Universidad de las Ciencias Informáticas

Facultad 4

Titulo: Desarrollo de un módulo de SIGEP utilizando tecnología Spring WebFlow para la implementación de la interfaz de usuario.

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

Autor(es): Mabel Medina Rodríguez
Yoenia María Martínez Díaz

Tutor(es):
Hayron Corrales Ruiz

La Habana, junio del 2007.

"Para que pueda surgir lo posible es preciso intentar una y otra vez lo imposible."

Hermann Hesse

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al <nombre área> de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Mabel Medina Rodríguez
_____

Yoenia María Martínez Díaz
_____

AGRADECIMIENTOS

Mabel:

A mis abuelos maternos por ser quienes me impulsan a ser mejor y obtener resultados con sólo confiar en mí.

Agradezco a mi mamá, a mi papá, a mi hermana y a Lina por estar siempre para mí como un apoyo incondicional y por quererme por encima de todas las cosas.

A todas mis amistades que como Heidy, Daimara, Martori, Yoenia, Ana Marys y Elvio han reservado una parte de mi cariño.

Yoenia:

Agradezco a Lily por haber asumido mi formación profesional como una meta personal.

A mi familia, por brindarme esa palabra de aliento que siempre fortifica.

A Johanny, por hacer que cada barrera que encontré a lo largo de estos cinco años fuese menos infranqueable.

A Mayle, Mabel, Nolber, Daimara, Maikel y a todos esos amigos que han ocupado un espacio importante en mi vida.

A Orlando, Zequi, Leony, Yayo, Adrián y a ese montón de amigos que tanto se preocupan por mí y a los cuales nunca dejé de extrañar.

A Richard, por existir; por haberme enseñado a no mirar la vida en escala de grises.

Y a todos, todos los que me alentaron a llegar hasta aquí.

Ambas:

A Iosev por la presteza y desinterés con que nos apoyó durante el proceso de implementación del caso de estudio.

A los compañeros del proyecto SIGEP que nos brindaron su colaboración.

A Hayron por jugar un papel importante como tutor de la tesis.

A Ramón por ser el promotor de todo lo que hemos logrado.

A Julio por su apoyo y sinceridad.

A Erwin Vervaet por responder con tanta diligencia y haber evacuado nuestras dudas cuando lo necesitamos.

A nuestro Comandante en Jefe por depositar una confianza sin límites en nuestra superación, a nuestra Revolución y a la UCI por formarnos como informáticos y revolucionarios de primer nivel.

DEDICATORIA

Mabel:

A todos aquellos que como mi mamá, mi papá, mis abuelitos maternos, mi hermana y Lina han estado siempre presentes en mi vida y se han ganado mi cariño y admiración.

Yoenia:

A Lily.

RESUMEN

Un flujo Web puede ser llevado a cabo por varios *frameworks* de interfaz de usuario de la tecnología Java. Es preciso tener en cuenta ciertos factores para decidir qué usar y en qué circunstancias, para lograr resultados óptimos.

El presente trabajo aborda los diferentes *frameworks* de la tecnología Java utilizados por los desarrolladores de interfaces de usuario, las ventajas y facilidades que reporta cada uno de ellos al adaptarlos a una aplicación real así como las dificultades y problemas que dieron lugar al perfeccionamiento continuo a través del surgimiento de tantos otros.

La investigación se enfocó en la plataforma SWF[1] profundizando en sus principales características, sus principios de funcionamiento y su mayor aporte.

Finalmente se aplicará toda la teoría estudiada, a un módulo del proyecto SIGEP[2] como el objeto demostrativo de cuanta eficiencia puede llegar a proporcionar la inclusión, dentro de Spring MVC[3], del *framework* SWF.

[1] Spring WebFlow
[2] Sistema de Gestión Penitenciaria
[3] Model View Controller

TABLA DE CONTENIDO

TABLA DE ILUSTRACIONES

I. INTRODUCCIÓN.

Es una tendencia actual a nivel mundial el uso de Java como lenguaje de programación, según estudios realizados por empresas, basados en análisis de demanda laboral, Java quedó ubicado en el primer lugar de las mayores preferencias de uso de lenguajes en el mercado. (1)

Muchos *frameworks* se adaptan a este lenguaje, cada uno aportando disímiles funcionalidades a las aplicaciones Web de cualquier envergadura.

Desde los primeros cursos, en nuestra universidad, se introduce a los estudiantes en la tecnología de J2EE[4], facilitando así el proceso de migrar de software propietario a software libre. Muchos aplican dichas enseñanzas a proyectos productivos con entidades tanto nacionales como internacionales, usando las plataformas a nuestro alcance con suficiente documentación acerca de sus rasgos y funcionalidades que se adapten a los requerimientos y fines de los contratos.

El proyecto SIGEP[5] se une a esta forma de desarrollo de aplicaciones, pero no sólo basándose en los contenidos recibidos en clases, sino también profundizando en la investigación a través de los medios a nuestro alcance.

Como resultado de esta tarea se propuso el uso de la plataforma Spring MVC para desarrollar el proyecto SIGEP por razones lógicas y consistentes que hacen de este *framework* una opción relevante y factible.

El presente trabajo no contradice dicha selección, sin embargo propone una mejora considerable en escenarios específicos que pueden presentarse en cualquier aplicación Web como son los casos de uso que requieran de varias páginas para completar una única transacción del negocio.

Incluir SWF dentro de Spring MVC pudiera resultar una mezcla eficiente al usar recursos de uno o del otro cuando sea necesario en el proceso de implementación del proyecto. Así como se sugiere su uso conjunto dentro de SIGEP, pudiera extenderse el mismo para otros proyectos de la universidad que requieran de un desarrollo con características similares.

Precisamente nuestro trabajo introduce a los desarrolladores en lo que es SWF con todas sus ventajas y potencialidades, basando en esta teoría la implementación de un módulo que servirá de referencia para aquellos que decidan adentrarse en el mundo de SWF.

[4] Java 2 Enterprise Edition
[5] Sistema de Gestión Penitenciaria.

1.1. Situación problémica

En el proyecto SIGEP se requiere de una tecnología que provea las herramientas necesarias para implementar las interfaces de usuario de la aplicación con técnicas efectivas y de calidad.

No existe en el proyecto SIGEP una cultura general acerca de los *frameworks* de Java para el desarrollo de las interfaces de usuario ya que la necesidad de realización del proyecto y la falta de personal capacitado para orientar sobre el tema, forzaron a los integrantes de SIGEP a prepararse de manera individual y a través del intercambio de conocimientos en sólo uno sin evaluar a fondo las posibles ventajas y facilidades que podría reportar el uso de los restantes o de sus combinaciones.

1.2. Objetivos

1.2.1. Objetivo general

Desarrollar un módulo de SIGEP usando la tecnología de implementación de interfaz de usuario SWF como complemento de Spring MVC.

1.2.2. Objetivos específicos

- Utilizar un *framework* de desarrollo de interfaz de usuario de la tecnología Java.
- Conocer las técnicas más comunes para el uso del *framework* SWF en casos de uso de más de una página en su flujo Web.
- Utilizar SWF como complemento de Spring MVC en el módulo de "Ingreso de un nuevo individuo" del proyecto SIGEP.

1.3. Tareas de la investigación

- Investigación de las características y facilidades de los principales *frameworks* de Java.
- Estudio de los rasgos y características distintivas del *framework* SWF.
- Aplicación de la teoría abordada en el proceso de integración de Spring MVC y SWF a través de la implementación de un caso de estudio.

1.4. Marco teórico Conceptual

Desde el surgimiento de la plataforma Java en el año 1995, antes de la existencia de los Servlets Java, los desarrolladores tenían que realizar una serie de procesos complicados y primitivos para implementar una aplicación Web: escribían infinitas sentencias *println ()* para enviar código HTML al navegador, lo cual resultaba problemático y moroso; sin embargo después del descubrimiento de los Servlets, muchos programadores comenzaron rápidamente a notar su utilidad: eran rápidos, potentes, portables, y extensibles, y por tanto, podrían aprovechar sus funcionalidades para simplificar su labor.

Pero en realidad la solución a los problemas existentes fueron las Java Server Pages, que permitían incluso escribir Servlets dentro de ellas, de tal manera que pudieran mezclar sin ninguna dificultad código HTML con código Java, y mantener además todas las ventajas de los Servlets. Era de esperar que con un descubrimiento de esta envergadura, las aplicaciones Web Java se convirtieran rápidamente en "centradas-en-JSP". Esto, por sí sólo, ya era una mejoría, pero aun así, las JSP hacían poco por resolver los problemas de control de flujo y otros problemas característicos de las aplicaciones Web, por lo que se hizo inminente idear un nuevo modelo.

Muchos desarrolladores inteligentes de la época notaron que las Java Server Pages y los Servlets se podrían usar juntos para desplegar aplicaciones *Web*, de manera que los Servlets se encargaran del control de flujo, y las JPSs podrían encargarse de escribir el código HTML. (2)

El término *Model2* fue utilizado por la Sun Microsystem, creadora de Java, para referirse al modelo arquitectural recomendado para las aplicaciones *Web* desarrolladas sobre J2EE.

Dicha arquitectura consiste en el desarrollo de una aplicación según el patrón de diseño *Model-View-Controller (MVC)* (**Figura 1**), pero especificando que el controlador debe estar formado por un único *servlet*, que centralice el control de todas las peticiones al sistema, y que basándose en la URL de la petición HTTP y en el estado actual del sistema, delegue las diferentes gestiones y el control de la petición a una determinada acción de entre las registradas en la capa controladora. Esta centralización del controlador en un único punto de acceso se conoce como patrón *front controller*.

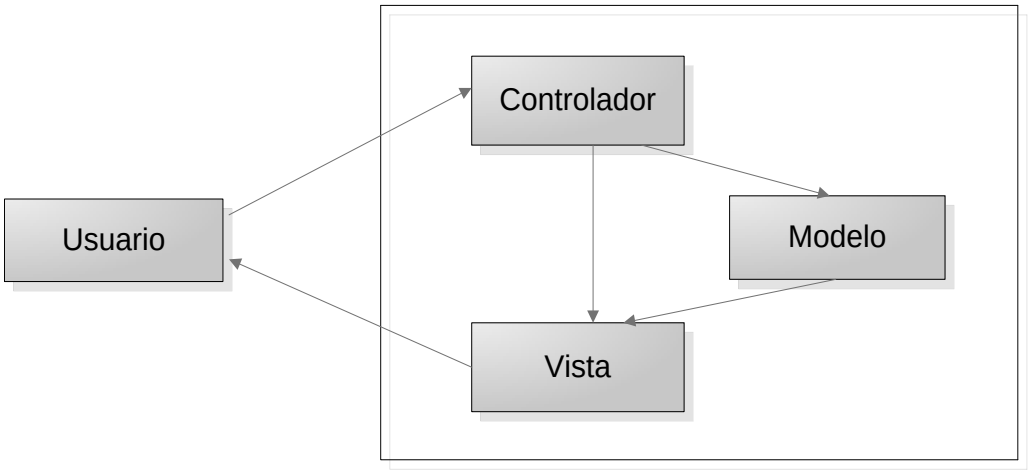

Figura 1 Patrón MVC.

Las ventajas que este patrón ofrece provienen de la capacidad de gestionar en un único punto la aplicación de filtros a las peticiones, las comprobaciones de seguridad, etc.

El concepto de MVC se basó fundamentalmente en separar el modelo de datos de la aplicación de su representación de cara al usuario y de la interacción de éste con la aplicación mediante la división de la aplicación en tres partes fundamentales: el modelo, que contiene la lógica de negocio de la aplicación; la vista, que muestra al usuario la información que éste necesita; el controlador, que recibe e interpreta la interacción del usuario, actuando sobre modelo y vista de manera adecuada para provocar cambios de estado en la representación interna de los datos, así como en su visualización.(**Figura 2**)

Por esta razón, en la actualidad es muy común usar los términos *Model2* y *MVC* indistintamente. (3)

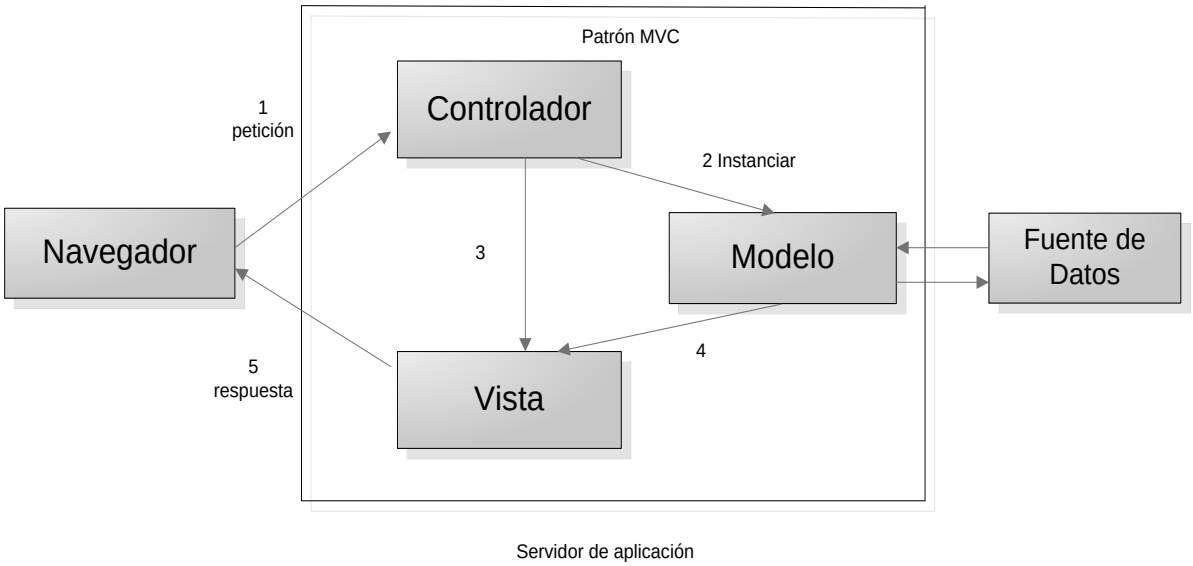

Figura 2 Interacción entre el modelo, la vista y el controlador en el patrón MVC.

Aunque algunos piensan que es mucho mas práctico no usar ningún *framework* e implementar nuestras propias funcionalidades con el objetivo de que se adapten mejor a nuestras necesidades y a las características de la aplicación que se esté desarrollando, la posibilidad de crear componentes reutilizables e integrables con otro entorno de programación además de organizar y hacer controlable el flujo de una aplicación, sigue siendo una opción muy atractiva y sensata. La utilización de un *framework* tiene objetivos especificos:

- Acelerar el proceso de desarrollo de software.
- Reutilizar código existente.
- Promover buenas prácticas de desarrollo como es el uso de patrones. (4)

Por esta razón utilizar *frameworks* es la estrategia mas comúnmente usada entre los desarrolladores, principalmente cuando se trata de una aplicación con algún nivel de complejidad.

1.4.1. Struts

En Mayo del 2000, Craig R. McClanahan lanzó el proyecto Struts para proporcionar un *framework* MVC estándard a la comunidad Java, cuya primera versión se liberó en julio del 2001. Struts es una herramienta de soporte para el desarrollo de aplicaciones Web que comenzó desarrollándose como parte del proyecto Jakarta de la Apache Software Fundation, pero actualmente es un proyecto independiente conocido como Apache Struts, que proporciona en la soporte a la creación de las capas vista y controlador de aplicaciones Web basadas arquitectura Model2. Está basado en tecnologías estándar como *Servlets*, *JavaBeans* y *XML*.

Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en que Java Enterprise este disponible, lo convierte en una herramienta altamente accesible. (2)

Struts se basa en el patrón del Modelo Vista Controlador (MVC), el cual se utiliza ampliamente y es considerado de gran solidez. Implementa un sólo controlador (*ActionServlet*) que evalúa las peticiones del usuario mediante un archivo configurable (*struts-config.xml*) el cual se integra con una vista realizada con páginas JSP, incluyendo JSTL y Java Server Faces, entre otros (5). **(Figura 3)**

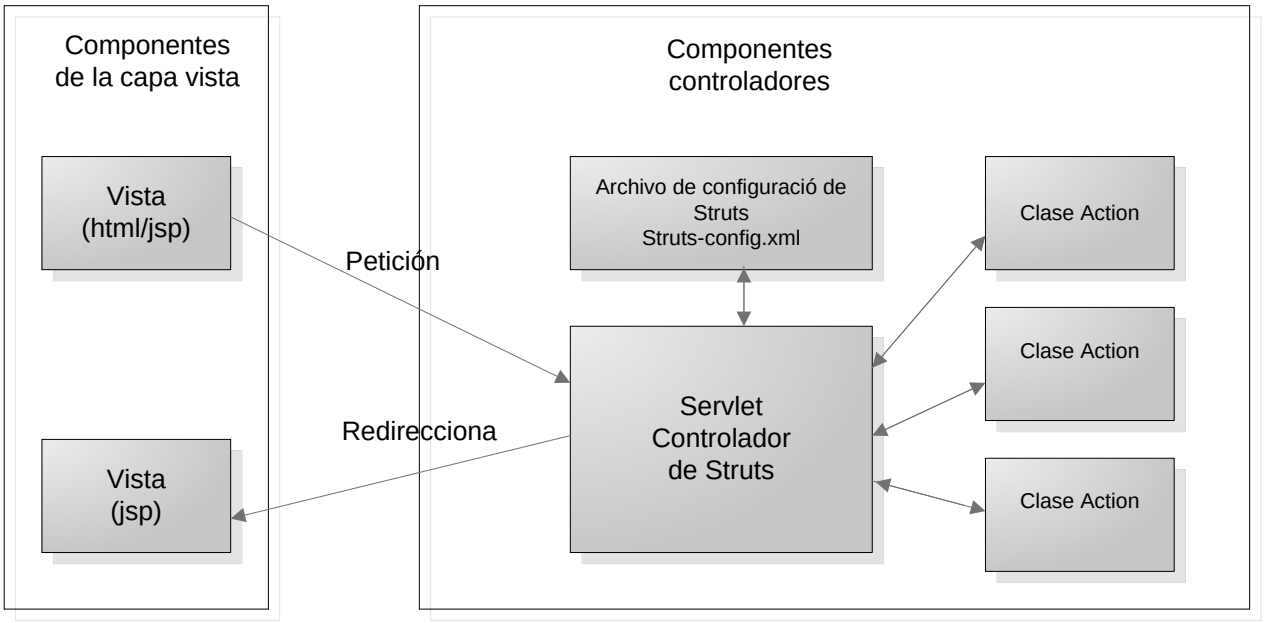

Figura 3 Arquitectura del Framework STRUTS.

