# Arquitectura de Comunicación entre Frameworks Jade-Symfony

# Communication Architecture between Jade–Symfony's Frameworks

Paola J. Rodríguez C., MSc. y Santiago Gómez R., Ing
Docente EISC - Universidad del Valle, Ing. Sistemas – Universidad del Valle
paolajr@eisc.univalle.edu.co, sagorico@univalle.edu.co

Recibido para revisión 26 de Marzo de 2007, aceptado 15 de Junio de 2007, versión final 21 de junio de 2007

***Resumen*—La inclusión del uso de Agentes dentro del campo del desarrollo de software es una de las principales áreas de estudio en la actualidad. Específicamente, el desarrollo de aplicaciones Web que aprovechen las ventajas que ofrece la tecnología de agentes y en especial las distintas propuestas de comunicación entre frameworks son esfuerzos importantes dentro de este campo. En este sentido, este artículo describe una arquitectura de comunicación entre dos de los principales frameworks de desarrollo de aplicaciones web (Symfony) y de agentes (Jade). La arquitectura propuesta fue usada para aplicar elementos de adaptatividad de interfaz en el proyecto Plataforma Experimental para Sistemas de Recomendación, Descubrimiento de Conocimiento Interfaces Adaptativas y Consultas Avanzadas (PREDICA).**

**Palabras Clave— Ingeniería del Software, Arquitecturas Web, Agentes Software, Symfony, Jade.**

***Abstract*—Use software agents are a main study topic in development software domain. Nowadays, develop Web applications that take advantage of agent's technology, and in specific manner proposals about communications between different development frameworks are important efforts at software field. The paper describes a communication architecture between two major frameworks for development Web applications (Symfony) and Agents applications (Jade). Our architecture was used to enforce adaptative Interfaces for "Plataforma Experimental para Sistemas de Recomendación, Descubrimiento de Conocimiento Interfaces Adaptativas y Consultas Avanzadas" project.**

***Key words*— Software Engineering, Web Software Architectures *Software Agents, Symfony, Jade.***

## I. INTRODUCCIÓN

El propósito del proyecto PREDICA, fue desarrollar una plataforma experimental para facilitar la búsqueda de documentos en el área de computación cuya interfaz se adapte a un modelo de usuario definido y que ofrezca recomendaciones con base en un perfil de consulta [3]. Globalmente la arquitectura de software de PREDICA (*Figura 1)*, se presenta mediante una vista del sistema que incluye sus componentes principales, la conducta de esos componentes y las formas en que estos interactúan y se coordinan para alcanzar la misión del sistema: Gestionar la interacción del usuario, Manejar las tareas de procesamiento internas, Efectuar la navegación y Presentar el contenido.

**Figura 1.** Arquitectura Global de PREDICA

Dado que se optó por el uso de la tecnología de agentes como mecanismo para monitorear el comportamiento del usuario, capturar sus interacciones, enriquecer el modelo de usuario definido y tomar las correspondientes decisiones de

adaptación, como restricción [1], los agentes deben ejecutarse tomando información de las interacciones del usuario sobre la interfaz que se le despliega y sin obstruir el resto de funcionalidades que provee el sistema, es decir, el trabajo de los otros módulos que conforman el aplicativo [2].

De acuerdo al contexto anterior, fue necesario definir una Arquitectura de comunicación entre el aplicativo Web desarrollado en Symfony [4] y el aplicativo de agentes desarrollado en Jade [5]. Para esto, se definieron dos perspectivas: Comunicación Inter-módulos (entre el Módulo de Interfaz y los otros módulos que componen la Biblioteca) y Comunicación Inter-Plataformas (entre el Componente que infiere la adaptación -agentes- y Módulo de Interfaz – aplicativo Web-).

## II. ARQUITECTURA DE COMUNICACIÓN INTER-MÓDULOS

Para la definición de la arquitectura de comunicación inter-módulos, se realizó un análisis de la arquitectura propia de Symfony y se propuso e implementó una adaptación de la misma.

