe la correspondencia y en que grado. Es decir que existen diferentes grados de macheo. Por este motivo la lista antes mencionada adjunta a cada servicio hallado un indicador del grado de correspondencia obtenido.

  El algoritmo y los criterios de macheo dependen del lenguaje de descripción semántica de servicios y la herramienta de resolución de macheo utilizados. Un caso concreto se detalla en [5].

- *Selector* se encarga de seleccionar uno de los servicios de la lista devuelta por el componente *Finder*. El servicio escogido depende de determinados criterios que puedan haberse especificado en el pedido de resolución de servicio (como ser aspectos de calidad) o definidos en el componente *Selector*. El criterio inmediato es el de escoger el servicio con mayor grado de macheo pero podrían utilizarse otros más complejos que implicaran una revisión más profunda de la lista de resultados; el aspecto de calidad de servicio antes mencionado es un ejemplo.

- Una vez seleccionado el servicio adecuado debe procederse a la invocación del mismo. El encargado de resolver dicha tarea es el componente *Invoker*. Para ello recibe la especificación semántica del servicio seleccionado junto con los parámetros de invocación. La especificación semántica contiene toda la

información requerida para ejecutar el servicio incluyendo los parámetros de entrada que son instanciados con los valores recibidos.

5.4.3.1 Diseño detallado del subsistema.

La presenta las interfaces del subsistema. Revisando las operaciones definidas puede verse la realización de lo descrito en la sección anterior. A continuación se repasa cada una de estas operaciones centrando la descripción en los parámetros y tipos de datos manejados.

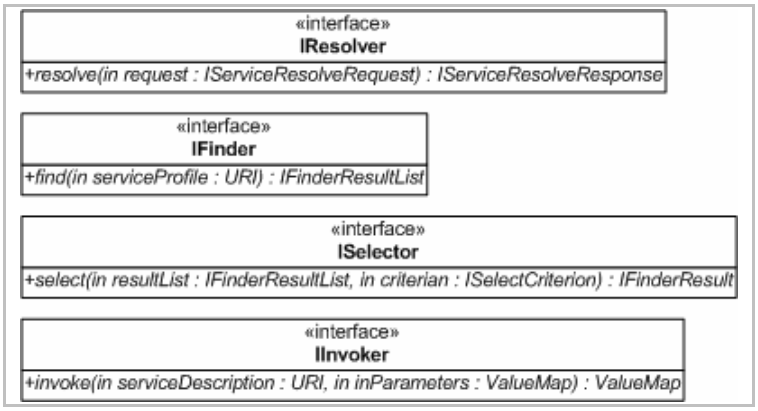

Figura 7 Interfaces – Subsistema Resolución de Servicios.

- *resolve*: es la operación que dispara la resolución de un servicio. Es invocada por el subsistema *Ejecución de Procesos*, en particular por el componente *Execution Engine* según se explica en la sección 5.4.2.1. Recibe como parámetro de entrada un objeto *IServiceResolveRequest*. La forma de este objeto podrá variar de acuerdo al lenguaje empleado para el manejo semántico de los servicios pero sustancialmente debe contener la descripción semántica del servicio a resolver y los parámetros de invocación. El resultado de esta operación se encapsula en un objeto de tipo *IServiceResolveResponse* el cual contiene los parámetros de retorno del servicio invocado y el objeto *IServiceResolveRequest* con el que se realizó la invocación.
- *find:* se encarga de hallar el conjunto de servicios que macheen el perfil de servicio deseado. A partir de un parámetro de tipo *URI* que indica dónde se halla la ontología que describe el tipo de servicio requerido, consulta el repositorio de servicios en busca de los que macheen el perfil y retorna en un objeto de tipo *IFinderResultList* la lista de servicios obtenida y el grado de macheo de cada servicio. Notar que el parámetro de entrada *ServiceProfile* que se emplea en esta operación es parte del contenido del objeto *IServiceResolveRequest* con el cual se invoca la operación *resolve*.
- *select*: se encarga de seleccionar el servicio a invocar de los incluidos en el objeto *IFinderResultList* devuelto por la función *find*. El criterio para seleccionar el servicio se representa mediante un objeto *ISelectCriterion*, el cual permite la parametrización de la selección en caso de que el componente que implemente *ISelector* así lo permita. El resultado de esta operación es el servicio elegido representado por un objeto *IFinderResult*; es decir, un elemento de la lista *resultList* recibida como entrada.
- *invoke*: mediante esta operación se ejecuta el servicio seleccionado en los pasos previos. Como parámetros de entrada recibe una *URI* apuntando a la ontología

que describe el servicio seleccionado y la lista de parámetros de invocación. La *URI* es parte del contenido del objeto *IFinderResult* devuelto por el componente *Selector*, mientras que la lista de parámetros es parte del contenido del objeto *IServiceResolveRequest* con el cual se invoca la operación *resolve*. El resultado de la operación se encapsula en un objeto de tipo *ValueMap* el cual contiene una lista de parejas que representan un parámetro de retorno del servicio y su respectivo valor.

5.5 Realización de los Casos de Uso Relevantes para la Arquitectura

5.5.1 Diseño de Proceso de Negocio

La realización del caso de uso queda a cargo del subsistema *Definición de Procesos*.

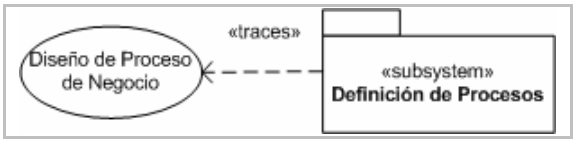

Figura 8 Realización del caso de uso Diseño de Proceso de Negocio.

Como se mencionó anteriormente el diseño de dicho subsistema no está dentro del alcance del proyecto por lo que no es posible profundizar en la realización del caso de uso.

5.5.2 Transformación e Instalación de Proceso de Negocio

La realización del caso de uso queda a cargo del subsistema *Ejecución de Procesos*.