En cuanto a la capa vista, Struts permite utilizar entre otras tecnologías páginas JSP para la realización de la interfaz. Para facilitar las tareas comunes en la creación de esta capa existen una serie de tecnologías que se integran con Struts: Taglib, JSTL, Tiles, StrutsValidator, Struts Menú, CSS, etc.

Pero Struts sólo implementa un controlador que extiende de la clase Action, si se necesita algo más simple, no se tiene, si por el contrario se requiere de algo más poderoso, es menester heredar de la clase Action e implementar la lógica que requiera nuestra aplicación. Además los *beans* [6]de formulario de Struts son clases que heredan de ActionForm, con los cuales el proceso de pruebas es algo complejo. Este *framework* tiene sus dificultades con el mantenimiento de los

[6] Programa de aplicación que tiene funciones específicas

archivos de configuración, a la hora de integrarlo con otro *framework* y al trabajar en un ambiente de desarrollo colaborativo. (6)

1.4.2. Stripes

Stripes fue creado por Tim Fennell; supera algunas de las desventajas de Struts antes mencionadas, a la vez que mantiene sus ventajas. Sin embargo para aprender Stripes con facilidad se debe estar familiarizado con Struts, pues de no ser así, el aprendizaje de dicho *framework* sería complicado. (7)

1.4.3. Spring

Spring también viene a resolver las limitaciones mencionadas, surgido en el 2003**,** se considera un *framework* que ha revolucionado la manera de programar aplicaciones Java debido a la facilidad que imprime al proceso de crear componentes reutilizables, además de que se adapta fácilmente con otros *frameworks* como lo son Hibernate, Struts, etc.

Spring es utilizado para aplicaciones de empresa. Ofrece varias opciones para la persistencia de datos y puede ser usado de forma modular. O sea, permite su uso en partes y dejar a un lado los otros componentes que no son requeridos por la aplicación.

Provee una capa de abstracción genérica para la gestión de transacciones. La capa de abstracción JDBC[7] de Spring ofrece una significativa jerarquía de excepciones, que simplifica la estrategia de manipulación de errores.

Provee una mejor integración con Hibernate que los *frameworks* surgidos hasta esa fecha. Constituye el mejor *framework* de Programación Orientada a Aspectos.

Tiene una muy bien organizada arquitectura consistente en 7 módulos.

[7] Java Data Base Conectivity

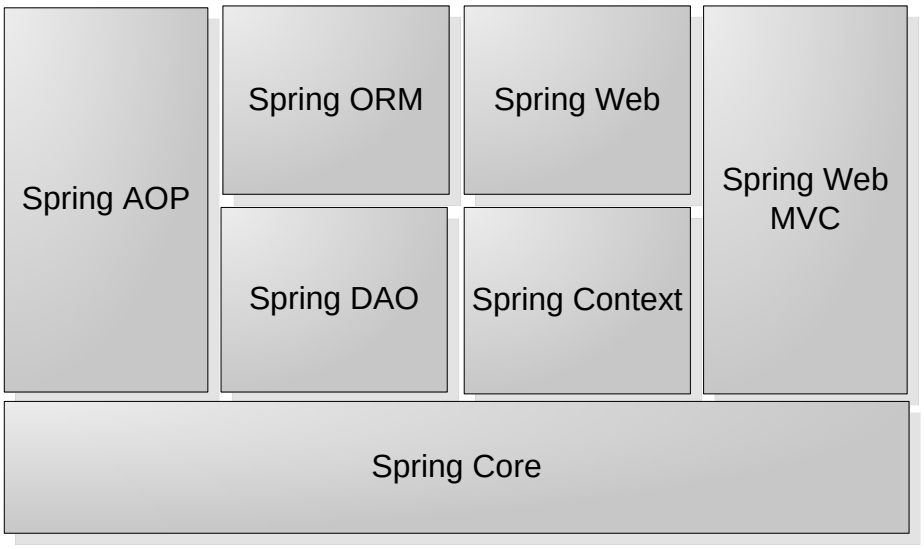

Figura 4 Arquitectura del framework Spring.

Spring proporciona una potente gestión de configuración basada en *JavaBeans*, aplicando los principios de *Inversión de Control* (IoC). Esto hace que la configuración de aplicaciones sea rápida y sencilla.

Proporciona un *framework MVC (Model-View-Controller*), construido sobre el núcleo de Spring. Este *framework* es altamente configurable vía interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser JSP, Tiles, iText o POI. De cualquier manera una capa de negocio realizada con Spring puede ser fácilmente utilizada con una capa Web basada en cualquier otro *framework* MVC, como Struts. El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos. Estos objetos pueden ser reutilizados tanto en entornos J2EE (Web o EJB), aplicaciones standalone, entornos de pruebas, etc. sin ningún problema.

La arquitectura en capas de Spring ofrece mucha flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar la gestión de configuración basada en JavaBeans sin utilizar el *framework MVC* o el soporte *AOP*. Spring tiene una rica selección de implementaciones de controladores que van desde el más simple (la interfaz *Controller*) hasta el más complejo y poderoso (*AbstractWizardFormController*) además de todos los que están comprendidos entre ellos en cuanto a complejidad se refiere. Sólo hay que elegir la implementación del controlador que más se ajuste a nuestras necesidades. Algunos consideran una desventaja de Spring MVC, esa amplia selección de controladores con que cuenta, podría ser un poco más complicado saber cual controlador es más conveniente usar. (6)

Por otro lado es muy sencillo realizar pruebas a los controladores de Spring, puesto que los objetos de respaldo de los formularios son cualquier cosa que herede de *Java.lang.Object*, a los cuales es muy sencillo realizarles pruebas. Otro factor a considerar es la incomodidad de algunos programadores con la longitud de los nombres de las clases de Spring.

Aunque existe una diferencia en lo que a cantidad de tiempo de creado y documentación disponible para el aprendizaje se refiere, Spring MVC, vigente desde el 2003, va tornándose más necesario y eficiente que Struts entre los desarrolladores de Java.

1.4.4. Wicket

Con Wicket**,** creado por Eelco Hillenius, el programador tiene el control de cómo se crean los componentes y de como se integran, además ofrece la posibilidad de usar directamente POO[8] en la capa de presentación.

Los componentes son auto-controlados, fáciles de crear y de usar y además son reusables. También usa MVC. Posee modelo de plantillas HTML fácilmente editables por cualquier editor de HTML. La gestión de contenido dinámico y manejo de formularios se realiza en Java con un modelo de objetos gráficos acompañado por un modelo de objetos POJO de datos. No son necesarios archivos XML de configuración del *framework*.

Wicket no se aconseja para aplicaciones de interfaz de usuario simples, en ese caso sería más aconsejable trabajar sólo con *scripts* y dejar la puerta abierta para usar JSF cuando sea necesario.

1.4.5. Swing

Swing es un paquete Java para la generación del GUI[9] en aplicaciones reales de gran tamaño. Utiliza componentes ligeros, y arquitectura Model-View-Controller (MVC). Proporciona nuevos componentes (árboles, tablas, marcos internos, etc.) así como utilidades para facilitar creación de aplicaciones gráficas.

Sin embargo estos componentes resultan ser muy pesados, por tanto presentan funcionamiento diferente en cada plataforma, la selección de diferentes apariencias que proporciona está ligada a la plataforma, no puede cambiarse. Estos componentes consumen más recursos (CPU, memoria).

1.4.6. Tapestry

En el 2004 surge Tapestry**,** creado por Howard Lewis Ship, una plataforma Open-Source que facilita la creación de aplicaciones Web dinámicas, robustas y altamente escalables, en Java. Complementa y construye el estándar Java Servlet API, por lo cual funciona en cualquier contendor de *servlet* o servidor de aplicación. Divide una aplicación Web en un conjunto de

[8] Programación Orientada a Objetos
[9] Interfaz Gráfica de Usuario

páginas, cada una construida desde componentes, lo cual proporciona una estructura consistente que permite que dicho *framework* asuma la responsabilidad por intereses claves como son la construcción y envío de URL, almacenamiento de estado persistente en el cliente o en el servidor, validaciones de entradas de usuario y el reporte de errores.

Se integra fácilmente con J2EE, Spring, etc.… Facilita el desarrollo orientado a objetos en las aplicaciones Web Java. La creación de este *framework* estuvo dirigida a lograr requerimientos básicos y claves de una aplicación Web como son la simplicidad, consistencia, eficiencia y retroalimentación.

Sin embargo la plantilla de Tapestry es un documento HTML con partes reemplazables, por tanto no es posible mezclar HTML con código Java. Además la documentación de este *framework* no está actualizada y no existe suficiente para conformar un conocimiento robusto y específico en el tema. (9)

1.4.7. Java Server Faces (JSF)

También en el 2004, salió a la luz Java Server Faces (JSF), gracias a Jacob Hookom. Es un *framework* de componentes de interfaces de usuario del lado del servidor para aplicaciones Web basadas en Java. Contiene un API para representar componentes de UI[10] y manejar sus estados, sus eventos, la validación del lado del servidor, y la conversión de datos, definir la navegación entre páginas, soportar internacionalización y accesibilidad; y proporcionar extensibilidad para todas estas características. También contiene dos librerías de etiquetas JSP (Java Server Pages) personalizadas para expresar componentes UI dentro de una página JSP y para conectar componentes a objetos del lado del servidor.

JSF encaja bien en la arquitectura de la capa de presentación basada en MVC. Ofrece una clara separación entre el comportamiento y la presentación. Une los componentes UI con los conceptos de la capa-Web sin limitarnos a una tecnología de *script* o lenguaje de marcas particular.

Los componentes UI de JSF son elementos configurables y reutilizables que componen las interfaces de usuario de aplicaciones JSF. Se puede extender un componente UI estándar y desarrollar componentes más complejos, como barras de menú y árboles. Basados en los convertidores y validadores estándar, se pueden desarrollar convertidores y validadores personalizados, que proporcionan un mejor modelo de protección.

A pesar de su potencia, el *framework* JSF no esta maduro aún. Los componentes, convertidores y validadores que vienen con JSF son básicos. Y el modelo de validación por componente no puede manejar validaciones muchos-a-muchos entre componentes y validadores. Además, las etiquetas

[10] Interfaz de Usuario

personalizadas de JSF no se pueden integrar con JSTL (*JSP Standard Tag Library*) simultáneamente. La implementación de referencia de JSF de Sun no soporta subida de ficheros, lo que constituye un impedimento significativo para usar este *framework*.

1.4.8. WebWork

También en ese año, Patrick Lightbody crea WebWork; un framework Web que implementa el patrón MVC y que ha sido diseñado para la creación rápida y sencilla de sitios Web dinámicos utilizando el mínimo esfuerzo y con una flexibilidad máxima. Soporta múltiples vistas, permitiendo el uso de HTML, PDF, etc., todo ello sin cambiar para nada la lógica o los datos. Además de eso ofrece un potente lenguaje propio y una completa librería de etiquetas. Soporta internacionalización, redireccionamiento, control de errores, validación, acciones basadas en el patrón *Command* y conversión de tipos. La tecnología de vista provee soporte para los reportes.
Es activamente soportado por la comunidad pero no entrega el desarrollo Web con estilos de componentes. Su arquitectura sigue la arquitectura común de la mayoría de los *frameworks* de aplicación Web del Modelo 2, o sea que incluye un controlador central el cual maneja las peticiones. Pese a que toma experiencia de los *frameworks* pasados y corrige varios errores, carece de una variedad de controladores que enriquecen el trabajo de los desarrolladores y que se enfocan en situaciones específicas.

1.4.9. Apache Cocoon

Otros *frameworks* han surgido como producto de la inconformidad de algunos con las limitaciones de los ya existentes. La creación de Apache Cocoon, de publicación XML, ofrece un ambiente flexible basado en la separación entre contenido, lógica y estilo y está basado en los componentes de desarrollo Web, especializándose cada uno en la ejecución de una operación en particular. Un sistema de configuración centralizada y sofisticado provee las herramientas para crear, desplegar y mantener una sólida aplicación. Interactúa con la mayoría de las fuentes de datos incluyendo sistemas de ficheros, RDBMS[11] y bases de datos XML nativas. Adapta el contenido de entrega a diversos formatos como HTML, PDF, RTF, etc.…, transfiriendo elegantemente la información de una representación a otra. Actualmente corre como un *servlet.* Ofrece brillantes conceptos pero es muy complejo para trabajar dentro de un equipo ya que resulta difícil su aprendizaje. (10)

1.4.10. Maverick

También surge Maverick, un *framework* que se enfoca solamente en el patrón de desarrollo MVC (Model 2) para la publicación Web usando Java. Combina los mejores rasgos de Struts, WebWork

[11] Paquete de programas que contiene utilidades que facilitan la administración y el acceso a bases de datos relacionales.

y Cocoon2. Es bien simple de usar, pero no provee las herramientas necesarias para construir una aplicación Web aunque presenta algunas funcionalidades como servidor de aplicaciones, *frameworks* de validación, lenguajes de plantillas, etc. Con respecto a la tecnología de vistas se torna complicado.

Y muchos son los *frameworks* Web sobre el lenguaje Java que han usado los programadores en estos años, se puede mencionar algunos como Barracuda, Turbine, SOFIA, Verge, JPublish, Jucas, Chrisalys, VRaptor, Millstone, DWR, JAT, JOSSO, etc.

1.4.11. SWF

Pero aún no se había encontrado la manera de capturar y controlar el flujo Web. Hasta que en el otoño del 2005 surge SWF creado por Keith Donald. SWF es un módulo de *Spring Framework* que apenas esta emergiendo. Dicho módulo es parte de la suite de desarrollo de aplicaciones Web de Spring, la cual incluye Spring MVC.

Este *framework* constituye actualmente la mejor solución para el manejo del flujo de página de aplicaciones Web. Es un controlador poderoso a usar cuando una aplicación requiere de una navegación múltiple y compleja para guiar al usuario a través de una serie de pasos en una transacción de aplicación.

Spring, desde el 2005, incorpora oficialmente SWF, plataforma Web de alto nivel, muy innovadora para la creación y configuración de flujos Web reutilizables dentro de una aplicación Web.

Aquellos que están familiarizados con Spring podrían erróneamente asumir que SWF es lo mismo que Spring MVC. No lo es. Spring MVC es un *framework* ligero para el desarrollo de aplicaciones Web, mientras que SWF introduce una propuesta completamente nueva.

Spring es un *framework* inmensamente popular y actualmente el líder para desarrollo de aplicaciones en Java; SWF hereda el poder de Spring para adicionarle una innovadora funcionalidad. Con ayuda de SWF los desarrolladores tienen la posibilidad de crear módulos auto-controlados y reusables llamados flujos para guiar al usuario a través de una secuencia organizada de páginas.

Inspirado por requerimientos actuales de los usuarios y refinado a través de una rigurosa retroalimentación del cliente, SWF es el *framework* para aplicaciones Web de próxima generación. Fue sometido a un estricto programa de aseguramiento de calidad, por lo que el producto ha sido utilizado en la producción en aplicaciones de misión crucial, incluyendo la Oficina de Patentes Europea, donde SWF juega un papel crucial en la Gestión de Propiedad Intelectual a través de la Unión Europea.

Keith Donald, líder técnico del proyecto SWF, ha puesto muchas esperanzas en este proyecto, plantea que las aplicaciones Web realizadas con Spring WebFlow ofrecen numerosos beneficios para los usuarios finales; por ejemplo, pueden pulsar el botón atrás del navegador libremente sin que se muestre la temida pagina de “Recargar”, los usuarios pueden ser guiados a través de una secuencia de tareas con facilidad, y al mismo tiempo cobra un gran valor para los desarrolladores incrementando la productividad, calidad y facilidad para realizar pruebas en el proceso de desarrollo. (12)

Algunos de los programadores que han adoptado SWF plantean que este *framework* direcciona uno de los aspectos de la programación Web que más tiempo consume en el desarrollo, pues usualmente el código que controla el flujo de página está disperso en toda la aplicación en todas y cada una de las paginas, por lo que el proceso de extraer toda la lógica del flujo en un sólo lugar impulsa la productividad y la facilidad de realizar pruebas en una gran. (12)

Según las referencias y la documentación consultada se considera que SWF, en poco tiempo ha logrado mucho, pues se ha convertido en una opción a valorar cuando se implementa una aplicación Web. No ha madurado como otras plataformas por una cuestión de tiempo ya que data de sólo 2 años, lo que no impide que sea estable y que se supere constantemente hasta el punto de ser usado por empresas virtuales con resultados satisfactorios.

1.5. Conclusiones

Controlar la transición de estados es una gran ventaja sobre los *frameworks* MVC clásicos. Muchas otras son las funcionalidades que ofrece el *framework* SWF, que lo convierten en el candidato perfecto para ser utilizado en aplicaciones Web complejas y que consten de una secuencia organizada de vistas.

Realizar dicho proceso utilizando cualquiera de los *frameworks* MVC clásicos, daría paso a una solución funcional pero no reutilizable y por supuesto, mucho menos elegante y consistente. En el caso de SWF se podría decir que está diseñado básicamente para eso.

Existen razones para pensar que la utilización de dicho *framework* en la aplicación simplificará enormemente nuestro trabajo, principalmente a la hora de implementar un *wizard*[12], lo cual será muy común en nuestro proceso de desarrollo; realizando este proceso a través de la captura del flujo de página que generalmente usará la misma técnica consecuente, se tendría esa parte de la aplicación lista para ser reutilizada en diferentes situaciones con mínimos cambios. Véase en el próximo capítulo más detalladamente las características de este *framework*.

[12] Secuencia de páginas que guían al usuario a través de un proceso.