Symfony es un *framework* basado en el modelo MVC (*Model-View-Controller*), donde el *Modelo* representa la lógica del negocio, es decir, las reglas, restricciones y condiciones definidas para la operación de la aplicación, la V*ista* se encarga de generar la página Web con la cual el usuario va a interactuar, y el *Controlador* responde a las interacciones del usuario, y se encarga de generar cambios ya sea en la *Vista* o en el *Modelo*.

Específicamente, la parte donde se centró el análisis realizado fue el *Controlador*, el cual se divide en un *Controlador Frontal* y un conjunto de *Acciones*. El primero se encarga de redireccionar los pedidos que el usuario hace sobre la vista, al par modulo/acción correspondiente. El segundo, es el componente de código mínimo que se puede crear en el framework, donde se ilustra como una función recibe unos parámetros de entrada (abstracción del *request HTTP* proveída por Symfony), y retorna el llamado a una vista.

La propuesta desarrollada consistió en crear tres tipos de acciones definidas:

a. *ContentManager,* acción principal dentro del Módulo de Interfaz que permite la entrada única a cada componente dentro del sistema. Se encarga de las siguientes funcionalidades:
   a. Validación básica del contenido esperado del *request HTTP*, con el fin de evitar manipulaciones de URLs a través del navegador o manipulaciones del *request* a través de un *socket* (intento de acceso ilegal al sistema); evitando con esta validación de alto nivel dentro de la arquitectura del *framework* el procesamiento extra a las capas mas bajas.
   b. Redireccionamiento al modulo/acción correspondiente dependiendo de las variables enviadas a través de formas o enlaces en la interfaz Web.
   c. Validación de privilegios del usuario (basados en variables de la sesión inicializadas por el modulo de usuarios), para el acceso a cualquier pagina del sitio.
   d. Llamados al *framework* del Agente. Componente que se encarga de las inferencias para realizar la adaptación.

   El *ContentManager* es entonces la única acción a la que implícitamente tiene acceso el usuario. No obstante, ninguno de los módulos, por definición arquitectural podrá llamar a esta acción. Finalmente es importante resaltar que esta acción maneja un identificador de escenario que indica la interfaz a la que desea acceder el usuario. Estos identificadores se configuran en un archivo que contiene las reglas de enrutamiento de la aplicación.

b. *TemplateManager*, acción que hace las veces de administrador de plantillas del sistema y a la que comúnmente llamarán las acciones de los otros módulos. Esta acción tiene como funcionalidades el llamado a la plantilla correspondiente y la asignación de ciertas variables que serán utilizadas en la vista de validación de formas.
c. *SessionStartManager,* acción encargada de realizar las gestiones pertinentes al momento del inicio de sesión del usuario, las cuales comprenden:
   . Carga del menú personalizado del usuario.
   . Inicialización del agente del usuario (comunicación con plataforma JADE)
d. *UserMenuManager*, acción cuya función es actualizar el menú del usuario, en caso de ser cambiado de un grupo de trabajo y por ende sus privilegios.

El llamando a cualquiera de las acciones definidas se hará por medio de la abstracción *request HTTP*, anteriormente mencionada, y usando entre otras, las siguientes variables:

. T_ID: El identificador de la plantilla que se necesita desplegar en la vista. Este identificador es el que usa el *TemplateManager* para saber cual plantilla cargar, con sus datos apropiados, y encargarse de realizar el retorno a la vista apropiado.
. T_VARS: Arreglo asociativo (PHP) que contiene datos a ser utilizados en la plantilla correspondiente, que usualmente corresponden a una consulta realizada, como por ejemplo la información personal de usuario o la información de un documento digital a ser modificado.
. E_ID, S_ID: Los identificadores de error o éxito (respectivamente), que indican a la plantilla correspondiente que hay un error, una advertencia o un mensaje de éxito a ser desplegado al usuario, para lo cual se utilizo un *Template Partial*, que sirviera como

repositorio central de todos los mensajes de los diferentes módulos.

. M_VARS: Arreglo asociativo (PHP) que contiene información adicional sobre un mensaje de error o éxito, específicamente a ser usada con los denominados mensajes dinámicos. Un ejemplo es una lista de usuarios a la que no se les pudo enviar un correo electrónico, o una lista de usuarios que fueron eliminados exitosamente.