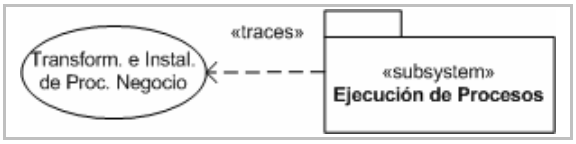

Figura 9 Realización del caso de uso Transformación e Instalación de Proceso de Negocio.

Profundizando en el diseño del subsistema *Ejecución de Procesos* de acuerdo a lo desarrollado en la sección 5.4.2.1, se presenta en la figura 10 la realización del caso de uso por parte del componente *Bpm*.

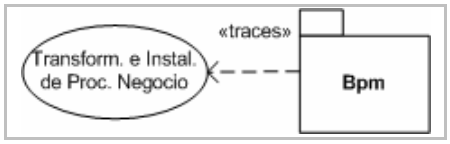

Figura 10 Realización del caso de uso Transform. e Instal. de Proc. de Negocio – nivel 2.

Bajando un paso más en el nivel de abstracción, la presenta la realización del caso de uso por parte de los subcomponentes del componente *Bpm*.

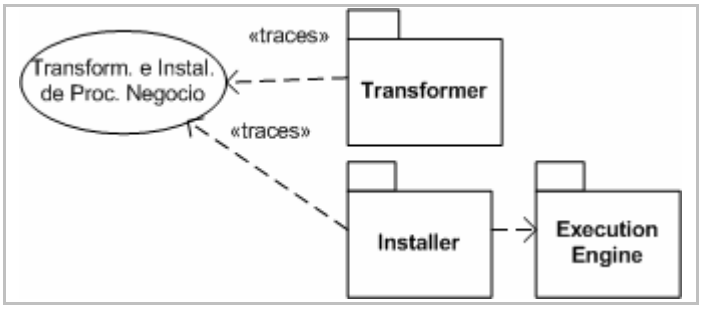

Figura 11 Realización del caso de uso Transform. e Instal. de Proc. de Negocio – nivel 3.

Finalmente, se incluye el diagrama de secuencia que describe la interacción ente los componentes para realizar el caso de uso en cuestión. La figura 12 presenta dicho diagrama.

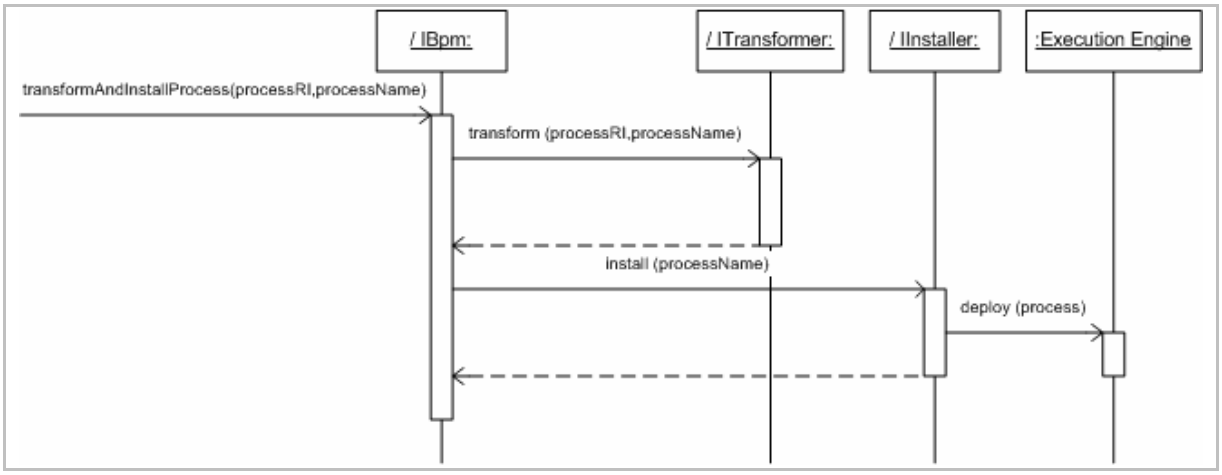

Figura 12 Diagrama de secuencia – realización CU Transform. e Instal. de Proc. de Negocio.

El componente *Bpm* recibe el pedido de transformar e instalar un proceso mediante la operación *transformAndInstall*. A partir de ello, solicita la transformación al componente *Transformer* y una vez completada dicha acción solicita la instalación al componente *Installer* el cual deja el proceso disponible para ejecutar en el *Execution Engine.*

5.5.3 Ejecución de Proceso de Negocio

La realización del caso de uso queda a cargo del subsistema *Ejecución de Procesos* el cual utiliza para cumplir tal función el subsistema *Resolución de Servicios*.

Bajando en el nivel de abstracción, se presenta en las figuras 14 y 15 la realización del caso de uso de acuerdo a los componentes presentados en las secciones 5.4.2.1 y 5.4.3.1.

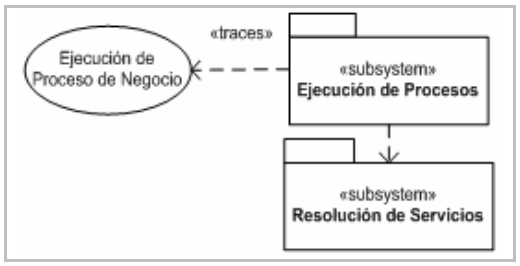

Figura 13 Realización del caso de uso Ejecución de Proceso de Negocio.