II. SPRING WEBFLOW.

2.1. Entorno de desarrollo

La especificación de los *servlets* de Java define tres ámbitos diferentes, la petición, la sesión y la aplicación, que de acuerdo a sus características de alcance estipulan la visibilidad y ubicación de los objetos y atributos asociados con una solicitud HTTP. El de menor tiempo de vida es la petición, contiene todos los datos enviados del explorador y es descartado cuando la vista retorna al buscador. La sesión implementa sus funciones tan pronto como cada explorador accede al servidor, almacena los objetos que permanecen activos durante varias peticiones, por ejemplo los datos de autenticación de un usuario y culmina cuando se destruye explícitamente. El de aplicación se mantiene con vida durante todo el despliegue del proyecto Web puesto que contiene los componentes compartidos de la aplicación y los elementos de configuración. (13)

Los ámbitos anteriormente explicados se usan para resolver disímiles situaciones en aplicaciones Web exceptuando el escenario de casos de uso que requieran de varias páginas para culminar una transacción. Es común usar la sesión para almacenar los recursos de una secuencia de páginas que conforman un flujo Web, esto puede traer una serie de afectaciones que inciden negativamente en el funcionamiento de la aplicación.

La afinidad al servidor es una dificultad que pudiese presentarse en casos de este tipo, cada sesión es única para cada navegador que se conecte a un servidor determinado, si el servidor cambia, todos los recursos asociados a la sesión son eliminados automáticamente pues se crea una nueva sesión que responde al cambio de servidor. El incremento del uso de memoria al almacenar los datos en la sesión también es un factor negativo pues tiende a retardar la aplicación debido a ciertos recursos fuera de fecha que permanezcan almacenados en las sesión sin necesidad de uso. En situaciones donde existan múltiples ventanas en un navegador o etiquetas no es eficiente almacenar datos en la sesión ya que hay altas probabilidades de existencia de colisiones entre espacios de nombres, por ejemplo, en una ventana del navegador guardar un elemento determinado con su identificador, y en otra ventana, usar el mismo identificador para almacenar otro dato diferente al anterior, la primera información se pierde. Esto se debe a la presencia de una sola sesión compartida por todas las ventanas en el navegador. (13)

Esto evidencia la necesidad de un ámbito no tan extenso como la sesión, ni tan breve como la petición, que trate estas dificultades que se pueden presentar en cualquier aplicación Web. Se habla de un espacio que maneje una conversación lógica entre el cliente y el servidor, que conste

de un tiempo de vida que coincida con una transacción determinada, que ninguna interfiera con los recursos de las demás y que contemple múltiples peticiones al servidor. Los elementos almacenados en dichos ámbitos, independientes unos de otros en una misma sesión de un navegador, nunca van a estar fuera de fecha y no precisamente por manejo manual para eliminarlos, una vez terminada la ejecución de un flujo los datos de la conversación son eliminados. (13)

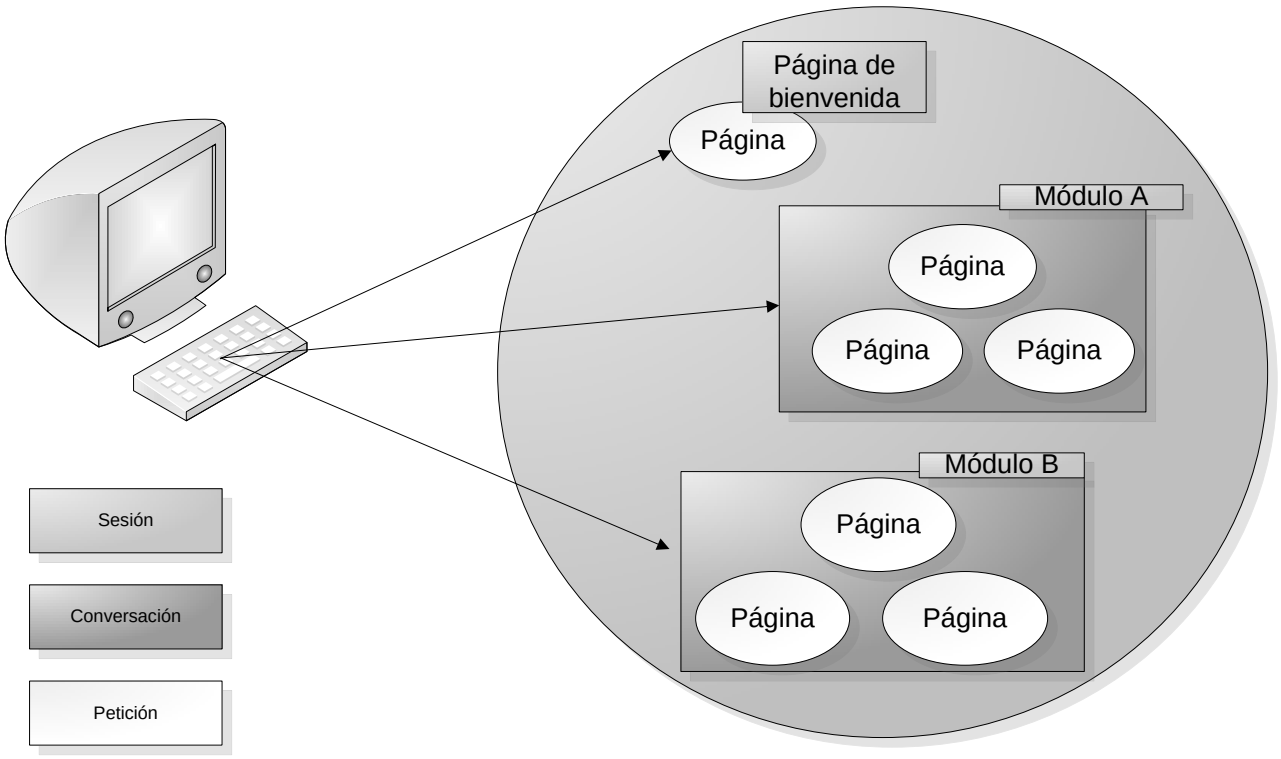

Figura 5 Ejemplos de casos de uso y ámbitos correspondientes.

2.2. Flujos Web

El flujo de páginas de una aplicación Web consiste en la secuencia de páginas por las que pasa una aplicación en función de la conversación que mantenga con el usuario. Dependiendo de las opciones que escoja el usuario, resultados de las operaciones de proceso…etc., la aplicación seguirá una ruta de páginas especifica u otra.

Tradicionalmente, definir y entender un flujo de páginas en una aplicación Web es difícil y a veces incomprensible. Por ejemplo, Struts fuerza a los desarrolladores a cortar el flujo de páginas en controladores (acciones) individuales y vistas. O sea, para poder construir un flujo de varias páginas usando este *framework* se necesitan encadenar acciones individuales a través de las distintas vistas, además de escribir forzosamente las URLs de acciones en cada vista, así como también se debe de emplear un método apropiado de manejo de sesión para poder controlar el estado del flujo. El controlador requerido procesará la petición, selecciona una vista y la envía para mostrarla. Este proceso petición-acción-vista continúa.

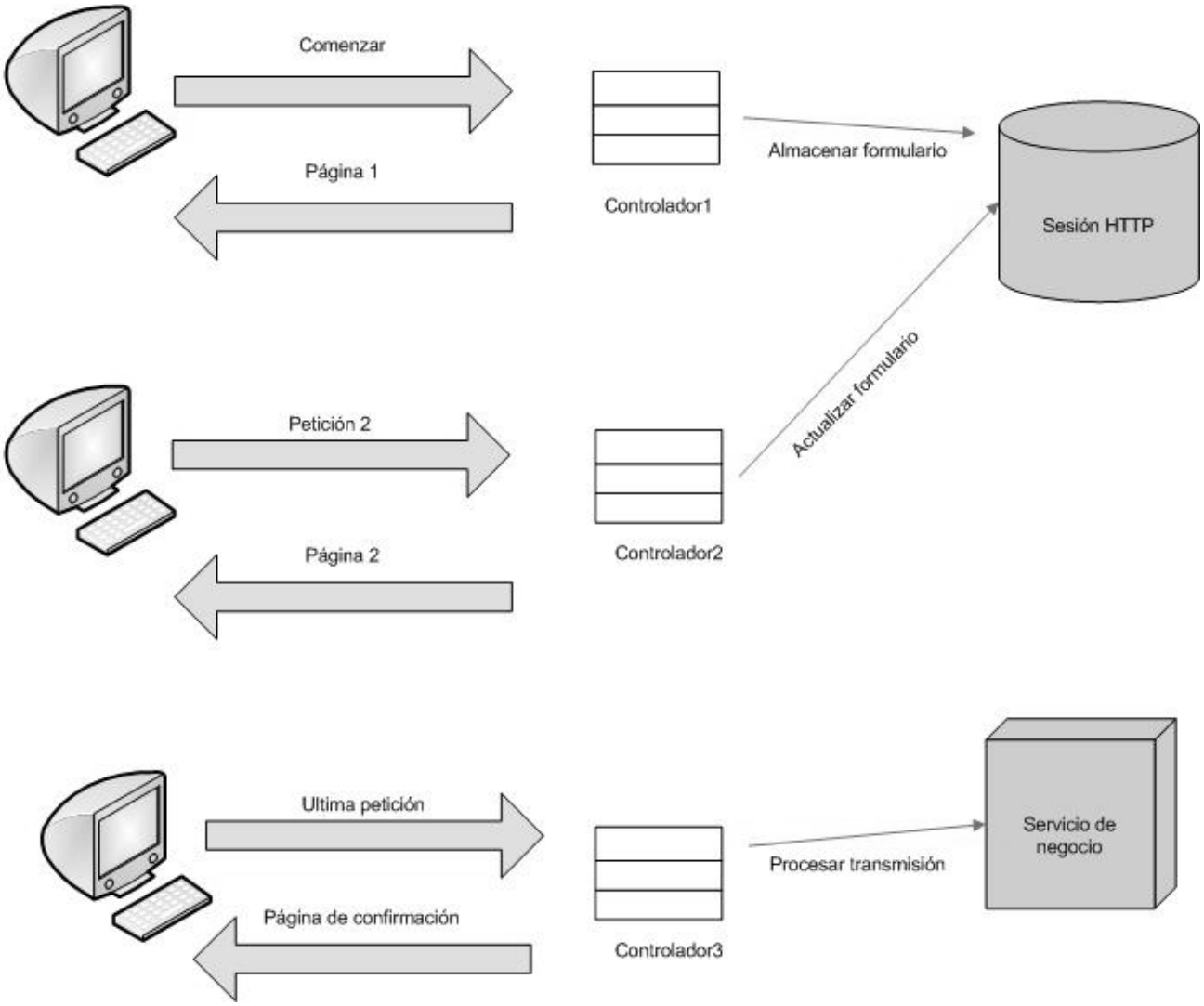

Figura 6 Flujo Web utilizando Struts. (14)

Todo esto es simple y funcional pero reporta varias desventajas en aplicaciones Web con flujos de página relativamente complejos como por ejemplo la falta de claridad de una aplicación Web completa vista desde las definiciones individuales de los controladores, o sea, se requiere de un proceso de traceado a través del archivo de configuración de Struts para averiguar como las partes se relacionan entre sí. Se ve afectada también la flexibilidad y capacidad de mantenimiento ya que las acciones individuales no pueden ser fácilmente rehusadas en diferentes contextos y es muy difícil rastrear el ciclo de vida de una ejecución lógica porque se encuentra dispersa por toda la aplicación.

En el caso del *framework* Spring este provee un poderoso controlador que hace simple el manejo de un flujo Web dentro de un caso de uso, el `AbstractWizardFormController`, que controla

todo el flujo por debajo de la cubierta, o sea, no hay forma de saber la disposición de las páginas o en qué orden aparecen hasta configurar el controlador que hereda de este último en el archivo de configuración XML, para de esta forma, darle a conocer al *wizard*, la lista lógica de vistas a procesar cuyo orden determina la secuencia de aparición de las mismas, pudiendo ser esto alterado mediante la modificación del método que resuelve esta ejecución en el controlador que hereda del `AbstractWizardFormController`.

Usando un controlador de este tipo se crea una dependencia de las peticiones URL y de sus respectivos parámetros ya que para cada cambio de página se requiere de una petición HTTP. La primera llamada accede al controlador y almacena en la sesión los recursos para posteriormente, a medida que continúen las peticiones al mismo controlador, proceder a actualizar los datos en sesión hasta llegar al final del flujo donde se debe realizar la lógica correspondiente a la transacción.

Se usa solamente para cuando se está manipulando un único bean de comando a través de varias páginas lo que le impide soportar más de un flujo de trabajo multipágina, tampoco está diseñado para soportar una decisión de bifurcación arbitraria y de ninguna manera puede manejar una máquina de estado genérico. Los cambios de estado en este controlador se traducen en llamadas a los diferentes métodos del controlador para cambiar de página, cancelar o finalizar un flujo (**Figura 7**). (14)

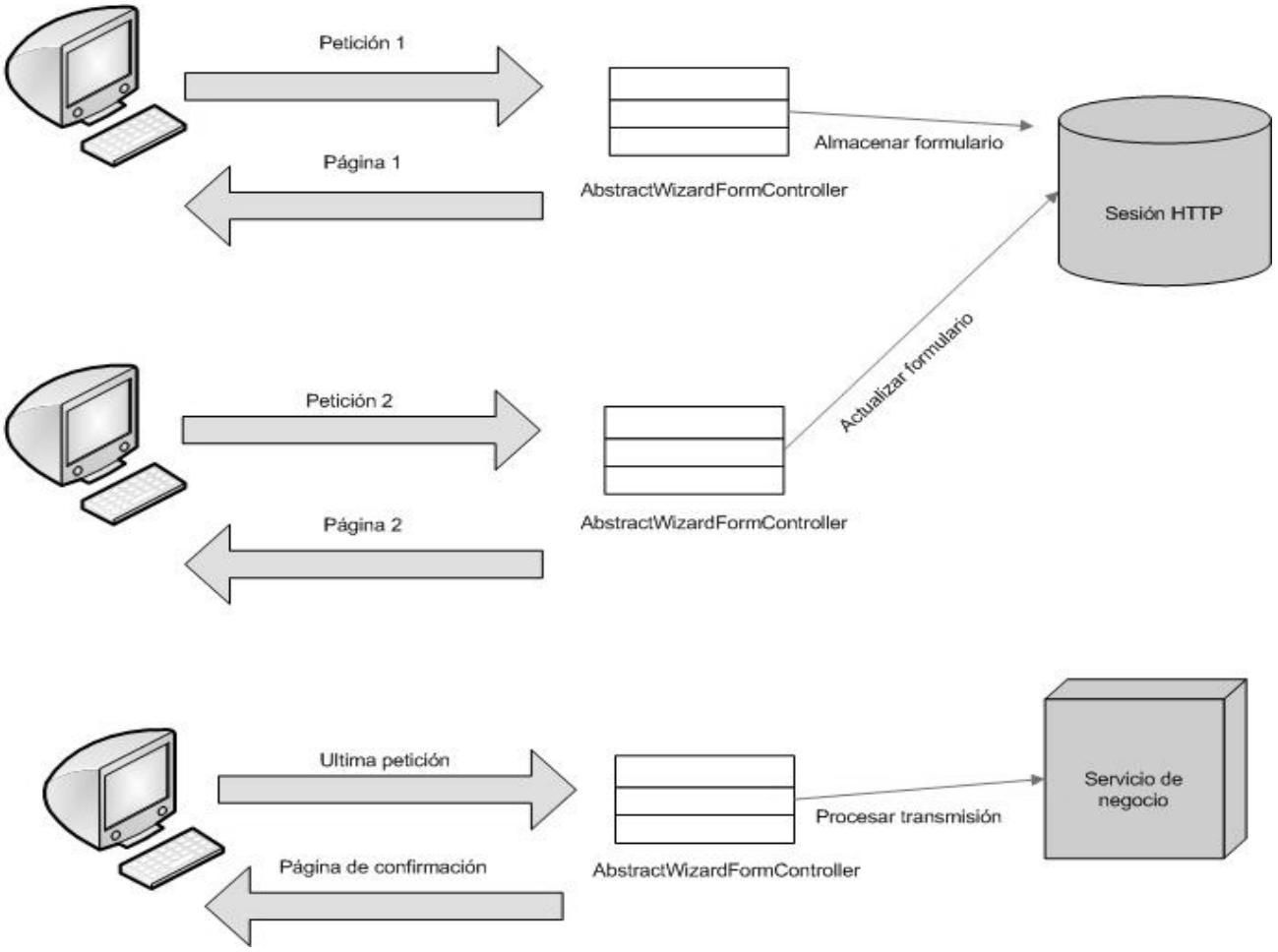

Figura 8 Flujo Web utilizando Spring MVC.

2.3. SWF

SWF[13] surge como una respuesta a la funcionalidad limitada del flujo de página ofrecida por los *frameworks* MVC clásicos. Provee ese espacio que requieren módulos independientes de una misma aplicación para almacenar los recursos de una ejecución de flujo en caso de tenerla.

Los flujos Web en este *framework* son diseñados para ser auto-controlados dando la posibilidad de definir reglas de navegación y centralizando toda la lógica controladora para una tarea del usuario final en un módulo o definición de flujo.

Se implementa la integración con Struts Clásico, Spring MVC y JSF para que sea sencillo ejecutar flujos en esos entornos, pero en todos esos casos, el sistema de WebFlow y las definiciones de flujo permanecen auto-controladas. (15) Esto trae muchos beneficios: el encapsulamiento y la facilidad de realización de pruebas constituyen los más importantes. Este último es también debido a que SWF elimina el acoplamiento entre componentes, colaboradores y el ambiente de despliegue lo que facilita la realización de pruebas en aislamiento.

Debido a su alto grado de integración con Spring MVC incluye el *BeanFactory* de Spring para la configuración, los conceptos de *DataBinder* y *Validator*, el *DispatcherServlet* y los subsistemas de resoluciones de vistas además de ser reutilizable en diferentes contextos. Al usar SWF en una aplicación ya creada con una tecnología determinada, no requiere ningún tipo de exclusividad y se integra fácilmente como un controlador más con técnicas consistentes como la de anidar flujos que actúan como pequeños módulos de la aplicación en la caja de herramientas y en la arquitectura de Spring MVC.

Su modelo de programación lineal ofrece una elegancia particular, ya que al tener una tarea que se ejecuta a través de múltiples pasos, los mismos, manejados con SWF, mantienen el orden predefinido.