El flujo de datos normal desde que el usuario realiza una petición al sistema, hasta que es desplegada la vista es la siguiente:

. El usuario realiza la petición por medio de algún elemento de la interfaz.
. El controlador frontal recibe la petición, y mediante la consulta en el archivo de configuración de las reglas de enrutamiento, decide llamar al *ContentManager* con el indicador de escenario correspondiente. En caso de que la URL pedida por el usuario no esté configurada, se llama a una acción alterna, que muestra en pantalla un error personalizado correspondiente al error HTTP 404. Según RFC 2616 [6].
. El *ContentManager* recibe el control de la petición, y valida si el usuario tiene los permisos suficientes para acceder a la interfaz pedida. En caso que no los tenga (manipulación de la petición por URLs u otro caso), redirecciona el usuario a la pagina de inicio. Si los privilegios de la sesión permiten al usuario acceder a la interfaz. pedida, evalúa las variables necesarias para validar si estas son correctas (integridad y no manipulación de la petición), y pasa el control a la acción adecuada en el modulo correspondiente.
. La acción del modulo correspondiente, toma el control de la petición y se encarga de realizar la funcionalidad requerida en el proceso. Luego, retorna el control llamando al *TemplateManager*.
. El *TemplateManager* recibe el control de la petición, y dado que en este punto ya todas las validaciones han sido realizadas a alto (*ContentManager*) y a bajo nivel (acción en otro modulo), simplemente se encarga de identificar la plantilla a cargar, establece los datos dinámicos necesarios y retorna el control a Symfony, mediante una llamada explicita a la vista.
. La plantilla correspondiente recibe el control, y se despliegue en pantalla.

Así, la propuesta desarrollada reemplaza el método previsto por Symfony, adaptándose a las necesidades del proyecto y sin violar el modelo MVC. En la figura 2 se presenta la representación gráfica de la arquitectura antes explicada:

**Figura 2.** Arquitectura de Comunicación Inter-Módulos

## III. ARQUITECTURA DE COMUNICACIÓN INTER-PLATAFORMAS (SYMFONY-JADE)

Para la comunicación entre el *framework* de la aplicación Web (Symfony) y el *framework* del Agente (Jade), se optó por implementar un servicio Web Service, ya que este provee un mecanismo sencillo y flexible para la transmisión de los datos de lado a lado. Este *Web Service* implementa un conjunto de funcionalidades que se complementan para lograr la comunicación interplataforma, como son permitir a Symfony el envío de bloques de datos hacia Jade para que el agente de usuario haga un procesamiento sobre ellos, y posteriormente desde Jade se retome la comunicación con el *Web Service* para enviar el bloque ya procesado.

El *Web Service*, localizado en el mismo servidor del aplicativo, no es accedido por ninguno de los *frameworks* directamente, sino a través de una pequeña interfaz definida para cada uno (un cliente de un *Web Service)*, esto con el fin de evitar introducir código especifico de arquitectura de *Web Services* en el Modulo de Interfaz o en el agente y así respectar el principio de modularidad.

De acuerdo a lo anterior, se puede especificar la siguiente ruta de comunicación:

. *Ruta Modulo Interfaz – Agente*: El modulo se comunica con un *script* (*Web Service Client*), al que le pasa un bloque de datos con la información pertinente. El *script* se encarga de armar el mensaje siguiendo la especificación de *SOAP* y enviarlo al *Web Service*. En

este momento el modulo cierra la conexión con el *script*, ya que mientras esto sucede el usuario final esta esperando una respuesta inmediata por parte del sistema (una interfaz desplegada en el navegador). Una vez el *Web Service* ha recibido el mensaje, lo envía a la plataforma de agentes comunicándose con el *ProxyAgent* (módulo propio de Jade). El *ProxyAgent* posee la capacidad necesaria para discernir que debe hacer con la información que acaba de recibir, como crear el agente (a través de otras herramientas internas de la plataforma) o enviarle un mensaje a un agente ya existente.