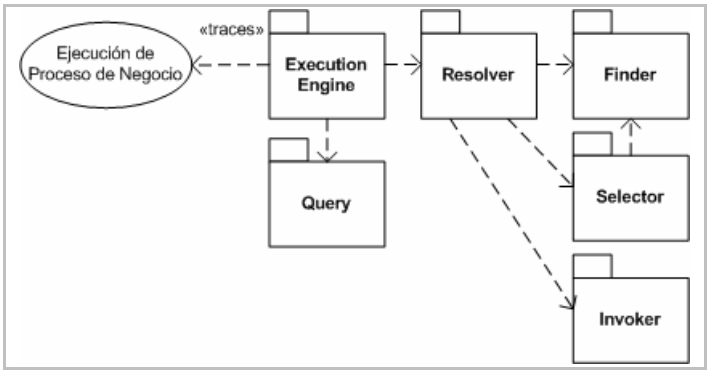

Figura 14 Realización del caso de uso Ejecución de Proceso de Negocio – nivel 2.

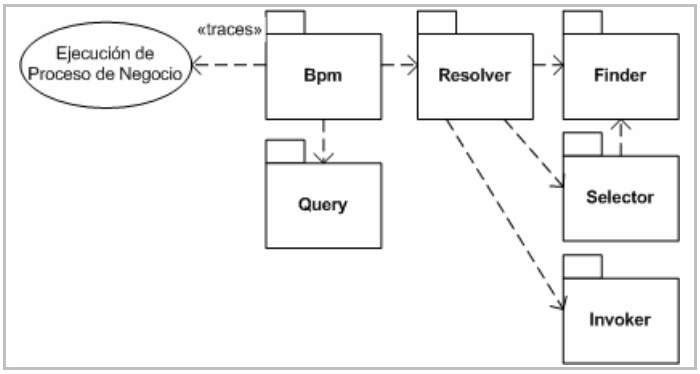

Figura 15 Realización del caso de uso Ejecución de Proceso de Negocio – nivel 3.

Finalmente se incluye el diagrama de secuencia que describe la interacción ente los componentes para realizar el caso de uso en cuestión. La figura 16 presenta dicho diagrama.

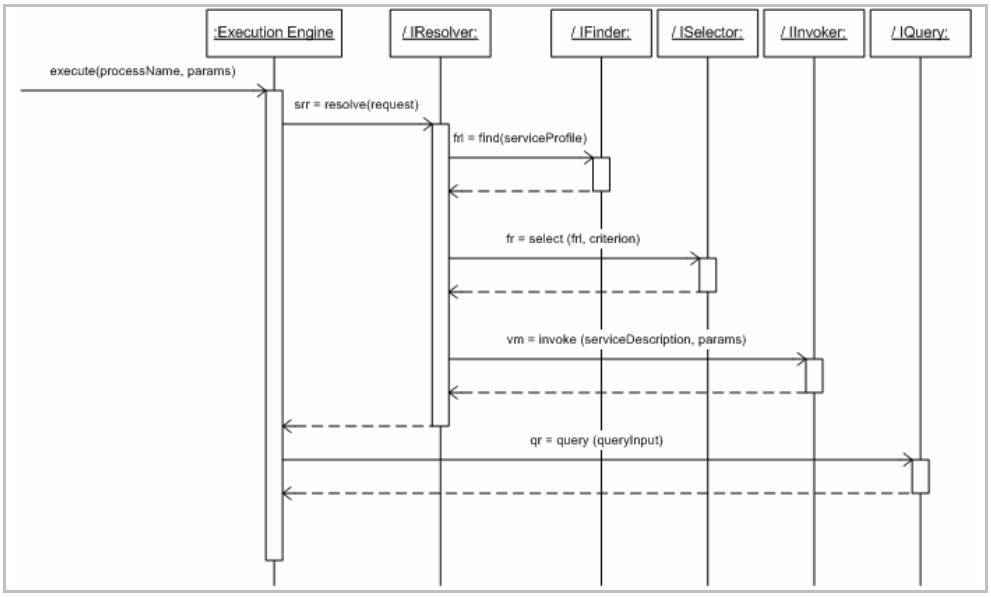

Figura 16 Diagrama de secuencia – realización CU Ejecución de Proceso de Negocio.

El diagrama grafica lo explicado en la sección 5.4.3.1. La interacción se presenta simplificada de forma de ganar claridad en la descripción.

A través de la interfaz expuesta por el motor de ejecución para acceder a cada proceso, se invoca la ejecución de un determinado proceso. Dicha ejecución implica, en el caso general, la resolución de servicios, lo cual se solicita al componente *Resolver* mediante la operación *resolve*. *Resolver* utiliza *IFinder* para obtener la lista de servicios que machean la descripción semántica del servicio a resolver, representada por el parámetro *frl* en el diagrama. Con esta lista se invoca la operación *select* de *ISelector* para escoger el servicio adecuado, el cual es invocado posteriormente utilizando el componente *Invoker*.

El resultado de la resolución del servicio es devuelto al componente *Execution Engine* el cual eventualmente podría requerir acceder a algún parámetro en particular del resultado para lo cual utiliza el componente *Query*.

6 Vista de Deployment

6.1 Introducción

Está sección describe una o más configuraciones físicas sobre las cuales se realiza el deploy del software y es ejecutado, así como la infraestructura necesaria para su instalación.

Para el caso del Framework Batuta se describe el escenario general de distribución esperado para los componentes de software antes descritos, las características de los nodos presentados y la comunicación entre los mismos.

6.2 Distribución y Deployment

La figura 17 presenta el escenario de distribución esperado para la instalación del framework. El mismo se ubica en el contexto de una organización, sobre una LAN privada y se prevé el acceso vía Internet a un repositorio de servicios.