Una definición de flujo puede ser modificada en medio de la ejecución del flujo correspondiente y ver los cambios efectuados sin reiniciar el contenedor de flujo lo que no sucede con otros controladores que tratan los flujos Web de una aplicación.

Introduce el concepto de continuaciones que se tratará más adelante pero que posibilita un ininterrumpido uso del botón atrás del navegador sin advertencias o mensajes desagradables muy comunes en otras situaciones como por ejemplo un flujo desarrollado con Spring MVC. Esto resulta idóneo para cuando el resultado de pasos futuros depende de lo que ha sido completado en pasos anteriores.

[13] Spring WebFlow

En SWF un flujo maneja la conversación completa, desde que inicia hasta que culmina y en este punto limpia la memoria automáticamente lo cual es una gran mejoría ya que el usuario no tiene que preocuparse por borrar los recursos en memoria que no estén siendo usados.

En un principio, SWF almacena su ejecución en el repositorio y se muestra la primera página. A continuación, luego de enviar los datos de esta, se carga del repositorio la ejecución del flujo que fue detenida producto de la entrada de un estado de vista y se almacenan los nuevos datos. El mismo procedimiento se sucede hasta que se entra al estado final que se cargan los datos del repositorio de memoria y se despliega una página de confirmación. (**Figura 8**) (14)

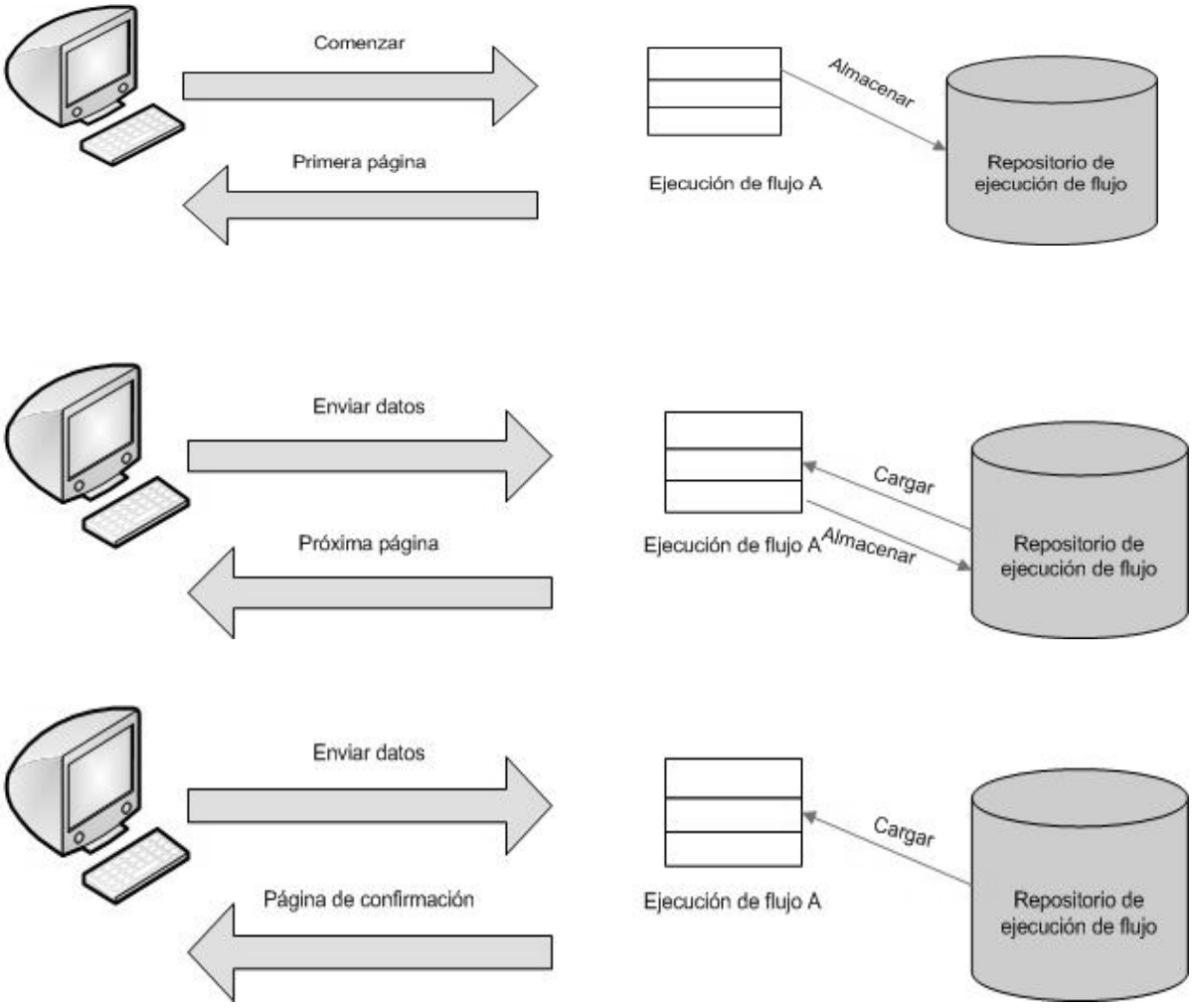

Figura 9 Flujo Web utilizando Spring WebFlow.

Pese a que SWF se considera potente en varios sentidos y con una arquitectura factible, es sólo la solución para la falta de un espacio intermedio entre la petición y la sesión, pues introduce el

concepto de conversación que se traduce en una progresión lógica a través de un predeterminado conjunto de pasos para completar un objetivo del negocio.

Solución	**Situaciones**
Spring MVC Controller	La página existe en aislamiento, no como parte de un flujo y requiere una pequeña lógica para generarla.
Spring MVC SimpleFormController	La página es parte de un simple envío de formulario.
Spring MVC AbstractWizardFormController	Proceso representado por un conjunto de páginas secuenciales a través de las cuales el usuario es guiado.
SWF	Compleja navegación multipágina guiada por estados.

2.3.1. Requerimientos y capas.

SWF requiere de ciertas dependencias para su uso y funcionamiento como son el *framework*, el *spring core* que contienen clases de variadas utilidades usadas internamente por el *framework*, *spring-binding* que es la información de *spring* vinculada al *framework* y usada internamente, *commons-logging*, una simple fachada de *logging* usada también internamente y finalmente OGNL[14] que es el lenguaje de expresión por defecto.

SWF es un *framework* de capas las cuales dependen unas de otras (**Figura 9**).

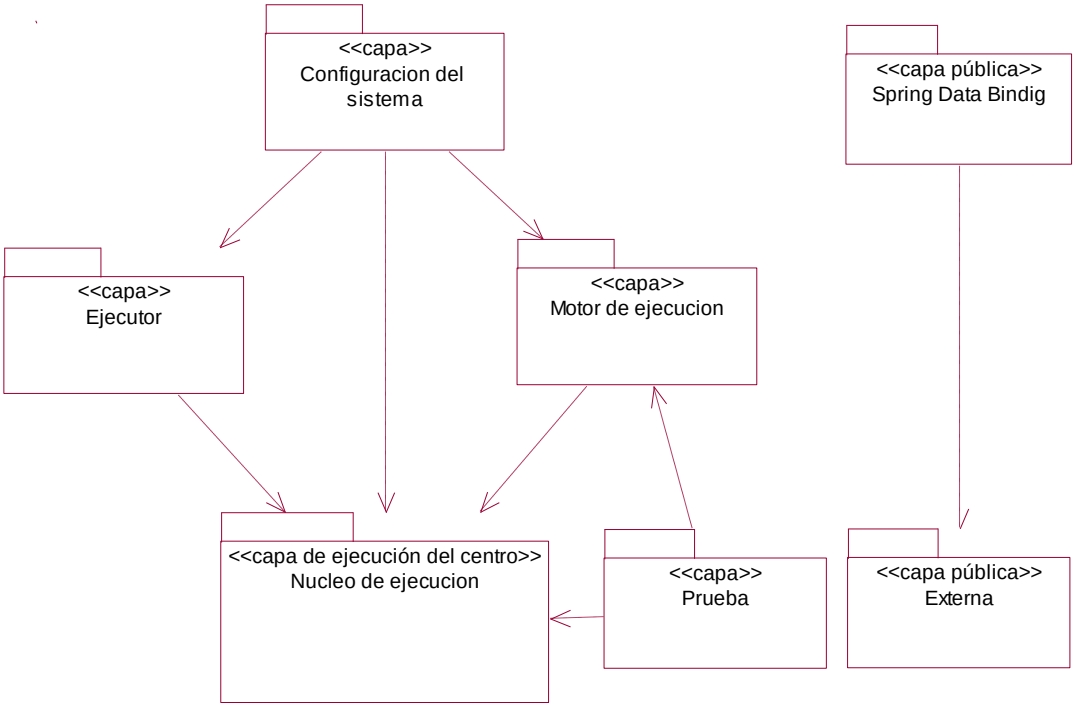

Figura 10 Paquetes que componen la arquitectura de SWF.

Cada capa es particionada dentro de uno o más subsistemas, que unidos, llevan a cabo el rol de cada capa dentro del sistema completo. Cada subsistema trabaja con un conjunto de paquetes que están enraizados en `org.springframework.Webflow`, paquete raíz en la jerarquía de paquetes, cada uno aportando disímiles funcionalidades. Cuentan además con dependencias internas que pueden ser de otros subsistemas de la misma capa o de librerías externas (Tabla 4).

[14] Object-Graph Navigation Language(Lenguaje de navegación de un gráfico de objetos)

2.3.2. Elementos que componen SWF

Los elementos que lo constituyen son los flujos que incluyen estados relacionados por transiciones, todos sencillos de entender, de emplear y factibles para reutilizarlos, manejarlos y configurarlos como un modelo de componentes optimizado con este fin.

Los estados son los puntos en que sucede algo, por ejemplo, desplegar una vista o ejecutar una acción lo que está sucedido por una o varias transiciones que son activadas por eventos y se usan para moverse a otro estado. Los estados tienen diferentes tipos y comportamientos. Todos, exceptuando el estado de terminación, soportan transiciones.

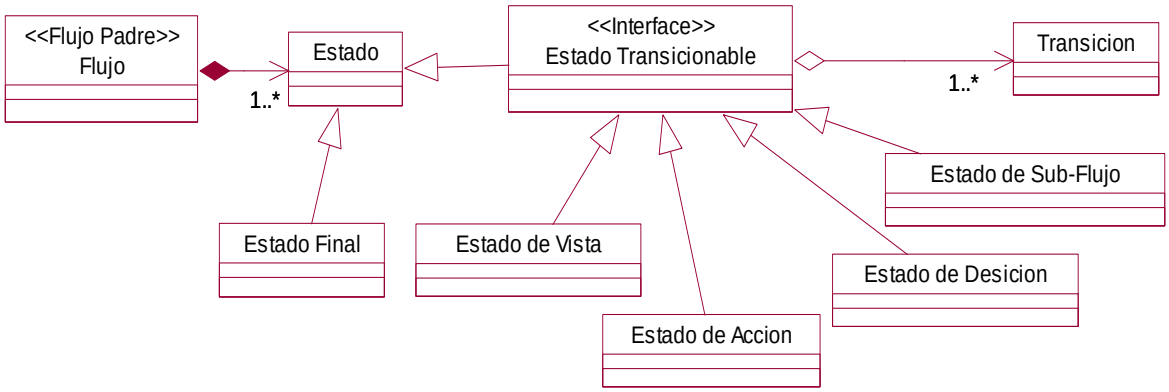

Figura 11 Relaciones entre estados.

Cuando se usa el término estado de inicio se hace referencia al estado que dará comienzo al flujo Web.

2.3.2.1. Flujo

Un flujo es un conjunto predefinido por la aplicación de interacciones del usuario con el sistema expresado a través de un árbol de estados de acciones y vistas conectados los mismos a través de las transiciones. Define un diálogo de usuario que responde a eventos para manejar la ejecución de código de aplicación para completar un objetivo del negocio.

La definición del flujo es una instancia del paquete `org.springframework.Webflow.definition.FlowDefinition`. Este es el artefacto de dominio central que representa la definición de un diálogo de usuario. Contiene uno o más estados, los que definen un paso en el flujo, que cuando es alcanzado, ejecuta un comportamiento determinado. Las transiciones entre estados se corresponden con los eventos que se activan debido a una acción determinada del usuario o al resultado de un método específico. Sólo puede existir un estado inicial sin embargo el flujo puede tener uno o más estados finales.

La implementación de definición de flujo por defecto de SWF es el paquete `org.springframework.Webflow.engine.Flow` el cual contiene una serie de propiedades configurables (Tabla 1).

2.3.2.2. Estado

Una definición de estado define un comportamiento para un paso de una definición de flujo. La clase de implementación base para todos estos tipos de estados de flujo es `org.springframework.Webflow.engine.State`.

Esta clase abstracta define propiedades comunes aplicables a todos los tipos de estados (Tabla 2).

2.3.2.2.1. Estado de acción

El estado de acción ejecuta una o varias actividades especificando el identificador de la implementación de la acción exportada en el contexto de la aplicación (Spring Application context) lo que permite a esta acción ser manejada por Spring y configurada por inyección de dependencia. Retorna un resultado lógico en su ejecución, el cual es mapeado a un estado de transición que encamina hacia otra fase el flujo Web que forma parte del camino configurado del flujo de página. Dichas acciones pueden ser encadenas conjuntamente, rehusadas e invocadas declarativamente. SWF brinda completa libertad sobre la implementación y configuración de las acciones cuando deben ser invocadas dentro del ciclo de vida del flujo. La propiedad usada por `org.springframework.Webflow.engine.ActionState` es precisamente `actions`, lista de acciones a desarrollar cuando el estado es alcanzado.

Las acciones constan de varios atributos que proveen varias características y pueden ser usados para afectar la ejecución de la acción en un contexto de uso específico. Estos pueden resumirse en:

- `Caption`: Breve descripción de la acción.
- `Description`: Amplia descripción de la acción.
- `Name`: Como su nombre lo indica es el nombre de la acción usada para habilitar los eventos resultados de las propias acciones.
- `Method`: Es el nombre del objetivo del método en la instancia de la acción a invocar para llevar a cabo una ejecución.

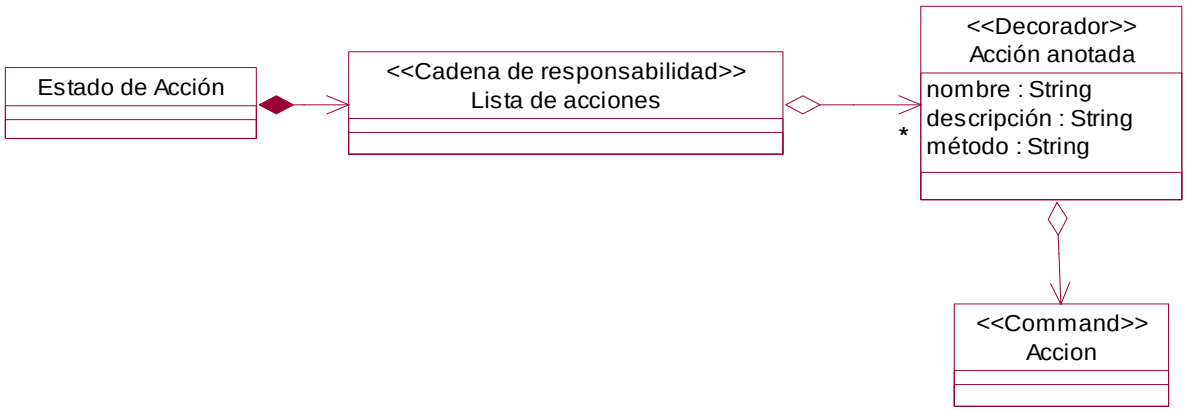

Figura 12 Diagrama de clases que representan el estado de acción.

Existen otros puntos dentro del flujo, que no tienen por qué ser precisamente en este estado, que ejecutan acciones, por ejemplo al inicio de un flujo, cada vez que un estado es activado, cada vez que una transición es determinada pero no ejecutada, cuando un estado de transición es finalizado, una vez que la selección de una vista es hecha y al terminar la sesión de un flujo. Todos requieren de una definición y configuración determinada en el XML (Tabla 3). Estos puntos no permiten ejecutar un estado de transición en respuesta al evento del resultado de la acción ya que esta funcionalidad está sólo presente dentro de un estado de acción.

Dentro de un estado de acción, luego de la declaración de un *bean* de acción con su respectivo método a usar, es posible la transición de parámetros al mismo y la recogida del resultado bajo el nombre de un método de resultado y posteriormente proceder a la transición correspondiente o a la decisión de transición en caso de bifurcación, dependiente esta, del resultado del método. En este caso de decisión, la acción adaptadora juega un papel importante siendo la responsable de la conversión para adaptar la respuesta del método al identificador de un evento. Por ejemplo un método que devuelve true o false tiene asignado por defecto un identificador de evento de Yes o No (Anexo 1).

Es posible también asignar acciones cuando se necesita darle un valor determinado a un atributo en el flujo o en otro ámbito durante el curso de la ejecución del flujo ya que estos atributos son preservados por un tiempo determinado dentro de la ejecución del flujo.(Anexo 2)

2.3.2.2.2. Estado de vista

El estado de vista cuando es accedido causa una pausa en la ejecución del flujo y retorna el control al cliente con las instrucciones para dibujar la vista configurada, para luego, según la indicación del usuario y el evento activado que es mapeado a un estado de transición, describir la acción que próximamente será tomada, o sea, determinar el siguiente paso en el flujo valiéndose de un campo oculto que almacene el identificador de la ejecución del flujo.

SWF brinda control absoluto sobre el proceso de selección de la vista y el cómo un estado de vista responde al evento de entrada de un usuario determinado, no obstante, no es responsable de interpretar respuesta como un controlador, un flujo hace la selección de una vista lógica cuando se requiere una entrada del usuario, donde una selección de vista funciona como una instrucción de respuesta. Como es usual, el paquete contenedor de todas estas funcionalidades, `org.springframework.Webflow.engine.ViewState`, cuenta con propiedades que lo hacen funcional.