. *Ruta Agente-Modulo Interfaz*. El agente en Jade, se comunica con Symfony a través del *ProxyAgent*, enviándole datos que usualmente son para almacenamiento en la Base de Datos. El *ProxyAgent* le envía entonces los datos al *Web Service*, que a su vez se comunica con una acción del Modulo de Interfaz correspondiente.

La explicación anterior permite entrever que en la ruta *Modulo Interfaz – Agente* se encuentra dos tipos de información: Mensajes y Bloques de Datos. Los mensajes van enfocados a la creación/eliminación del agente en la plataforma, una vez que el usuario se ha autenticado o terminado su sesión, y los bloques de datos son la información que constantemente fluye hacia el agente en una sesión activa del usuario. Esta información es usada para alimentar el modelo de usuario y posteriormente es procesada para la adaptación de la interfaz de usuario.

En la ruta *Agente – Modulo Interfaz* se encuentra el flujo de información cuando un escenario particular emerge: El usuario se ha salido el sistema o por tiempo de inactividad el agente decide desactivarse. En este caso, se debe enviar la información que se hasta el momento se haya procesado, para su almacenamiento y posterior carga en una nueva sesión del usuario (si es el caso).

En este punto surge un problema importante y es que, debido a la naturaleza no orientada a conexión del protocolo HTTP, es difícil saber cuando el usuario salió del sistema, en el caso que no haya usado los métodos convencionales (como un enlace de cerrar sesión). Para esto, se estableció que el agente tendrá un *tiempo de duración* (igual al definido en la sesión por cokies en el aplicativo Symfony) para recibir un mensaje antes de decidir terminar su propio proceso (desactivación).

Del mismo modo, surgen nuevamente algunos inconvenientes, relacionados con el envió/recepción de los mensajes de lado y lado. A pesar de que se espera que este proceso sea rápido, puede que no siempre se dé el resultado esperado, por ejemplo, en el caso en que el servidor del aplicativo Web esté muy cargado y no sea posible obtener rapidez en los procesos.

De acuerdo al análisis y pruebas realizadas sobre el sistema, se pudo definir 6 escenarios básicos de interacción Aplicativo Web – Jade. En estos, el agente del usuario se refiere al agente en Jade que se le asigna a cada usuario autenticado:

1. El usuario se dispone a iniciar la sesión y el agente del usuario no existe: el usuario apenas esta iniciando una sesión, por tanto aún no hay un agente en la plataforma que maneja a este usuario. No representa ningún problema para el sistema.
2. El usuario se dispone a iniciar sesión y su agente ya existe: Este el primer caso problemático. La razón por la que ya hay un agente corriendo en la plataforma a pesar de que el usuario no este autenticado, es que previamente haya iniciado sesión en otro navegador (en el mismo u otro computador). Para este escenario la solución consiste en probar, antes de crear un agente si este ya existe (haciendo uso de los agentes facilitadores que provee Jade), y de ser así simplemente seguir procesando información. La única implicación para el usuario, es que en esta nueva sesión abierta, no disfrutara de los cambios en la interfaz (si estos debieran mostrarse).
3. El usuario está en una sesión y el agente de usuario existe: El escenario no presenta problemas, ya que mientras el usuario está navegando, su agente está procesando información dinámicamente.
4. El usuario esta en medio de una sesión y el agente de usuario no existe: Segundo caso problemático. Hay dos razones por las cuales llegamos a este caso crítico. La primera es que el usuario inicio una sesión en otro navegador y luego cerro una de las sesiones abiertas, por lo que el agente fue destruido. La segunda es que justo pasados los $n$ minutos establecidos para que se acaba la sesión por inactividad, el usuario evito que su sesión se cerrara automáticamente, pero el mensaje que indica que el usuario esta activo no le fue enviado a tiempo al agente. La solución para este caso resulta ser sencilla, simplemente se crea un nuevo agente para el usuario. Dado que en condiciones normales de operación, un agente antes de desactivarse manda los datos para ser almacenados, no habría pérdidas con respecto a lo que monitoreo el agente que se desactivó. La implicación para el usuario es que, podría ver antes de tiempo (iniciar una nueva sesión) la interfaz adaptada.
5. El usuario se dispone a terminar la sesión y el agente del usuario existe: es un caso esperado, que como los anteriores no presentan inconvenientes, ya que se supone que cuando el usuario desee salirse del sistema, se debe informar a su agente para que éste realice sus respectivas tareas.
6. El usuario se dispone a terminar la sesión y el agente de

usuario no existe: es similar al caso problemático dos, pero difiere en el hecho que aquí no se toma ninguna acción. Si el agente no existe es porque se cerró la sesión automáticamente.