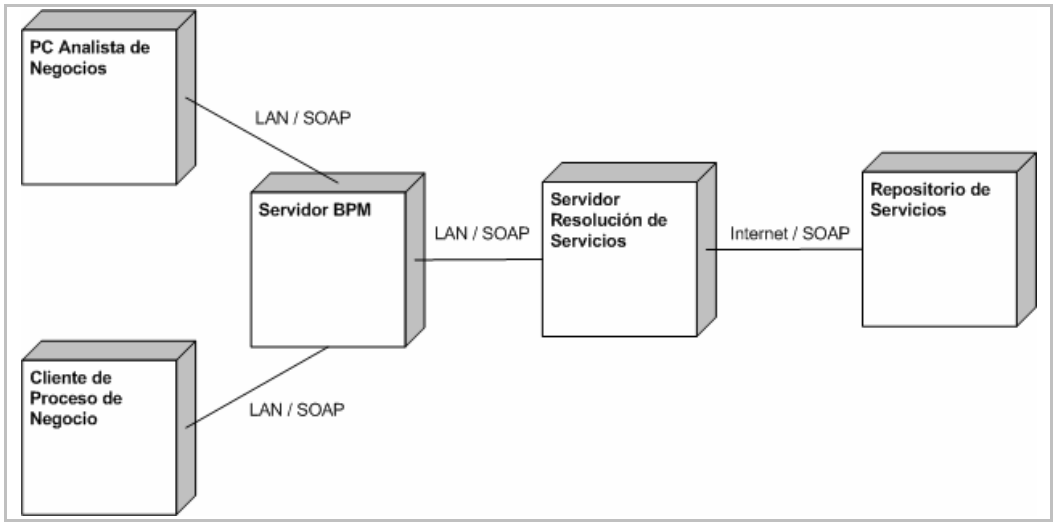

Figura 17 Escenario de distribución

A continuación se describen los nodos presentes en la figura:

- **Servidor BPM**: representa el equipo donde correrá el subsistema *Ejecución de Procesos*. Por tanto, se ejecutan aquí las funcionalidades de transformación e instalación de procesos de negocio así como el motor de ejecución de procesos. Tales servicios son expuestos mediante respectivas interfaces web service.
- **PC Analista de Negocios**: representa la estación de trabajo de un usuario analista de negocio. En este nodo correrá el subsistema *Definición de Procesos*. Se comunica con el *Servidor BPM* para la transformación e instalación del proceso definido; la comunicación es vía SOAP a través de la interfaz web service expuesta por dicho nodo.
- **Cliente de Proceso de Negocio**: representa el equipo donde corre una aplicación cliente que desea ejecutar un proceso de negocio disponible en el *Servidor BPM*. No hay

componentes del framework corriendo en este nodo. Los procesos son ejecutados invocando el web service publicado para cada uno de ellos.

- **Servidor Resolución de Servicios**: representa el equipo donde correrá el subsistema *Resolución de Servicios*. Recibe pedidos del motor de ejecución de servicios corriendo en el *Servidor BPM*. La comunicación es nuevamente a través de una interfaz web service.
- **Repositorio de Servicios**: representa un equipo donde se mantendrá un registro de servicios disponibles a los cuales podrá acceder el subsistema *Resolución de Servicios* para buscar los servicios adecuados para una determinada solicitud. Podría existir más de una instancia de este nodo a la cual acceder.

Las interfaces de comunicación previstas para los nodos, basadas en web services, posibilitan plantear múltiples escenarios de distribución y permiten el planteo de una infraestructura multiplataforma.

7 Arquitectura del Sistema Batuta

7.1 Introducción

Sobre la base arquitectónica presentada, se describe a continuación la arquitectura del Sistema Batuta realizado como prototipo – prueba de concepto del Proyecto Batuta. Dicho sistema puede verse como una realización particular del framework general desarrollado y fue construido a partir de la integración de diferentes tecnologías y herramientas presentes en el área de investigación abarcada.

La descripción se organiza repasando las vistas presentadas anteriormente, detallando en cada una de ellas como se resolvió la realización de los componentes definidos, que componentes de infraestructura, tecnologías, herramientas y lenguajes se utilizaron.

La vista de casos de uso claramente no aplica en esta descripción puesto que no hay nada que haya debido adaptarse respecto a los requerimientos en la solución desarrollada. El Sistema Batuta es una solución particular al problema general planteado y fue construido sobre la arquitectura del framework la cual fue guiada por los casos de uso presentados.

Se considera prudente repasar los siguientes puntos para entender el resto del desarrollo de la sección:

- El lenguaje de modelado de procesos utilizado es *BPMN*.
- El lenguaje de ejecución de procesos utilizado es *BPEL*.
- El lenguaje de representación semántica de conceptos utilizado es *OWL*.
- El lenguaje de descripción semántica de web services utilizado es *OWL-S*.

Los fundamentos de estas decisiones pueden hallarse en *Descripción del Modelo Batuta* [3] y los conceptos clave de cada lenguaje pueden consultarse en el documento *Estado del Arte* [4].

7.2 Vista Lógica

La figura 18 presenta una vista global de los subsistemas del framework y las herramientas externas que se utilizaron para lograr cumplir los cometidos definidos para cada subsistema.

7.2.1 Subsistema de Definición de Procesos

Como se menciona en la sección 5.4.1 el diseño de este subsistema no está dentro del alcance del proyecto. Para resolver este aspecto se recurrió a la utilización de una herramienta externa. Dada la tarea a cumplir por este subsistema se requería una herramienta que permitiera el modelado de procesos de negocios de forma clara e intuitiva y la posibilidad de manejar aspectos de semántica referidos a los servicios involucrados en el proceso.

Ante la no disponibilidad de una herramienta que soportara de forma nativa el manejo de representaciones semánticas (*OWL* y *OWL-S*), se recurrió a la utilización de un producto de modelado de procesos de negocio, integrando los aspectos semánticos a través de referencias a ontologías manejadas como parámetros de invocación o retorno de los servicios invocados.