- El `viewSelector`, autor de la estrategia que hace la selección de la vista cuando el estado es alcanzado y el `renderActions`, lista de acciones a ejecutar cada vez que una selección de vista restituible es hecha. Permite ejecutar lógica antes de dibujar la página.
- La `org.springframework.Webflow.execution.ViewSelection` clase base es abstracta y actúa como un marcador indicando que una respuesta debe ser publicada al cliente que interactúa con el flujo. Tipos concretos existen para cada una de las respuestas soportadas:
- `ApplicationView`: Solicita la interpretación local de un recurso de vista interno de la aplicación como un JSP o Velocity.
- `FlowExecutionRedirect`: Solicita una redirección atrás al estado de vista en una única URL de ejecución de flujo. Cuando esta URL es accedida en subsecuenciales peticiones, una vista de aplicación, `ApplicationView`, será reconstituida y traducida. La URL es refrescada mientras el flujo de ejecución permanece activo.
- `FlowDefinitionRedirect`: Solicita una redirección que inicia una ejecución de un flujo en su totalidad. Es usado para reiniciar flujos de casos de uso y para concatenación de flujo.
- `ExternalRedirect`: Solicita una redirección a una URL externa arbitraria, usualmente usada para interactuar con un sistema externo.
- `NullView`: Solicita que la respuesta no sea publicada, usada en casos extremos donde el flujo por vías individuales e independientes ha publicado la respuesta.

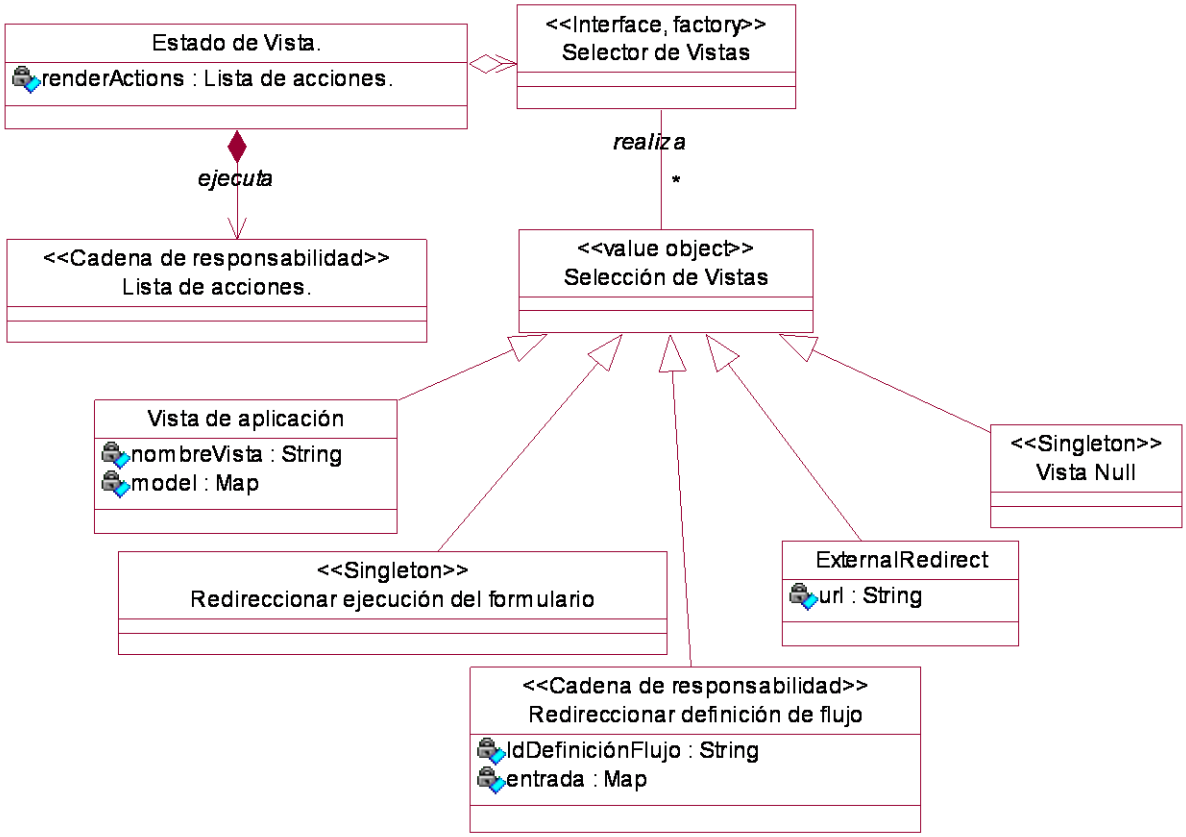

Figura 13 Definición del estado de vista.

Es importante tener en cuenta que varios formularios requieren de ciertas ejecuciones antes de desplegar la vista para que luego de enviar sus datos al servidor exista lógica a ejecutar en el mismo. Se habla de funcionalidades como cargar el objeto de respaldo del formulario y llenar controles con datos iniciales. Esta lógica generalmente implica una correspondencia de las entradas del formulario con su objeto de respaldo, ejecutar un tipo de conversión y validación de datos. Este comportamiento de un estado de formulario, el despliegue y la ida atrás son manejados elegantemente por SWF a través de las capacidades del constructor del estado de vista. (Anexo 3).

2.3.2.2.3. Estado de decisión

Los estados de decisión dinámicos constituyen los elementos de bifurcación que se desenvuelven según las preferencias y la información suministrada por el usuario. Pertenecen al paquete `org.springframework.Webflow.engine.DecisionState`.

Puede tener de una a varias decisiones y las mismas son evaluadas en la ocurrencia de un evento que forma las bases para la decisión (Anexo 4). Las definiciones de flujo no deben ser vehículos para lógica de negocio, en este caso, la decisión tomada es lógica de controlador, razonamiento con un valor precalculado para decidir a qué paso dirigirse próximamente, este es el tipo de lógica que debe existir en un flujo.

2.3.2.2.4. Estado de Subflujo

Cuando se alcanza este estado se activa el subflujo especificado en su interior. La habilidad de un flujo de llamar a otro flujo brinda la posibilidad de construir módulos independientes que unidos crean complejos flujos de trabajo de controlador. Un estado final perteneciente a un subflujo puede retornar atributos de salida que son mapeados por el progenitor o el flujo contenedor en su propio ámbito.

La llamada a un subflujo puede acercarse a la llamada de un método ya que lo mismo reciben que retornan valores, aunque los subflujos son más poderosos al expandir más de una petición en el servidor. Las propiedades que presenta `org.springframework.Webflow.engine.SubflowState` son dos básicamente:

- `Subflow`: Definición del flujo a ser activado como un subflujo.
- `attributeMapper`: Es la estrategia responsable de mapear los atributos de entrada al subflujo y los atributos de salida del subflujo.

Cuando se alcanza un estado de subflujo, primeramente se activa el `attributeMapper` para preparar un mapa de los atributos de entrada para pasarle al subflujo que inicia su ejecución. Al término del subflujo, un evento de resultado es retornado y el flujo contenedor comienza de nuevo con los valores de salida del subflujo que son mapeados por el `attributeMapper` (Anexo 5).

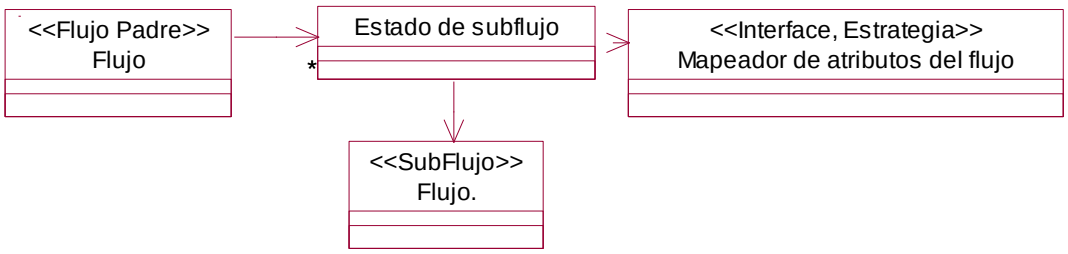

Figura 14 Estado de subflujo.

2.3.2.2.5. Estado final

Finalmente, cuando un estado final llega, la sesión del flujo activo termina y los recursos asociados con el flujo son eliminados automáticamente luego de estar disponibles por todo el tiempo de vida del flujo. Este estado es el responsable de hacer una selección de vista para la respuesta de terminación que puede ser una página de confirmación, la redirección de una petición a otro flujo o a una URL externa. Un estado final de un subflujo no tiene que seleccionar una vista ya que esa responsabilidad es del flujo raíz que inicia la ejecución.

Como ya se ha dicho, al entrar en un estado final el flujo culmina su ejecución, en caso de ser este un subflujo, retorna un evento resultado que el flujo progenitor usa para manejar la transición a otro estado desde el estado de subflujo. Este evento resultado por defecto se corresponde con el identificador del estado final del subflujo.

Propiedades como el `viewSelector` que constituye la estrategia que hace la selección de la vista cuando el estado es alcanzado siempre que el flujo sea de tipo raíz, y el `outputMapper` que es el servicio responsable de exponer los atributos de salida de los flujos, están presentes en este estado de terminación (Anexo 6) .

2.3.2.3. Transiciones

Las transiciones simplemente son conectores entre estados a través de los cuales viajan los datos a lo largo del flujo de ejecución. La implementación de la definición de transición está definida por una instancia de `org.springframework.Webflow.engine.Transition`, que al igual que todos los paquetes, consta de una serie de propiedades como son:

- `attributes`: Atributos adicionales que describen la transición.
- `matchingCriteria`: Estrategia que hace corresponder la transición con la ocurrencia del evento.
- `executionCriteria`: Estrategia que determina si la transición, una vez reconocida, se puede ejecutar.
- `targetStateResolver`: Estrategia que resuelve el estado objetivo de la transición. Permite resolución dinámica.

Una transición puede ser configurada con una o más acciones antes de que se ejecute, pero después que esta es asignada a un evento. Esto es útil cuando se requiere de validación después del evento submit de la página.

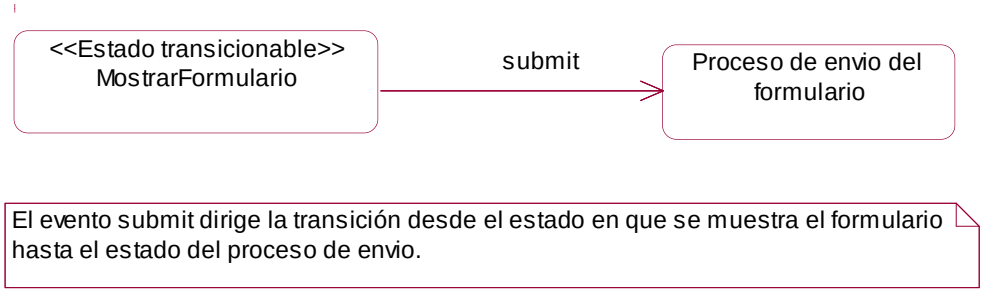

Figura 15 Ejecución de transición.

Las transiciones que se denomina globales le deben el término al hecho de que son compartidas por todos los estados del flujo. Evitan la sobrecarga de declaraciones al encapsularlas como transiciones globales y así los estados puedan tener sus funcionalidades sin tener que declararlas nuevamente (Anexo 7). No sólo en la activación de un evento se puede acceder a una transición, en el caso de excepciones también se puede cambiar de estado vía una transición, ya que esta contiene un atributo denominado `on-exception` que especifica el tipo de excepción, la que es almacenada en el ámbito para que de esta forma esté disponible para procesarla en el flujo.

2.3.3. Manejo de acciones

Todos los estados, de una forma u otra, pueden ejecutar acciones que son implementadas en clases que se configuran en el fichero de configuración de Spring de la aplicación. Estas clases pueden heredar de tres tipos diferentes de clases, la `AbstractAction`, la `MultiAction y la FormAction`, cada una con funcionalidades y requerimientos específicos que se adaptan a las situaciones que se pudieran presentar en un flujo Web. La clase `AbstractAction` constituye la implementación de acción base que provee métodos de conveniencia para las subclases. Sólo puede llevarse a cabo una acción determinada. Su uso es análogo al de la implementación de una interfaz `Controller` en Spring MVC. Similar a esta se tiene la clase `MultiAction` que constituye una implementación que une varias ejecuciones de métodos en una sola clase, o sea, a diferencia de la `AbstractAction`, puede soportar la implementación de varias acciones que tengan algún tipo de relación. Los nombres de las acciones son especificados en un atributo `method` que representa a cada uno de ellos en un momento dado de la ejecución del flujo.

Los métodos de cada una de estas clases reciben un objeto del tipo `RequestContext` que maneja el tiempo de vida de una petición y establece comunicación con el flujo en ejecución accediendo a los valores de los parámetros que vienen en la petición, al contexto externo al flujo, a los diferentes niveles de los ámbitos de datos, al estado de la conversación en un momento determinado, entre muchas otras variantes. Básicamente este objeto representa la ejecución de una única petición dentro de SWF que es activada por la ocurrencia de un evento externo.

La clase `FormAction` es una subclase de la clase `MultiAction` que contiene la lógica para negociar con los datos de entrada del formulario y contiene dos métodos fundamentales:

- `setupForm`: crea un nuevo formulario y lo ubica en el ámbito.
- `bindAndValidate`: luego de hacer corresponder los valores entrados con el objeto de respaldo del formulario, valida los datos si se especifica un *Validator* .
- `bind`: Hace corresponder las entradas del formulario con los atributos del objeto de respaldo del formulario.

Esta subclase recibe la clase que representa al objeto del formulario y el nombre para acceder a sus datos una vez llenados por la secuencia de formularios de la aplicación. También requiere de la entrada del espacio de memoria donde se almacenará la información de los formularios para usarla posteriormente al completar la transacción que dio lugar al flujo.

2.3.4. Máquina de estados

SWF está basado en el concepto de la *Finite State Machine* [15](**Figura 5**), conocida como Autómata Finito. (16) Las interacciones del usuario y las transiciones dentro de la aplicación Web son modelados sobre los principios de esta máquina común de estados y transiciones, donde el flujo cambia de estado según eventos externos, un modelo muy popular en Ingeniería de Software. También puede ser descrita como definiendo un lenguaje, el cual comprendería cada palabra aceptada por la máquina pero ninguna de las rechazadas, se puede afirmar en este punto que el lenguaje es aceptado por la máquina.

El diseño basado en autómatas es una interesante opción de diseño que está ganando popularidad entre los arquitectos de aplicaciones debido a su claridad y disciplinada aproximación al modelamiento de interacciones Web.

La filosofía de SWF dice que cualquier flujo de página puede ser dibujado como un simple diagrama de flujo, donde cada estado en un flujo de página es o una vista o una acción. (16) SWF maneja la transición entre estados y requiere entradas desde las acciones o las vistas para determinar el próximo paso del camino de ejecución configurado del flujo de página.

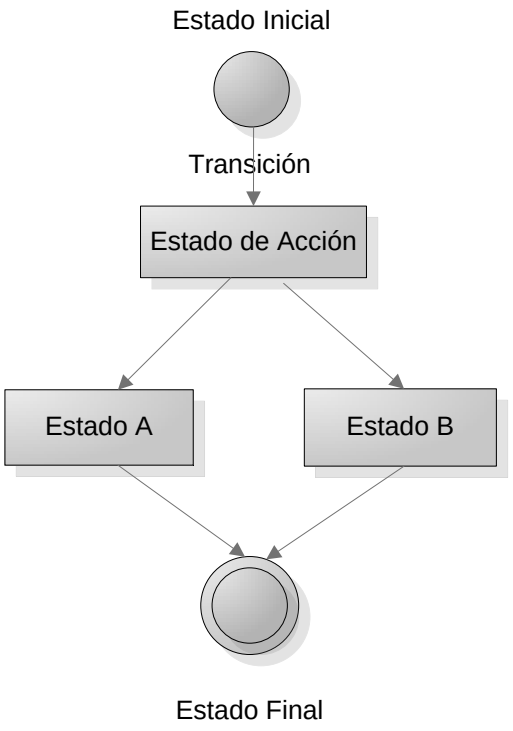

Figura 16 Funcionamiento de la Máquina de Estados.

[15] Máquina de estados finita. Más conocida como Autómata Finito.

En SWF cada flujo es una máquina de estados finita que consta de estados y transiciones, un estado representa la ejecución de acciones o una vista que es desplegada para la entrada de datos.

Cuando la máquina de estados comienza, el flujo entra en el estado inicial y a partir de ahí continúa hacia uno de los tantos tipos de estados que pueden integrar una definición de flujo. Un evento activa una transición desde un estado y mueve el flujo hacia el próximo estado.

Cuando entra un estado de acción el flujo ejecuta una o varios objetos de acciones configurados para ese estado. La ejecución de una acción puede producir un resultado lógico en la forma de un evento, el cual es procesado para seleccionar la transición adecuada, que luego de aplicada, encamina al flujo hacia el estado especificado en la misma.

Cuando se alcanza un estado de vista, el flujo es detenido y una vista es desplegada. Cuando el flujo recibe un evento que se activa producto de una acción de un usuario, se reactiva y se procesa el evento para pasar al próximo estado a través de la transición correspondiente.

Este proceso continúa de la misma forma hasta que se entra al estado final donde el flujo culmina su ejecución y una vista es desplegada al usuario.

La siguiente imagen muestra como los estados son procesados en la máquina de estados.