En la figura 3. se presenta la arquitectura de comunicación entre plataformas:

**Figura 3.** Diagrama de la Arquitectura de Comunicación JADE - Symfony

## IV. CONCLUSIONES Y TRABAJO FUTURO

La arquitectura propuesta contribuyó a la definición de un estándar de trabajo para realizar el proceso de integración de código de cada uno de los grupos de desarrollo definidos en el proyecto. Esto puede ser extensible a cualquier tipo proyecto que involucre desarrollo Web con Symfony; y a la definición de un mecanismo de comunicación para la Plataforma Jade con frameworks externos, usando como elemento base el *SocketProxyAgent*.

Finalmente, es de interés ampliar el potencial del uso de agentes mediante la implementación de agente de interfaz tipo ayudante que guíe al usuario a través de la aplicación, de tal forma que se pueda hacer predicciones y sugerencias sobre lo que el usuario va o puede hacer dentro del sistema. Asimismo, se pretende extender la arquitectura propuesta para permitir la comunicación entre un aplicativo Web desarrollado con las tecnologías Web 2 (Rich Internet Applications - RIA) y el Framework Jade.

## AGRADECIMIENTOS

Los autores extiende su agradecimiento a todo el equipo de trabajo del proyecto Predica (Plataforma Experimental para Sistemas de Recomendación, Descubrimiento de Conocimiento, Interfaces Adaptativas y Consultas Avanzadas), a la Universidad del Valle y a la entidad financiadora – COLCIENCIAS-.

## REFERENCIAS

[1] André Elizaberth and Rist Thomas, "From Adaptive Hypertext to Personalized web companions". Commnunicatios of the ACM. Vol. 45, 2002. pp. 44–45.

[2] Billus Daniel, Brunk Cliefford et all, "Adaptative Interfaces for Ubiquitous Web Access". Commnunicatios of the ACM. Vol. 45, 2002. pp. 34 – 38.

[3] Brusilovsky Peter and Maybury Mark, "Adaptative Interfaces for Ubiquitous Web Access". Commnunicatios of the ACM. Vol. 45, 2002. pp. 31 – 33.

[4] http://www.symfony-project.com Grupos de discusión y documentación en línea de Symfony. Consultada en agosto de 2006.

[5] http://jade.tillab.com Grupos de discussion y documentación en línea de Jade. Consultada en agosto de 2006.

[6] http://rfc.net/rfc2616.html Especificación HTTP/1.1. Consultada en agosto de 2006.

# Universidad Nacional de Colombia Sede Medellín
# Facultad de Minas

120 años
TRABAJO Y RECTITUD

## Escuela de Ingeniería de Sistemas

### Grupos de Investigación

**Grupo de Investigación en Sistemas e Informática**

Categoría A de Excelencia Colciencias
2004 - 2006 y 2000.

**GIDIA: Grupo de Investigación y Desarrollo en Inteligencia Artificial**

Categoría A de Excelencia Colciencias
2006 – 2009.

**Grupo de Ingeniería de Software**

Categoría C Colciencias 2006.

**Grupo de Finanzas Computacionales**

Categoría C Colciencias 2006.

**Centro de Excelencia en Complejidad**

Colciencias 2006

Escuela de Ingeniería de Sistemas
Dirección Postal:
Carrera 80 No. 65 - 223 Bloque M8A
Facultad de Minas. Medellín - Colombia
Tel: (574) 4255350 Fax: (574) 4255365
Email: esistema@unalmed.edu.co
http://pisis.unalmed.edu.co/