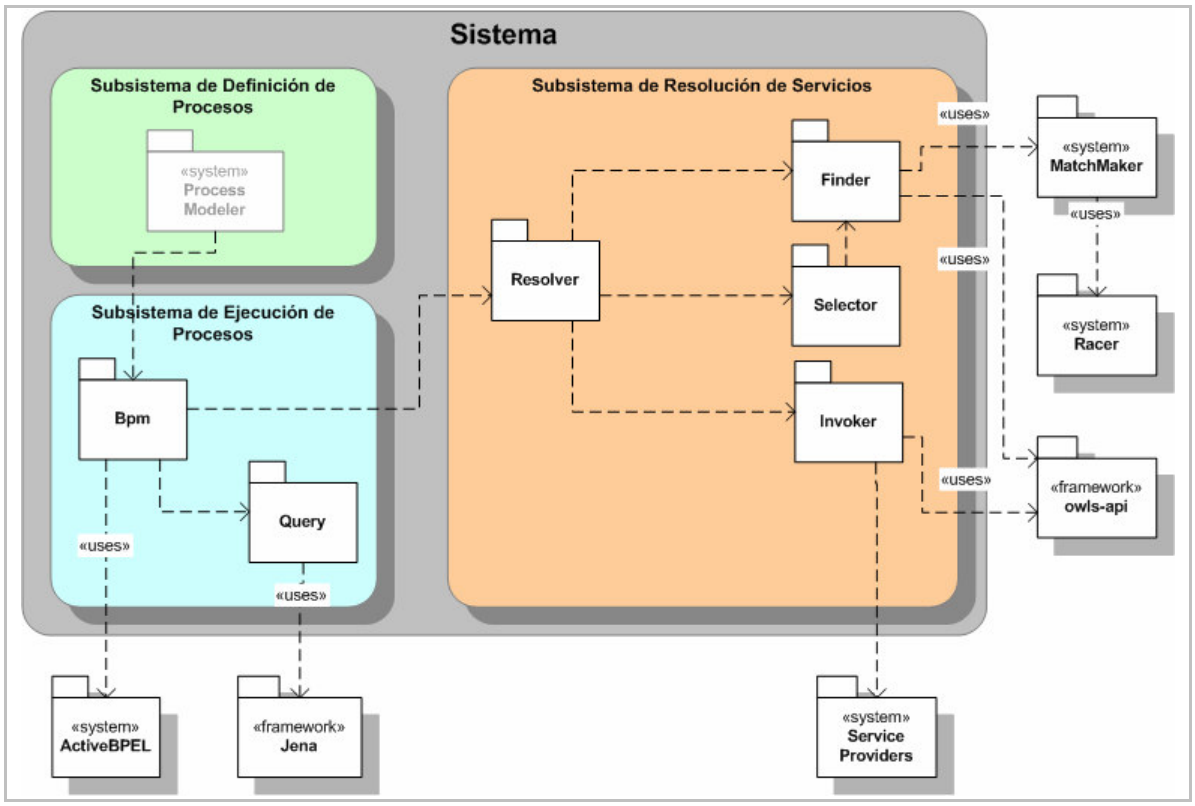

Figura 18 Vista lógica general de la realización del framework para el Sistema Batuta

Más concretamente, ante la invocación de un servicio, se especifica en un parámetro la *URI* donde hallar la descripción del perfil de servicio requerido (el archivo *ServiceProfile*), y entre los parámetros de invocación, a los que sean conceptos semánticos se le asigna como valor la *URI* donde ubicar la instancia de ontología adecuada.

El detalle de este manejo puede consultarse en [5] donde se puede encontrar información técnicamente más detallada.

El producto utilizado, en calidad de versión de evaluación, es *Process Modeler for Microsoft Visio* de la compañía *ITP Commerce* el cual funciona como *add-in* de *MS Visio* [6].

La definición de un proceso de negocio mediante este producto puede exportarse a una representación *XML* la cual dadas sus características es adecuada para ser utilizada como lenguaje de representación intermedia, *LRI* en la terminología del Modelo Batuta.

Este archivo *XML* es por tanto la salida del *Subsistema de Definición de Procesos*.

7.2.2 Subsistema de Ejecución de Procesos

Como se explica en la sección 5.4.2, el subsistema se divide en dos componentes: *Bpm* y *Query*.

7.2.2.1 Componente Bpm

Dentro del componente *Bpm* tenemos tres grandes funcionalidades: transformación de la representación intermedia de un proceso a su representación ejecutable, instalación del proceso en un motor de ejecución y ejecución del proceso de negocio.

De acuerdo a lo desarrollado en la sección 5.4.2.1 las funcionalidades de transformación e instalación son resueltas por un módulo que implementa la interfaz *IBpm*, en particular la operación *transformAndInstallProcess*. La misma recibe una representación intermedia, *LRI* – el archivo *XML* salida del *Subsistema Definición de Procesos* - y un nombre de proceso. A partir de tal entrada, transforma la representación *LRI* hasta obtener una representación ejecutable, en lenguaje *BPEL*. Para dicha tarea recurre al módulo *Transformer* el cual mediante transformaciones *XSLT* logra la salida deseada.

Los detalles de las transformaciones realizadas se encuentran en [5].

Una vez obtenida la representación ejecutable –*BPEL*- del proceso de negocio, se recurre al módulo *Installer* para deployar el proceso en el motor de ejecución. Este módulo empaqueta los archivos resultado de las transformaciones: *BPEL*, *WSDL* y descriptores de deployment específicos del motor de ejecución, y realiza el deploy.

La funcionalidad de ejecutar un proceso de negocio es realizada mediante una herramienta externa: el motor de ejecución, *open source*, *ActiveBPEL* de la compañía *Active Endpoints* [7].

Los procesos deployados en el motor son expuestos como web services y por tanto son ejecutados invocando el *web service* adecuado.

7.2.2.2 Componente Query

Como se menciona en la sección 5.4.2, este componente se encarga de realizar consultas sobre ontologías devueltas como resultado de la resolución de un servicio. El mismo es utilizado por el motor de ejecución de procesos y resuelve su tarea mediante la utilización del *Framework Jena* para el manejo de ontologías [8]. El componente *Query* desarrollado para el *Sistema Batuta* soporta dos lenguajes de consulta de ontologías *RDQL* y *SPARQL* [4].