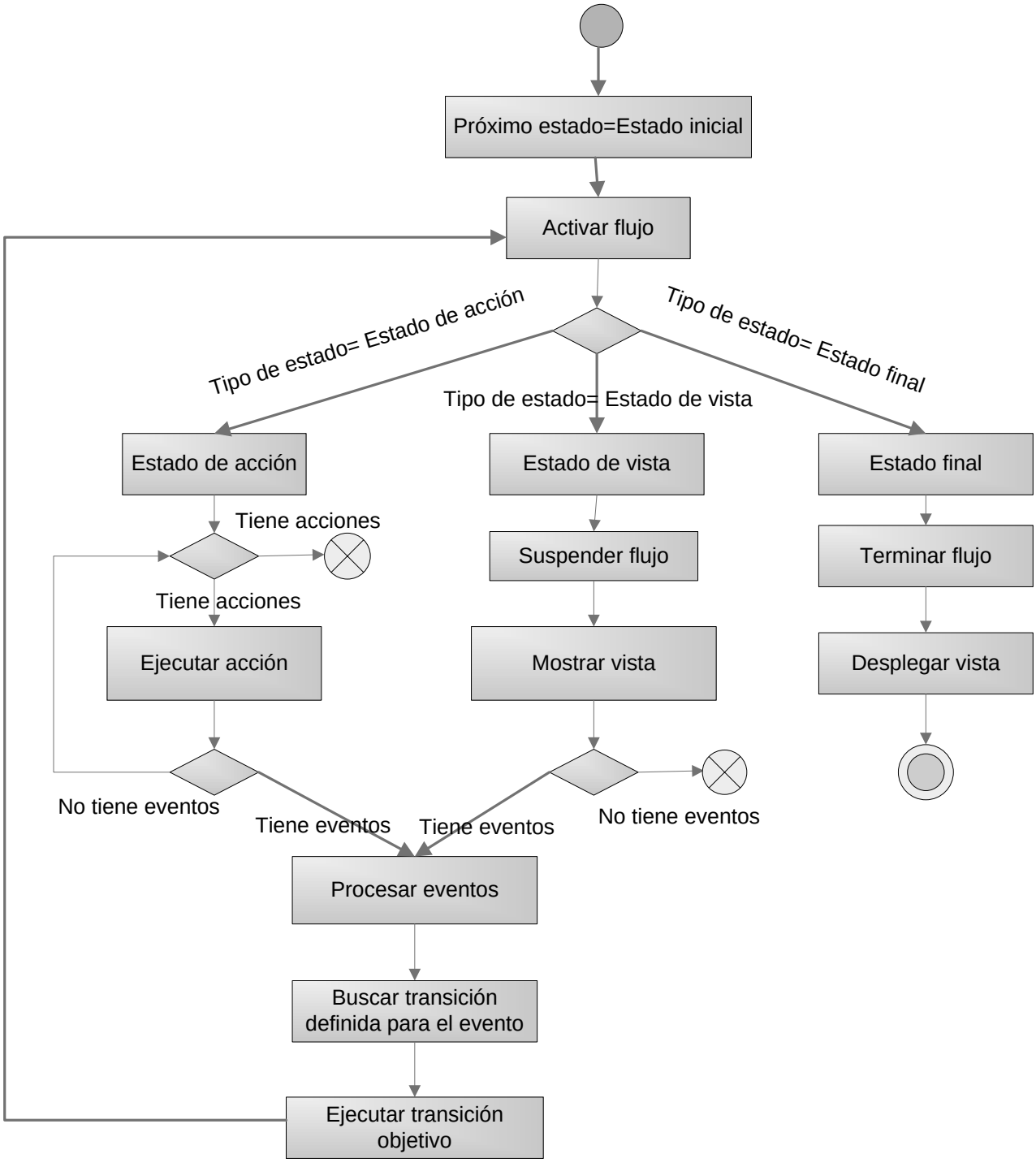

Figura 17 Diagrama de flujo que muestra la filosofía de SWF.

2.3.5. Ejecución de flujo

Una vez que un flujo es definido, cualquier cantidad de ejecuciones del mismo pueden ser iniciadas en paralelo en tiempo de ejecución. La ejecución de un flujo es llevada a cabo por un sistema especializado que está basado internamente en una máquina de estado que corre encima de la máquina virtual de Java. Como el tiempo de vida de la ejecución de un flujo puede abarcar más de una petición en el servidor, este sistema también es responsable de persistir los estados de ejecución a través de las peticiones.

El `org.springframework.Webflow.execution.FlowExecution` es una ejemplificación en tiempo de ejecución de la definición de un flujo. Dada una sola definición de flujo cualquier número de ejecuciones de flujo independientes pueden ser creadas, generalmente por un `FlowExecutionFactory`. La ejecución de un flujo lleva a cabo las instrucciones de programas, plasmadas en su definición, en respuesta a eventos de usuario. Se puede pensar en la definición de un flujo como una clase de Java y la ejecución de un flujo como una instancia u objeto de esa clase, teniendo en cuenta que un evento de ejecución sería como el envío de un mensaje al objeto. (Anexo 8).

Propiedades como las siguientes forman parte de una ejecución de flujo:

- `Definition`: Definición del flujo para ser ejecutado.
- `Listeners`: Grupo de observadores del ciclo de vida de la ejecución del flujo.
- `Attributes`: Atributos globales del sistema que pueden ser usados para afectar el comportamiento de la ejecución del flujo.

La ejecución de flujo, una vez creada, representa el estado de un flujo en un punto del tiempo. Está inicialmente inactivo, hasta que es activado y entra en el estado inicial para proceder con los demás pasos estipulados en el flujo.

Un flujo, en su ciclo de vida puede pasar por las siguientes fases: creado, activado, detenido y finalizado. Con SWF se puede observar el flujo de ejecución implementando el `FlowExecutionListener`.

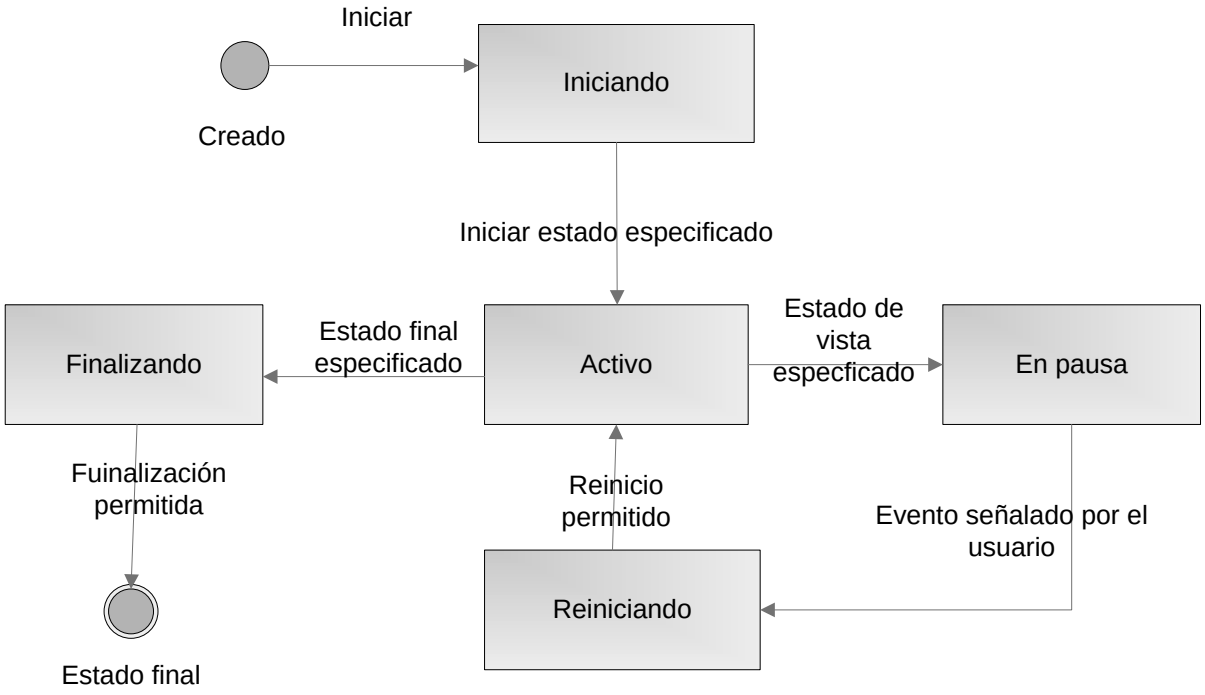

Figura 18 Fases del flujo de ejecución.

Las propiedades contextuales asociadas con la ejecución de un flujo son:

- `active`: Bandera que indica si el flujo de ejecución está activo. Una ejecución de flujo inactiva o no ha terminado o no ha empezado aún.
- `definition`: Definición de la ejecución de flujo.
- `activeSession`: Sesión activa del flujo, rastrea el flujo que está ejecutándose actualmente y en qué estado se encuentra. La sesión activa puede cambiar la vida de la ejecución de un flujo ya que este puede expandir otro flujo como un subflujo. Puede ser interrogada cuando la ejecución está activa.
- `conversationScope`: Mapa de información que forma las bases para el ámbito de conversación. Es compartido por todas las sesiones de flujo.

2.3.6. Sesión de flujo

La distinción o separación entre el flujo raíz y el flujo activo es modelada por la clase `org.springframework.Webflow.FlowSession`. La clase `FlowExecutionImpl` mantiene una

estructura de datos pila donde almacena las sesiones de flujo. En el caso simple de un flujo que no contenga subflujos, en esa pila nunca habrá más de una sola sesión de flujo. Si el flujo contiene subflujos, cada vez que se genera uno nuevo, la sesión de flujo que se encuentra en el tope de la pila, que resulta ser la activa, se suspende, y una nueva sesión de flujo se crea y se inserta dentro de la pila. Cada flujo que se genera tiene su propia sesión de flujo. Cada instancia de una sesión de flujo tiene su propia estructura de datos local, que es la base para el ámbito de flujo o "*flow scope*".

Las sesiones de flujo cuentan con una serie de propiedades como son:

- `definition`: La definición del flujo del que la sesión es una instancia.
- `state`: Estado actual de la sesión.
- `scope`: Mapa de datos que forma las bases para el ámbito del flujo. Los atributos puestos en este mapa serán retenidos para el ámbito de la sesión del flujo. Es local a la sesión.
- `flashMap`: Mapa de datos que forma las bases para el ámbito de flash. Los atributos puestos en este mapa serán retenidos hasta que el próximo evento de usuario sea señalizado en la sesión.
- `status`: Indicador de estado que describe lo que hace actualmente la sesión (Tabla 5).

2.3.6.1. Oyente de la ejecución de un flujo

SWF define la interfaz `org.springframework.Webflow.execution.FlowExecutionListener` que permite observar el ciclo de vida de un flujo y que por supuesto, puede ser implementada lo que permite llevar a cabo diferentes operaciones en todo el transcurso de la ejecución del flujo. Un *listener* pudiese ser una implementación que asegure la entrada a los estados de solamente usuarios autorizados, y en caso contrario se lance una excepción, otro pudiese rastrear la historia de navegación de la ejecución de un flujo, también pudiese ejecutar un determinado proceso transaccional.

El paquete `org.springframework.webflow.execution.FlowExecutionListenerLoader` define cual *Listener* aplicar a cada definición de flujo especificando el nombre del *bean* que representa al *Listener* y el criterio que recibe la definición del flujo.

```
<flow: execution-listeners>
<flow: listener ref="listener" criteria="*"/>
</flow: execution-listeners>
```

2.3.6.2. Identidad de la ejecución de un flujo.

Cuando una ejecución de flujo es creada marca el comienzo de una nueva conversación entre el navegador y el servidor. A la ejecución le es asignado un identificador persistente por el repositorio que consta de dos partes y se reconoce como el parámetro `_flowExecutionKey`. Esta llave es usada por los clientes para restaurar la ejecución del flujo en subsiguientes peticiones.

La primera parte de la identidad persistente de la ejecución de un flujo es un único identificador de la conversación lógica que ha comenzado entre el navegador y el servidor. La segunda parte es un identificador de continuación que funciona como un índice dentro de la ejecución del flujo representando el estado de la conversación en un momento del tiempo.

Unidas estas dos llaves constituyen la única clave de la ejecución del flujo que identifica un estado de una conversación en un punto del tiempo. Enviando esta clave al servidor vía submit en una próxima petición el navegador puede restaurar la conversación y continuar desde ese punto.

De esta forma con una siguiente petición la conversación es reiniciada restaurando la ejecución del flujo del repositorio usando las dos partes de la llave. Después del procedimiento de evento, si la ejecución del flujo sigue activa es salvada nuevamente en el repositorio, automáticamente es generada una nueva clave de ejecución de flujo. Por defecto esta mantiene el mismo identificador de conversación ya que la misma conversación lógica está en progreso, sin embargo la llave de continuación cambia para proporcionar un índice dentro del estado de la ejecución del flujo en este nuevo punto en el tiempo. Al terminar un flujo asociado con una conversación de múltiples peticiones, esta es eliminada del repositorio.

2.3.6.3. Ámbitos de la ejecución de un flujo

Un flujo de ejecución maneja varios contenedores conocidos como ámbitos lo que permite el almacenamiento de diferentes y arbitrarios atributos por un período de tiempo. Existen cuatro tipos de ámbitos, cada uno con una semántica de manejo de almacenamiento diferente:

- petición: Expira cuando una sola llamada dentro del flujo de ejecución termina. O sea, su tiempo de vida es el más corto y los atributos que sean almacenados en este ámbito perderán su valor en cuanto una solicitud sea completada.
- flash: Limpiada cuando el próximo evento de usuario es activado dentro de la sesión de flujo.
- flujo: Expira cuando la sesión de flujo termina. Los atributos almacenados mantienen su valor hasta que el flujo termina, pudiendo ser este de tipo raíz o un subflujo específico. Todos los

flujos, sea cual sea su tipo, tienen su propio ámbito de flujo y los elementos almacenados en el mismo son locales al flujo, o sea, no pueden ser accedidos desde otros flujos. Los objetos situados en el espacio de memoria de flujo mantienen su valor durante toda la sesión de un flujo.

- conversación: Expira cuando la sesión raíz de la ejecución de flujo gobernante o conversación lógica termina. Este ámbito mantiene sus atributos sin afectación desde que comienza el flujo principal hasta que termina, a través de todos los estados, pudiendo ser estos de tipo subflujo, o sea, la información de este ámbito no se pierde aunque el flujo padre active un flujo hijo y detenga temporalmente su ejecución hasta el término del mismo. Los elementos almacenados son compartidos por todos los flujos que puedan generarse durante el curso de una sola ejecución. Representa un subconjunto de la sesión que trata de manera individual cada caso de uso que expanda más de una página en su flujo Web, almacenando así los recursos de ese caso de uso específico. Es el contenedor externo para las variables de ejecución del flujo global.

2.3.7. Continuaciones en SWF

En términos de computación, una continuación es una representación del estado de ejecución de un programa en cierto punto. Muchos lenguajes implementan conceptos que permiten que un programador salve el estado de ejecución en curso de un objeto, y luego restituya el estado de este objeto en un momento posterior y así reanudar su ejecución.

O sea, una continuación es una salva de un estado específico de un programa, una fotografía instantánea de un estado ejecutable de un programa en cualquier momento dado. De manera que sea posible restablecer este estado y reiniciar la ejecución del programa a partir de ese punto. (17)

Las continuaciones ofrecen una manera elegante de tratar la navegación libre en una aplicación Web.

El sistema de SWF emplea las continuaciones para tratar precisamente esto: el uso flexible de algunas funcionalidades de la utilización del navegador como son, un botón atrás, refrescar una página, abrir una nueva ventana, mientras aún se esté usando una definición de flujo del lado del servidor que dirija la navegación, en otras palabras, se usan para permitir libre navegación dentro de la aplicación Web y así persistir y almacenar nuevamente estados del flujo.

Por tanto las continuaciones en el marco de SWF se pueden definir como una fotografía tomada de una ejecución de un flujo en cualquier momento especificado, con lo cual es posible restablecer este estado y reiniciar la ejecución del flujo desde ese punto en adelante. Con esto el estado entero de la ejecución del flujo puede ser restablecido. (17)

Generalmente el uso apropiado del botón atrás del navegador se convierte en un problema. Cuando se usa ese botón, existe el peligro de que los datos mostrados estén dañados o que estén no sincronizados con el servidor. La mayoría de los navegadores populares (como el Internet Explorer), no siguen estrictamente los detalles y además refrescan los datos cuando se usa el botón atrás. Esto se considera como un problema adicional para resolver. Un flujo se realiza a nivel de estados, y mantiene un historial de dónde te encuentras dentro de la aplicación. Es a menudo aceptable, especialmente para las aplicaciones de intranet, informar al cliente que no debe hacer uso de los botones del navegador, que debe ceñirse a los controles ofrecidos por la misma aplicación (vínculos, botones…). Es algo complicado deshabilitar el botón atrás del navegador a través de la programación, por lo que generalmente las aplicaciones Web corren en una ventana pop up sin botones del navegador desplegados.

SWF no es una solución para todo tipo de situaciones, es un sistema poderoso que implementa la navegación controlada que representa las conversaciones que guían el proceso de negocio, pero no es recomendable para todo tipo de aplicaciones.

2.3.7.1. Uso del Repositorio de Ejecución de flujo basado en Continuaciones.

El sistema de SWF típicamente almacena el estado del flujo en tiempo de ejecución en un repositorio respaldado por la sesión de usuario por defecto. Como la estrategia de repositorio de ejecución de flujo se puede cambiar manualmente, es posible seleccionar una estrategia basada en continuaciones. Al hacer esto se está básicamente transformando nuestro flujo en una continuación. Las continuaciones pueden usarse para manipular elegantemente el uso del botón atrás y refrescar del navegador e incluso la apertura de múltiples ventanas en aplicaciones Web. SWF ofrece múltiples repositorios de ejecución de flujo basados en continuaciones, los cuales soportan el almacenamiento de las continuaciones tanto del lado del servidor como del lado del cliente.

2.3.8. Repositorio de flujo

Los ámbitos por defecto que se usan en aplicaciones Web tradicionales son la sesión y la petición, almacenándose en el servidor y en los datos que se envían al servidor respectivamente. El ámbito conversacional requiere de un repositorio para almacenar los datos que persistirán a lo largo de la ejecución de un flujo. El ámbito de conversación no es solamente un subconjunto de la sesión, es la posibilidad de almacenar los datos referentes a una ejecución de flujo en un lugar específico para ello.

Por tanto el responsable de manejar la persistencia de la información de petición en petición a través del objeto flujo, salvándose los datos luego de una petición y cargándolos inmediatamente después de otra, es el repositorio de ejecución del flujo `FlowExecutionRepository`.

Cuando una copia de una ejecución de flujo es salvado en un repositorio, dicho objeto es indexado con una única llave llamada llave de continuación, la que contiene suficiente información para rastrear una conversación lógica que está en progreso y para restaurar un punto determinado dentro de la conversación.

Dicho repositorio consta de tres implementaciones diferentes con rasgos distintivos:

- `SimpleFlowExecutionRepository` (Simple): Almacena sólo un objeto de ejecución de flujo por conversación invalidándolo cuando el estado final es alcanzado. El uso de este repositorio previene el envío de información duplicada cuando se usa el botón atrás. Si se intenta volver atrás y reenviar datos el identificador de continuación almacenado en la historia del navegador no va a machear con el actual identificador de continuación necesitado para acceder a la ejecución del flujo ya que dicho acceso será deshabilitado. Debido a esto es generalmente usado cuando no se tiene que soportar el uso de un botón con navegabilidad en el servidor.
- `ClientContinuationFlowExecutionRepository` (Cliente): Almacena el estado del flujo de ejecución del lado del cliente no requiriendo el uso del estado del lado del servidor. Esto es realizado a través de la codificación de una ejecución de flujo serializada directamente dentro de la clave de continuación de la ejecución del flujo que es enviada en la respuesta. Cuando se requiere la carga de la ejecución de un flujo por su llave en una próxima petición, el repositorio decodifica y deserializa la ejecución del flujo, restaurándolo al estado en que estaba cuando fue serializado. Este repositorio no soporta actualmente la invalidación de conversación después de terminación. Almacenar estado (continuación de

ejecución de flujo) en el cliente conlleva ciertamente a un riesgo de seguridad que debe ser evaluado además de que pone restricciones prácticas en el lado de la ejecución del flujo.

- SingleKey (Única clave): Mantiene la misma llave de ejecución en todo el ciclo de vida del flujo. Hace que el navegador asuma cada orientación de ir adelante o atrás como una orientación de refrescar, debido a que la url es la misma en todas las peticiones, por tanto no hay historial. Es factible pues deshabilita la funcionalidad de los botones de navegabilidad del navegador.

- `ContinuationFlowExecutionRepository` (Continuación): Almacena una ejecución de flujo Web separado para cada petición que entra en el flujo de ejecución. Esto les dará amplia libertad a los usuarios de ir adelante o atrás cualquier número de veces y obtener respuestas correctas. Como la implementación del más simple repositorio, este soporta invalidación de conversación después de terminación, donde una vez que la conversación lógica culmina, toda la conversación es invalidada. Es un tanto más complejo que el simple ofreciendo más poder al habilitar la existencia de varias continuaciones por conversación. Como el tipo de repositorio cliente, el de continuaciones manejará también varias copias del mismo flujo, pero en este caso la continuación es almacenada en el lado del servidor. Consta de una propiedad que define el número máximo de copias del flujo lógico actual, para en el caso de carga excesiva de memoria por intentos innumerados de un usuario determinado de volver atrás, poder borrar la réplica más antigua una vez alcanzada dicha cantidad y de esta forma evitar un ataque de denegación de servicio. Cada llave de entrada del mapa que representa al repositorio es un identificador de conversación, identificando una única conversación entre el cliente y el sistema de SWF, los valores correspondientes a cada uno de estos identificadores en el mapa son objetos del tipo `org.springframework.Webflow.execution.repository.continuation.Conversation`, conteniendo los detalles de una conversación lógica en progreso y cada uno de estos objetos de conversación contiene una lista donde se almacenan `org.springframework.Webflow.execution.repository.continuation.FlowExecutionContinuations` y cada continuación representa un estado restaurable de la conversación en un punto en el tiempo. Es el repositorio asignado por defecto.

2.3.8.1. Funcionamiento del repositorio de flujo

Cuando un flujo alcanza el estado de vista se hace una especie de pausa esperando por los datos de entrada requeridos para continuar, luego de la pausa el `ViewSelection` retornado es usado para publicar una respuesta al usuario que se traduce en un vehículo para la entrada de la información. El usuario entra los datos señalando un evento que reactiva el flujo. Cada vez que el flujo es detenido es salvado en un repositorio y cuando la próxima petición se hace vigente, es restablecido desde el repositorio, reactivado y continuado. Este proceso continúa hasta que la ejecución del flujo llega al estado final en el cual es eliminado del repositorio.

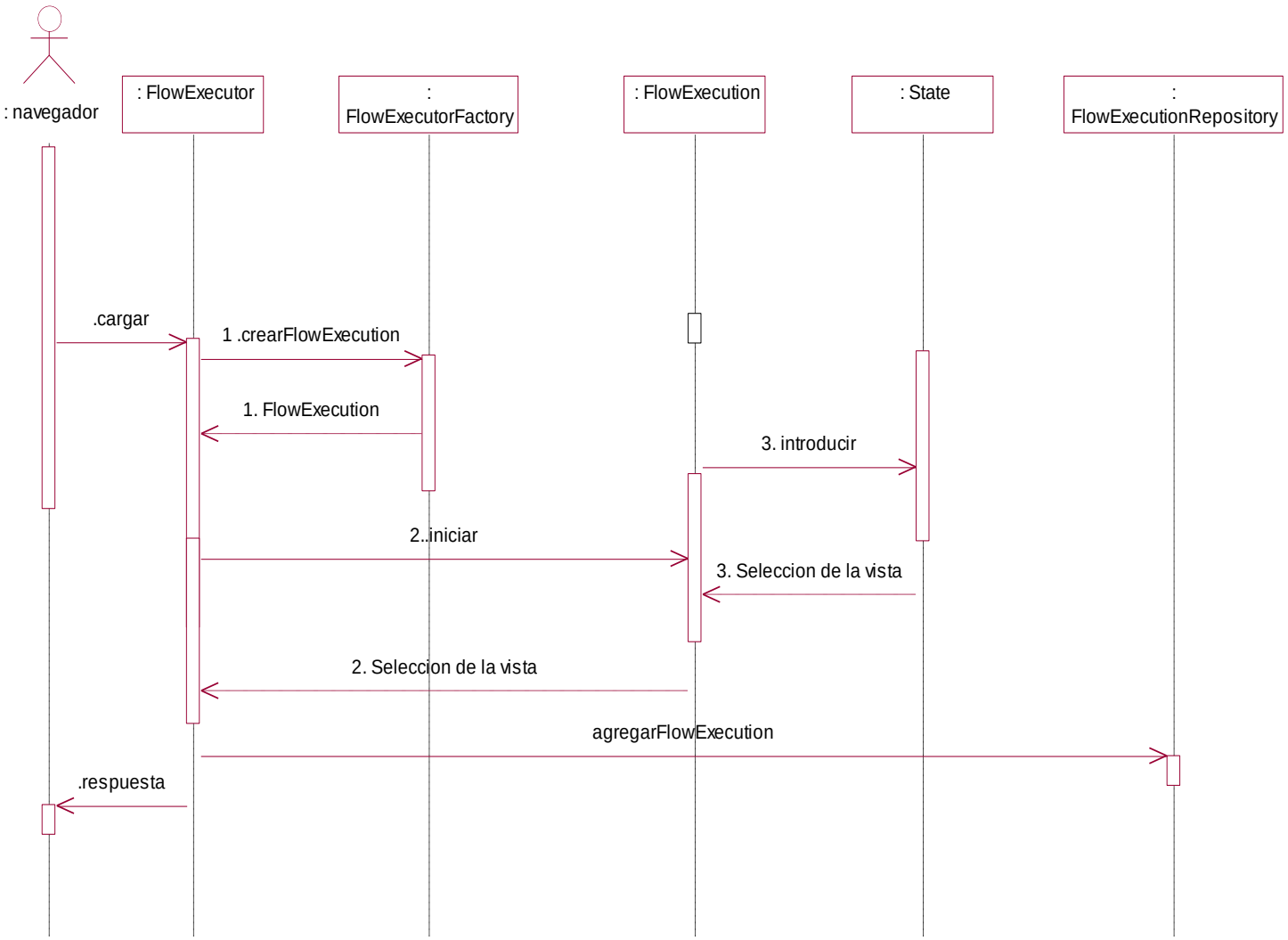

Figura 19 Persistencia de la ejecución del flujo.

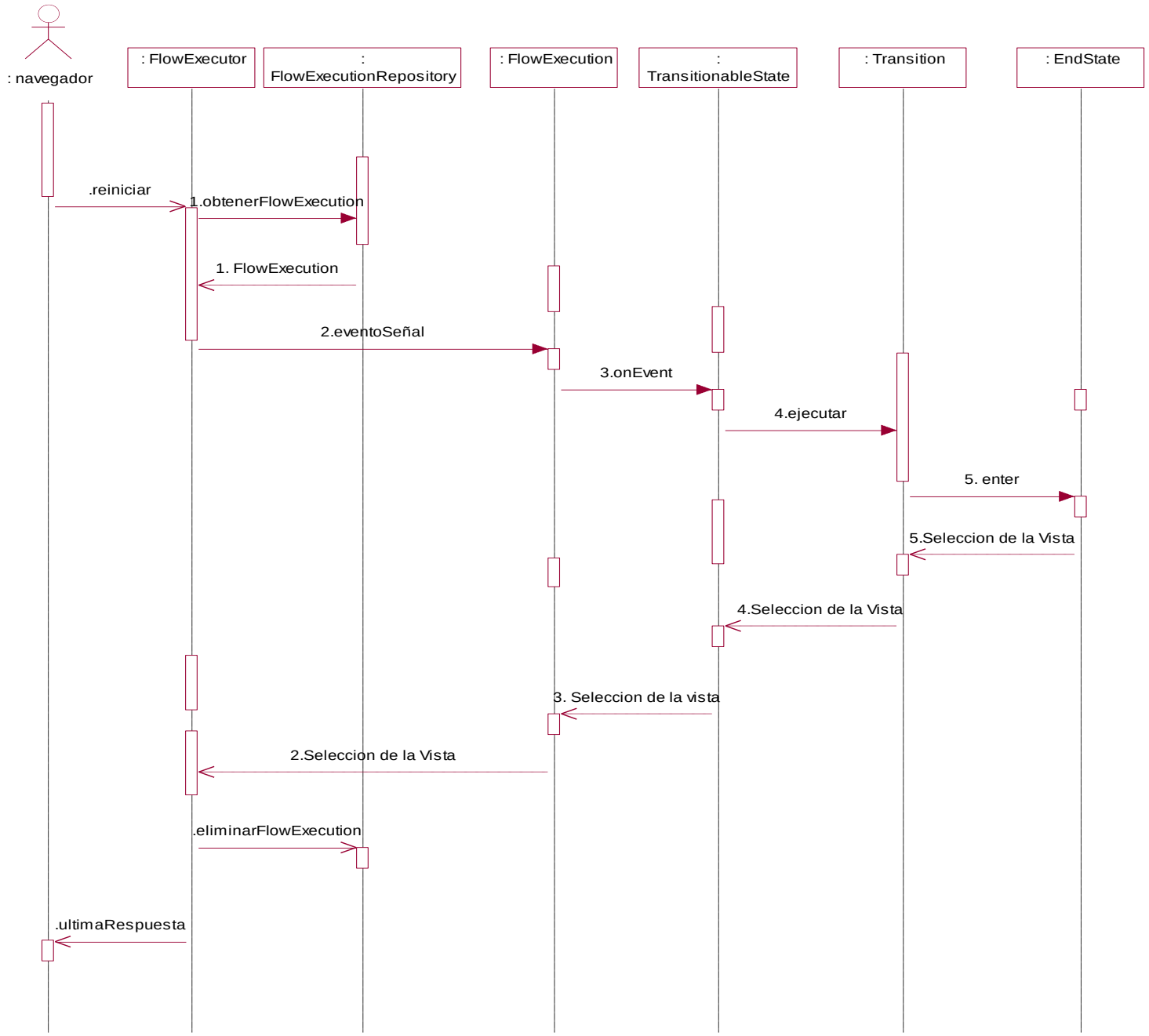

Figura 20 Restauración de la ejecución del flujo y removerlo en un final.

2.3.9. Controlador de flujo Web

El componente que realiza la integración de los flujos Web dentro de la plataforma de Spring MVC es precisamente un controlador especializado en este tema, el `FlowController`, que como parte de sus funcionalidades, carga un flujo Web cuando este es inicializado por Spring. Como cualquier otro controlador requiere de definición como un *JavaBean* en el contexto de la aplicación del *DispatcherServlet* usando el *bean* como nombre de la petición que levanta el flujo. De esta forma, Spring MVC encaminará las peticiones hacia ese *bean* (Anexo 9).

Cuando el controlador de flujo es invocado para manejar una petición analiza los parámetros para determinar qué hacer. Los parámetros de las peticiones por defecto son:

- `_flowId`: Identificador de la definición del flujo, se requiere para comenzar una nueva ejecución de flujo.
- `_flowExecutionKey`: Llave de ejecución de flujo, se requiere para reactivar o refrescar una ejecución de flujo ya existente.
- `_eventId`: Identificador del evento que ocurre, se requiere para reactivar una ejecución de flujo ya existente.

(Anexo 10)

Con los parámetros anteriores, el controlador de flujo, cuando llega una petición, recupera de la misma el `_flowExecutionKey`. Si está presente, el controlador de flujo restablecerá la ejecución de flujo en proceso y la reactiva. En caso de estar el `_flowId`, el controlador inicia una nueva ejecución para el flujo especificado como valor de dicho parámetro y a la nueva ejecución se le asignará una clave única que será precisamente la que se usará como parámetro en las páginas posteriores.

Para una ejecución de flujo existente, el valor de `_eventId` será usado para activar una transición definida en el estado actual del flujo. Si no se especifica ningún identificador de evento, la ejecución del flujo será sólo refrescada.

El resultado de los pasos anteriores es una instrucción de respuesta que contiene un modelo y una vista a mostrar. La clave de ejecución del flujo está disponible en el modelo y puede ser usada por la vista para referenciar atrás en la ejecución de flujo en progreso.

Este controlador, para su correcto funcionamiento, usa un ejecutor de flujo que a su vez usa un registro de flujo, cada uno con una tarea específica dentro de SWF.

2.3.9.1. Ejecutor de flujo

Los ejecutores de flujo son los puntos de entrada de alto nivel dentro del sistema de SWF, responsables de manejar la ejecución de los flujos a través de una variedad de ambientes.

La fachada `org.springframework.Webflow.executor.FlowExecutor` actúa como un simple y conveniente servicio de acceso dentro del sistema de SWF que es reusable en varios ambientes (Anexo 11).

Existen tres casos de uso de esta interfaz:

- `Launch`: Iniciar una nueva ejecución de una definición de flujo.
- `Resume`: Reactivar una ejecución de flujo detenida.
- `Request`: Solicitar que la última respuesta publicada por la ejecución del flujo sea republicada. Al contrario del inicio y los eventos de señalización, la operación de refrescar no afecta el estado de la ejecución de un flujo.

Cada operación acepta un contexto externo `ExternalContext` que proporciona acceso normalizado a las propiedades de un sistema externo que sea llamado dentro de SWF, permitiendo acceso a los parámetros de peticiones de un ambiente específico, tan bien como a la petición, a la sesión y atributos del nivel de aplicación.

Cada operación retorna una instrucción de respuesta `ResponseInstruction` la que el sistema que llama se espera que use para publicar la respuesta. Existe un contexto externo por cada uno de los ambientes de soporte de SWF.

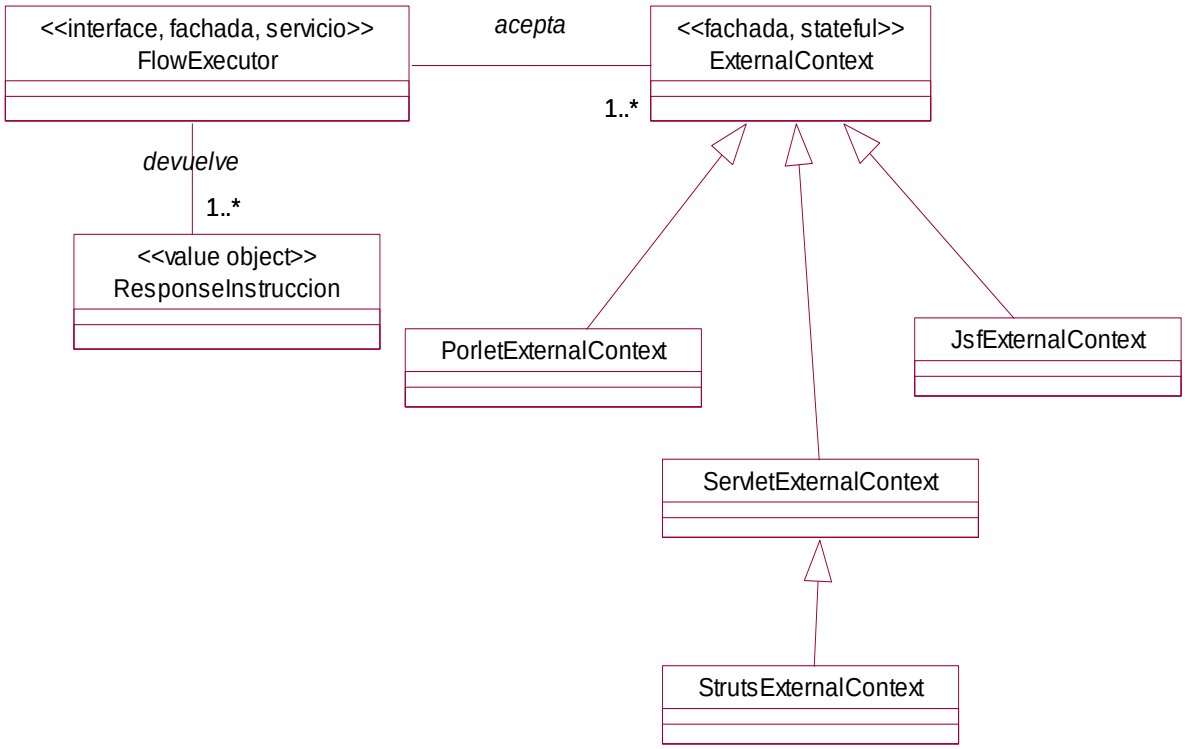

Figura 21 Ejecutor de flujo.