7.2.3 Subsistema de Resolución de Servicios

Tal cual se detalla en la sección 5.4.3, el subsistema se organiza en cuatro componentes. El componente *Resolver* recibe las solicitudes de resolución de servicios de parte del motor de ejecución y utiliza los componentes *Finder*, *Selector* e *Invoker* para resolver dichos pedidos.

7.2.3.1 Componente Resolver

Se resuelve implementando la interfaz *IResolver* especificada en 5.4.3.1. Tal interfaz es expuesta como web service y de esa forma la funcionalidad de resolución de servicios se hace accesible para el motor *ActiveBPEL*.

7.2.3.2 Componente Finder

La búsqueda de los servicios adecuados para una determinada especificación semántica se resuelve en base al lenguaje *OWL-S* como se menciona previamente.

El componente recibe la referencia a un perfil de servicio, la *URI* de un archivo *ServiceProfile*, y utiliza el componente externo *OWL-S / UDDI Matchmaker* [9] para resolver la búsqueda de servicios que macheen el perfil requerido. La interacción con el *OWL-S / UDDI Matchmaker*

se realiza a través de una *API* provista por la misma herramienta. Al realizar una consulta el marchmaker retorna una lista de referencias a los perfiles - *ServiceProfile* – de los servicios que cumplen con el perfil requerido junto con su *score* de macheo. Dichos perfiles son posteriormente procesados para obtener la referencia al *Service,* necesario para su posterior la invocación.

Los servicios son registrados en un repositorio *UDDI* extendido por la propia herramienta para soportar el manejo de publicación y consulta de servicios en base a perfiles *OWL-S*. La implementación de *UDDI* utilizada por *OWL-S/UDDI Matchmaker* es *jUDDI* [10].

Para el procesamiento semántico *OWL-S/UDDI Matchmaker* utiliza el razonador *Racer* de la compañía *Racer Systems* [11].

Más información sobre la utilización de *OWL-S/UDDI Matchmaker*, el algoritmo de macheo, y otros detalles pueden consultarse en [5].

La salida del *Finder* es una lista de resultados donde cada elemento contiene entre otras cosas: la referencia al servicio encontrado y un atributo denominado *score* que indica el grado de macheo entre el perfil de servicio consultado y el servicio hallado.

7.2.3.3 Componente Selector

Este componente se encarga de seleccionar uno de los elementos de la lista anterior de acuerdo a un determinado criterio. En el caso del *Sistema Batuta* el criterio utilizado es el de seleccionar el elemento de mayor *score*.

7.2.3.4 Componente Invoker

Una vez seleccionado el servicio que mejor cumple con el perfil, el componente *Invoker* procede a la invocación del mismo. La ejecución citada, así como la manipulación general de los archivos OWL-S se realiza utilizando el framework *OWL-S API* de la organización *Mindswap* [12].

Este componente recibe la referencia al servicio seleccionado con la cual crea una instancia del servicio utilizando la *OWL-S API*. Una vez que se cuenta con el servicio instanciado se puede realizar la invocación obteniendo su proceso y seteando los valores de los parámetros de invocación.

7.3 Vista de Deployment

La figura 19 ilustra el deployment del Sistema Batuta sobre la base del escenario de distribución presentado en la figura 17 como parte del desarrollo de la vista de deployment del Framework Batuta (sección 6).

7.3.1 Distribución y Deployment

- **Servidor BPM**: en este nodo y sobre el servidor de aplicaciones *JBoss* [13] se realiza el deploy de los componentes del subsistema *Ejecución de Procesos* y las herramientas externas utilizadas: el motor de ejecución de procesos *ActiveBPEL* y el framework *Jena API*.
- **PC Analista de Negocios**: en la estación de trabajo del analista de negocio se instala el componente externo *Process Modeler* previamente mencionado.

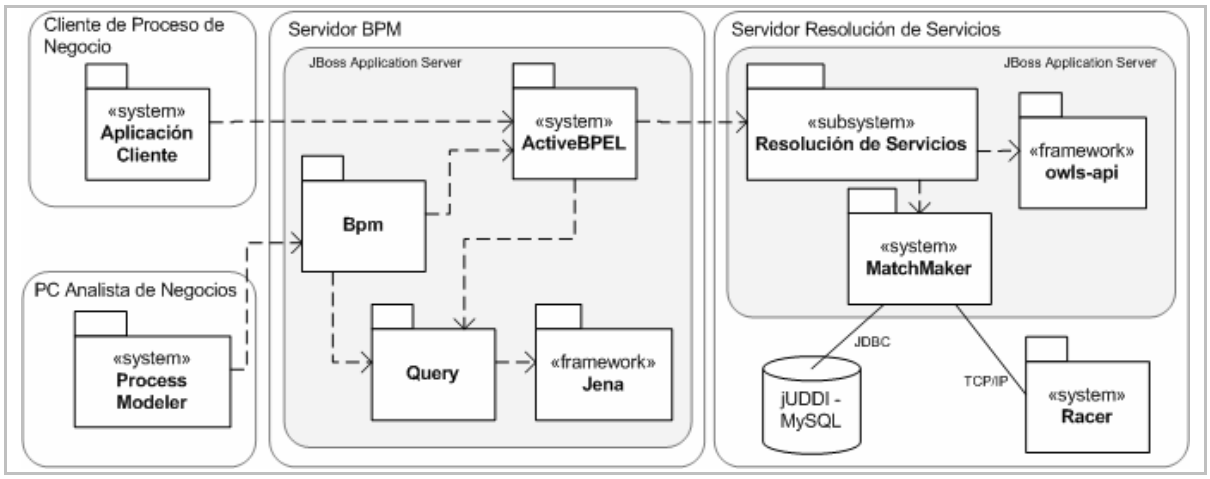

Figura 19 Deployment del Sistema Batuta.

- **Cliente de Procesos de Negocio**: en este nodo corre una aplicación cliente del subsistema *Ejecución de Procesos de Negocio* la cual accede a los procesos deployados en el motor *ActiveBPEL* a través de la interfaz de web service que esta herramienta expone para cada servicio.
- **Servidor Resolución de Servicios**: en este nodo y sobre el servidor de aplicaciones *JBoss* se realiza el deploy de los componentes del subsistema *Resolución de Servicios* y las herramientas externas utilizadas: el componente *OWL-S/UDDI Matchmaker* y el framework *OWL-S API*. Así mismo, corren en este nodo el sistema razonador *Racer*, dependencia del sistema *OWL-S/UDDI Matchmaker*, y un servicio *MySQL* donde se encuentra la base de datos *registry* con el esquema *JUDDI* adecuado para el registro de los web services que componen el repositorio de servicios. La comunicación entre el *matchmaker* y *MySQL* es vía *JDBC* y con el sistema *Racer* vía *TCP/IP*.

7.4 Vista de Implementación

En esta sección se presentan los ejecutables y artefactos construidos para la implementación del Sistema Batuta.

El sistema aquí planteado es una implementación del framework descrito en las secciones anteriores que cumple con la especificación 1.4 de *J2EE*.

7.4.1 Estructura del Framework

La implementación del framework se empaquetó en dos *JARs* independientes, uno para el subsistema *Ejecución de Procesos* y otro para el subsistema *Resolución de Servicios* tal como se muestra en la figura 20.

La implementación del Sistema Batuta se enmarca en la tecnología *J2EE*, más precisamente el subsistema *Resolución de Servicio*s está implementado por un conjunto de clases que utilizan los servicios de una serie de *session beans* que realizan las tareas concretas.

Los *beans FinderBean*, *SelectorBean* e *InvokerBean* realizan las tareas de búsqueda, selección e invocación de servicios respectivamente.

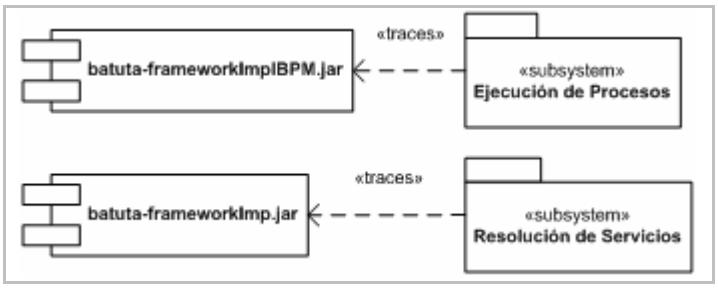

Figura 20 Mapeo de subsistemas

El *bean ReveiverBean* se encarga de recibir los mensajes de solicitud de resolución de servicios. Es el que implementa la interfaz de web service que utiliza el subsistema de ejecución de servicios para resolver los servicios abstractos.

Para el subsistema *Bpm* se cuenta con el *session bean BusinessProcessManagerBean* que depende de *batuta-frameworkImplBPM.jar* con la implementación del subsistema de ejecución de procesos de negocio.

Estos *beans* se empaquetaron en *JARs* separados e independientes de sus interfaces, es decir, sus interfaces local y remota se empaquetaron en un *JAR* independiente con el sufijo '-cliente'. Esto es así pues la implementación del framework solo requiere conocer sus interfaces.

Todos estos *JARs* fueron empaquetados en un *EAR* llamado *BatutaEAR.ear*.

7.4.2 Arquitectura de la Implementación

Se presenta a continuación un diagrama que muestra las dependencias entre los *JARs* que contienen la definición de las interfaces y la implementación concreta del framework.

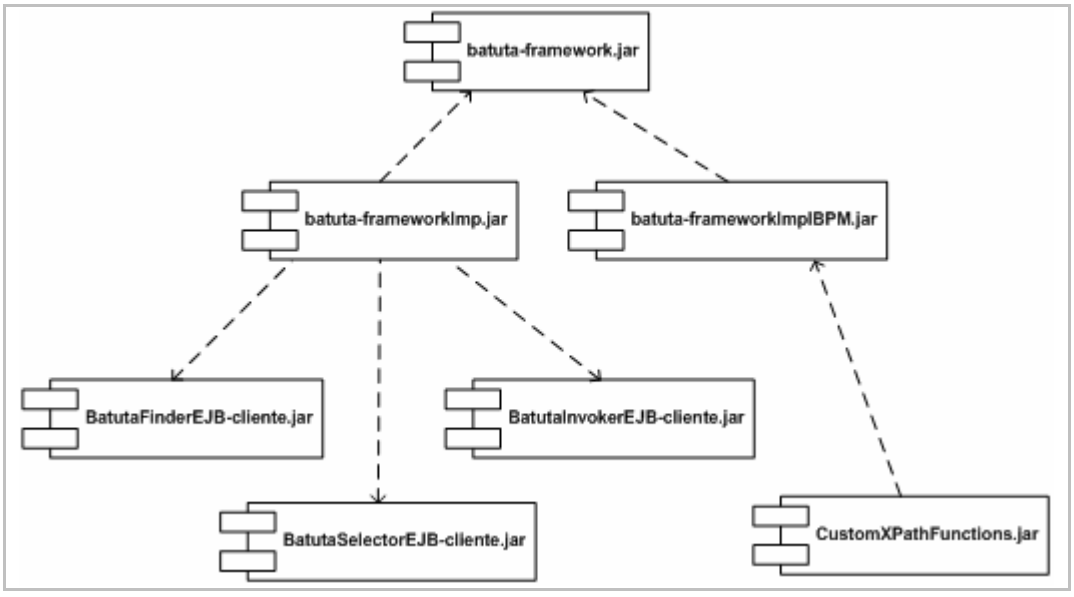

Figura 21 Dependencia de JARs del sistema

Se presenta a continuación la descripción detallada de los artefactos más importantes de los subsistemas *Ejecución de Procesos* y *Resolución de Servicios* desde el punto de vista de su implementación.

7.4.2.1 Subsistema Ejecución de Procesos

El *JAR CustomXPathFunctions.jar* contiene la implementación de las funciones de *XPath* creadas como extensión del motor de ejecución, con el fin de dotar a este último de la capacidad de ejecutar consultas sobre ontologías, y manipular el mensaje de invocación del componente *Resolución de Servicios*.

Como se muestra en la la implementación del subsistema presenta una dependencia al paquete *XSLT Templates*. Este paquete contiene el conjunto de plantillas *XSLT* que se utilizan para las transformaciones entre *LRI* y *BPEL*.

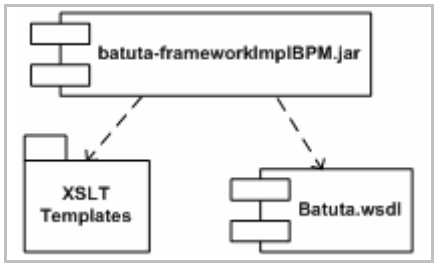

Figura 22 Dependencia del subsistema BPM

Se puede apreciar también la interfaz de servicio del subsistema *Resolución de Servicios*. Esta dependencia se da en el componente de transformación ya que las invocaciones abstractas se traducen a invocaciones al componente de resolución de servicios.

Se presenta a continuación un fragmento de *Batuta.wsdl* que describe la interfaz de servicio del subsistema *Resolución de Servicios*. Se omitieron las definiciones de *namespaces* y de tipos para simplificar el código.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/services/Batuta" ... >
   <wsdl:types>
       ...
   </wsdl:portType>
   <wsdl:binding name="BatutaSoapBinding" type="impl:Receiver">
      <wsdlsoap:binding                                          style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="semanticInvoker">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="semanticInvokerRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="semanticInvokerResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="ReceiverService">
      <wsdl:port binding="impl:BatutaSoapBinding" name="Batuta">
         <wsdlsoap:address location="http://localhost:8080/axis/services/Batuta"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions
```

7.4.2.2 Subsistema Resolución de Servicios

Como se mencionó anteriormente, se cuenta con un *session bean* para cada componente del subsistema de resolución de servicios. Como se ve en la todos los *EJBs* dependen de *batuta-framework.jar* pues este último contiene las interfaces definidas para el framework.

En el diagrama se agregaron las dependencias a las librerías *Matchmaker Client* y *OWL-S API*, necesarias para interactuar con el subsistema externo encargado del macheo semántico y para manipular las descripciones semánticas de los servicios respectivamente.

Se aprecia también la interfaz de web service que presenta el *EJB BatutaReceiverEJB* empaquetado en *BatutaReceiverEJB.jar*. Esta interfaz utiliza el *wsdl* mencionado en la sección anterior.

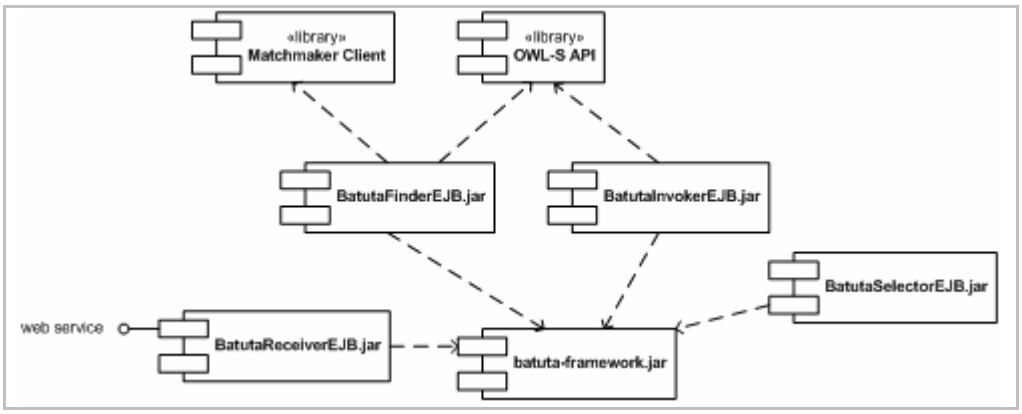

Figura 23 Dependencia de JARs del subsistema de Resolución de servicios.

8 Referencias

[1] Proyecto de Grado Batuta – Generador de Aplicaciones Orquestadoras. *Glosario*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005.
http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026-Glosario.pdf

[2] Rational Unified Process. Rational Software, IBM, 2003.
http://www-306.ibm.com/software/awdtools/rup/

[3] Proyecto de Grado Batuta – Generador de Aplicaciones Orquestadoras. *Descripción del Modelo Batuta*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005.
http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026-DescripcionModeloBatuta.pdf

[4] Proyecto de Grado Batuta – Generador de Aplicaciones Orquestadoras. *Estado del Arte*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005.
http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026-EstadoDelArte.pdf

[5] Proyecto de Grado Batuta – Generador de Aplicaciones Orquestadoras. *Especificación Complementaria del Modelo Batuta*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005.
http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026-EspecificacionComplementaria.pdf

[6] Process Modeler for Microsoft Visio™ – *ITP Commerce*.
http://www.itp-commerce.com/

[7] ActiveBPEL – *Active Endpoints*
http://www.active-endpoints.com/

[8] Jena Framework
http://jena.sourceforge.net/index.html

[9] OWL-S/UDDI Matchmaker
http://www.daml.ri.cmu.edu/matchmaker/

[10] jUDDI
http://ws.apache.org/juddi/

[11] Racer
http://www.racer-systems.com/

[12] OWL-S API – Mindswap – Maryland Information and Network Dynamics Lab Semantic Web Agents Project.
http://www.mindswap.org/2004/owl-s/api/

[13] JBoss
http://labs.jboss.com