La implementación de la interfaz del ejecutor es `org.springframework.Webflow.executor.FlowExecutorImpl`, la que permite configurar un localizador de definición de flujo `FlowDefinitionLocator` responsable de cargar la definición de flujo a ejecutar, así como la estrategia de repositorio de ejecución de flujo persiste las ejecuciones de flujo que permanezcan activas más allá de una simple petición al servidor. Hay tres propiedades configurables dentro de este paquete:

- `definitionLocator`: Servicio para cargar la definición de flujo a ser ejecutada, generalmente un registro de definición de flujo `FlowDefinitionRegistry`.
- `executionFactory`: Fábrica para crear nuevas ejecuciones de flujo.
- `executionRepository`: Repositorio para salvar y cargar las ejecuciones de flujo persistentes (Anexo 12)

El ejecutor de flujo puede recibir atributos de ejecución del sistema con sus respectivos valores y puede auxiliarse de un manejador de conversación para almacenar el estado conversacional. (Anexo 13)

2.3.9.2. Registro de flujo

El `org.springframework.Webflow.definition.registry.FlowDefinitionRegistry` se considera una especie de diccionario para ubicar las diferentes definiciones de flujo de una aplicación, las cuales son indexadas dentro de este registro por sus identificadores. Si hay gran variedad de flujos en la aplicación, es muy común usar patrones para evitar poner todos de forma consecutiva dentro del registro de flujo. Usando una especificación que se adapte a los nombres de todos los flujos ahorra el exceso de código y el registro de flujo localizará el flujo especificado a través de dicho patrón.

2.3.9.3. Manejador de flujo

Todas las implementaciones de repositorio de ejecución de flujo delegan el almacenamiento de la información actual y el manejo de las responsabilidades al manejador de conversación o flujo, el `ConversationManager`. El repositorio cliente usa un tipo de manejador que no ejecuta ninguna función mientras que los dos restantes usan el manejador de sesión `SessionBindingConversationManager`, que es la única implementación de manejador que se corresponde con SWF y que almacena el estado de la conversación en la sesión y que puede soportar un número máximo de conversaciones simultáneas, ya que un usuario puede estar involucrado con más de un flujo de ejecución y especificando esta cantidad, el manejador podrá terminar la más antigua conversación a medida que surja una nueva que exceda el límite señalado. Los datos se mantendrán activos sólo hasta que la sesión expire y se puede acceder a los mismos dentro de la ejecución del flujo a través del contexto de la petición `RequestContext` como ya se pudo observar anteriormente. Se recomienda configurar en el archivo descriptor de despliegue de la aplicación, el `Web.xml`, el intervalo de la sesión con el objetivo de expirar las sesiones desocupadas luego de una razonable cantidad de tiempo para liberar los recursos del servidor.

2.3.10. Acceso externo a la información del flujo

SWF, como ya se ha explicado almacena toda la información relacionada con la ejecución de flujo en el objeto `FlowExecution`, que puede ser accedido a través del contexto de la petición o `RequestContext`. Todo esto desde dentro del mismo flujo, lo que cambia considerablemente a la hora de obtener dichos datos desde un punto externo al flujo. Se requiere para esta tarea la llave del flujo que está incluida en cada una de las peticiones que entran a este ámbito y el repositorio donde se almacena el objeto del flujo contenedor de recursos.

Todas las especificaciones del flujo y su ubicación son configuradas en el *DispatcherServlet*, el cual puede ser accedido vía programación de esta forma:

```
ApplicationContext contexto =
 (ApplicationContext)getServletContext().getAttribute(
 DispatcherServlet.SERVLET_CONTEXT_PREFIX + "datos");
```

Una vez obtenido, se puede derivar el proceso de llegar al objeto del flujo accediendo al controlador `FlowController` que maneja el flujo que contiene el repositorio de datos que a su vez almacena la llave y el objeto que se necesita, creándose para este fin un contexto externo que requiere de visibilidad y acceso desde todas las partes del sistema. Finalmente se procede a la asignación del filtro creado.

```
FlowController controller = (FlowController) contexto.getBean("/datos.htm");
FlowExecutionRepository repository = ((FlowExecutorImpl)controller.getFlowExecutor()).getExecutionRepository();
FlowExecutorArgumentHandler handler = controller.getArgumentHandler();
ExternalContext externalContext = new ServletExternalContext(getServletContext(), request, response);
ExternalContextHolder.setExternalContext(externalContext);
FlowExecutionKey flowExecutionKey =
repository.parseFlowExecutionKey(handler.extractFlowExecutionKey(externalContext));
FlowExecution flowExecution = repository.getFlowExecution(flowExecutionKey);
```

De esta forma se puede llevar a cabo disímiles operaciones con propósitos específicos, por ejemplo la obtención del identificador de un flujo a través del objeto de ejecución del flujo quien

accede a la sesión activa y esta a su vez obtiene la definición del flujo cuya ejecución está representada por dicha sesión. Posteriormente, se puede obtener cualquier atributo de la definición obtenida, en este caso un identificador.

```
Int identificador=flowExecution.getActiveSession().getDefinition().getId()
```

Otro caso pudiera ser la obtención del valor de un elemento determinado que se encuentre en un ámbito determinado. Se debe tener en cuenta, que cada flujo, cuando es configurado, se le es asignado un ámbito donde se almacenarán todos los recursos involucrados con la ejecución del mismo. Dicho ámbito se puede obtener mediante la sesión activa, que como ya se sabe se obtiene del objeto de ejecución del flujo. En este caso se obtiene el valor del criterio de búsqueda empleado en un flujo determinado. (18)

```
flowExecution.getActiveSession().getScope().get("criterioBusqueda")
```

2.3.11. Pruebas de ejecución de un flujo.

SWF provee soporte dentro del paquete `org.springframework.Webflow.test` para probar ejecuciones de flujo con `JUnit`. Existen requisitos a cumplir para la utilización de estas funcionalidades, como son que las implementaciones propias de artefactos usados por el flujo, o sea, las acciones, los manejadores de eventos y los que mapean atributos deben ser probados separados de los demás. La ejecución de un flujo debe ser probado como parte de las pruebas de integración del sistema. Tales pruebas deben ejercitar todos los posibles caminos del flujo, asegurando que el flujo responde a los eventos como se espera.

En un ejemplo sencillo se puede hacer referencia a una búsqueda de un elemento por un criterio de búsqueda determinado, que evalúe la corrección del mismo, despliegue el resultado en caso de existir y ofrezca la posibilidad de detallar uno de los resultados. Primeramente se tiene un estado de vista que muestra la pantalla para entrar el criterio de búsqueda, se valida el valor entrado y se dirige el flujo hacia otro estado de vista que ejecute la búsqueda como parte de las acciones de entrada, y que brinde la posibilidad de regresar a buscar de nuevo en caso de ser no válida la entrada del criterio o a detallar un componente de la lista encontrada. Este proceso de detallar sería un estado de subflujo que active un subflujo dentro del inicial.

En este caso particular se podrían asegurar varios aspectos que de resultar satisfactorios, se cumpla con todo lo requerido en el flujo. Por ejemplo, que cuando la ejecución del flujo inicial entre al estado que despliega la vista para entrar el criterio de búsqueda, hace la selección de vista especificada conteniendo un objeto de formulario a ser usado como base para el conjunto de campos del formulario; que si en el submit, con entradas válidas, la búsqueda es ejecutada y la selección de vista de los resultados de la misma es hecha; que si en el submit de la página, con datos incorrectos la vista de entrada de criterio es seleccionada nuevamente; que si en la nueva búsqueda, la vista de entrada de criterio es seleccionada; que si en el evento seleccionar un elemento, el flujo de detalle es activado con el identificador del componente seleccionado.

Para asistir por escrito todas estas argumentaciones, SWF usa el soporte de pruebas de ejecución de flujos dentro del paquete especificado anteriormente, clases bases de prueba como las siguientes son las usadas en este contexto:

- `AbstractFlowExecutionTests`: La más genérica clase base para pruebas de ejecución de flujo.
- `AbstractExternalizedFlowExecutionTests`: Clase base para pruebas de ejecuciones de flujo cuyo flujo es definido dentro de un recurso externo, por ejemplo un fichero.

- `AbstractXmlFlowExecutionTests`: Clase base para pruebas de ejecuciones de flujo cuyo flujo es definido dentro de un recurso XML externo.

2.4. Conclusiones

SWF se considera una plataforma consistente y avanzada, compuesta por una serie de elementos que se ubican en las diferentes capas de su sólida arquitectura, cada uno prestando servicios diferentes de acuerdo a su ubicación. Cada capa provee recursos que son usados por las capas superiores para completar lo que es la gama de funcionalidades brindadas por esta plataforma. Su modelo de componentes esta optimizado para reutilización arquitectural y una configurabilidad y manejabilidad altamente efectivas.

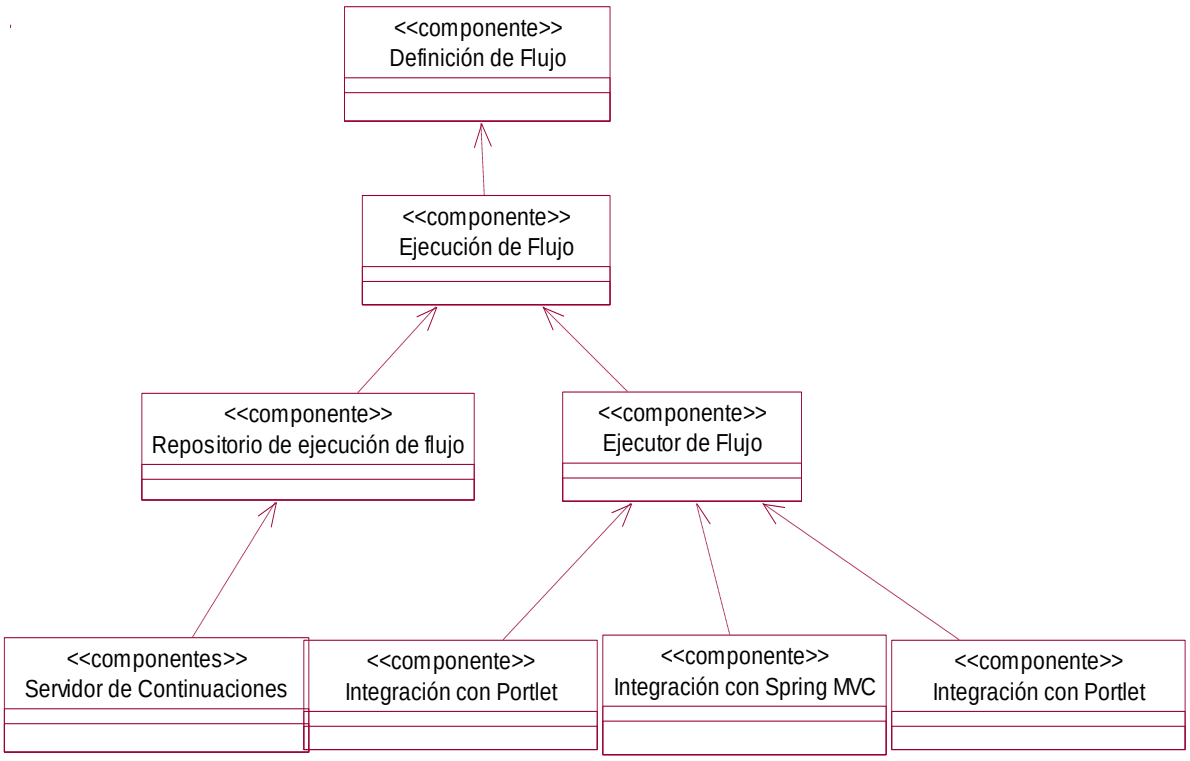

Figura 22 Arquitectura de alto nivel de SWF.

Con toda la teoría abordada anteriormente se puede comenzar la implementación de una aplicación que incluya casos de uso con varias páginas en su flujo, lo que se traduce como navegación ordenada, con una secuencia definida y configurable. Los elementos expuestos en este capítulo representan el todo funcional de la plataforma SWF, sus rasgos y características que la hacen adaptable y fácil de usar en aplicaciones que requieran de sus funcionalidades.

III. CASO DE ESTUDIO.

3.1. Introducción

El proyecto SIGEP, actualmente en desarrollo, cuenta con varios módulos, cada uno con un nivel de complejidad determinada por su funcionalidad final.

Lo que nos atañe es el ingreso de un individuo al centro penitenciario, lo cual lleva una secuencia de pasos inquebrantable, o sea, un flujo organizado y secuencial que en su término cuente con todos los datos necesarios para ingresar una persona en la institución correspondiente que puede ser un centro penitenciario común, un CTC [16]o un UTASP[17]. En este caso se trabaja sobre el flujo de ingreso en un centro penitenciario ya que es la parte del módulo que desarrollamos actualmente en el proyecto.

Cuando se inicia el proceso de ingreso, primeramente aparece una pantalla de presentación que explica brevemente el objetivo del wizard que se usará a continuación. Seguidamente aparece una página con los cupos asignados existentes en el sistema. Luego de seleccionar el cupo correspondiente al individuo que ingresará en el centro, se accede a una página que permite la entrada de datos mínimos para posteriormente, en una sexta pantalla, de acuerdo a una entrada específica del usuario, remitirnos hacia la pantalla de requisitoria o la de documentos del individuo y a continuación, sea cual sea la decisión anterior, se muestra un popup con información sobre el ingreso, que cuando es cerrado, se accede a la página de notificaciones.

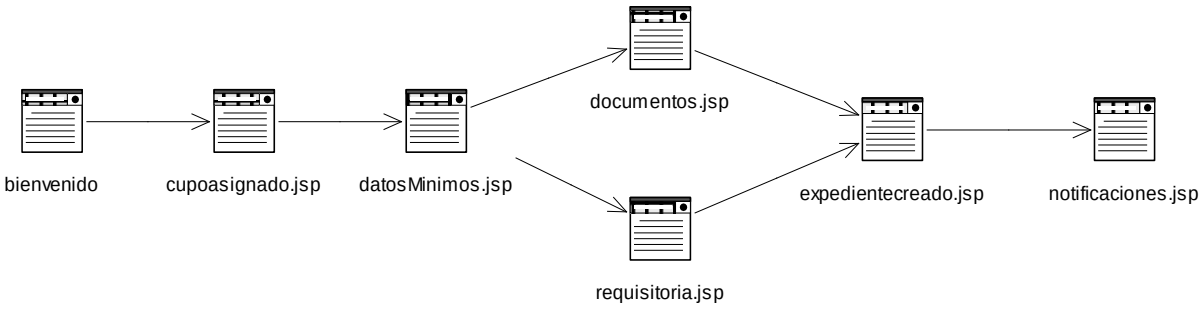

Figura 23 Secuencia para el ingreso de un individuo al sistema.

[16] Centro de Tratamiento Comunitario
[17] Unidades Técnicas de Apoyo al Sistema Penitenciario.

Este proceso, descrito a grandes rasgos, es toda la secuencia de pasos que se debe controlar para conformar la estructura del flujo que controlará la transición de estado a estado enviando los datos pertinentes y requeridos.

Todo este proceso está siendo elaborado con Spring MVC, *framework* contenedor de varios controladores que se especializan en diferentes tareas, en este caso el usado fue `AbstractWizardFormController`, para controlar los flujos Web de una aplicación. Debido a las ventajas y funcionalidades superiores en estas situaciones de casos de uso que abarcan más de una página en su flujo Web que ofrece el *framework* SWF, en todo el proceso de ingreso se propone el uso de esta tecnología como ya es sabido.

3.2. Pasos preliminares para la implementación del flujo de ingreso

Implementar de forma ordenada de manera que al necesitar de una funcionalidad ya esta esté disponible es uno de los aspectos a tener en cuenta a la hora de construir todos los artefactos necesarios que constituyen las partes del todo que es SWF.

Se hace necesario definir los estados por los que transita el flujo, los parámetros de entrada requeridos en cada uno, las respectivas salidas y las acciones a ejecutar en cada caso. Todo esto luego de saber a ciencia cierta qué se necesita para ello.

Primeramente se cuenta con un controlador de tipo `formAction` que maneja toda la lógica de los formularios ya que su función es mediar entre el contexto de la petición y el objeto del formulario, lo que se traduce en la creación del objeto del formulario para posteriormente posicionarlo en el ámbito que se esté usando con el método `setupForm` y la validación donde se mapean los parámetros de la petición a las propiedades del objeto del formulario con el `bindAndValidate`. El `formAction` recibe como entrada de una de sus propiedades la clase que representa al objeto de respaldo del flujo que se vaya a ejecutar, el nombre con que se accederá a los datos de la misma en el ámbito, que a su vez también es especificado como parte de las propiedades de esta clase y por último el validador encargado de validar las páginas.

Los flujos existentes en la aplicación no tienen por qué usar un mismo `formAction`, ya que la clase de respaldo del formulario, el validador de la misma y el ámbito que usa puede que no coincidan. La etiqueta `import` es la encargada de especificar dentro de la definición del flujo la ubicación del xml que contiene toda esta información referente al flujo, por tanto se puede usar un xml diferente para cada `formAction` de forma tal que los recursos que requiera un flujo en particular no estén ligados a los de otro flujo dentro de la aplicación. De esta forma se tendrá el `sigep-controlpenal-ingresosegresos-beans-ingreso.xml` para los recursos del ingreso.

La clase que representa al objeto del formulario es un artefacto inmediato a construir, con atributos que se encuentran en los formularios como campos de entrada, los que son validados luego de una petición al servidor con una clase que implementa la interfaz `Validator`. Dicha clase, denominada `IngresoValidator`, tiene un método de validación por página, el cual es especificado en el flujo con el atributo

`validatorMethod` y como valor, el nombre del método a usar, todo esto de acuerdo a la página que le corresponda estar en pantalla.

Puede que cada flujo recoja un conjunto de datos específico y diferente, en este caso, se requiere de una única clase de respaldo para los datos del individuo a ingresar, la `IngresoCommand`. El subflujo del popup sólo muestra información sobre el ingreso, no necesita coleccionar o almacenar información para completar una transacción.

Estas clases implementan la interfaz `Serializable` ya que pueden ser la entrada o la salida de algún método que se ejecute dentro de un estado del flujo en progreso y por tanto es necesaria la serialización que convierte una clase en `String` para ser accedida y usada desde el ámbito que se especifique como una de las propiedades del `formAction` correspondiente al flujo actual.

Se tiene también una clase para manejar todas las acciones de carga de datos que se requieran en los diferentes estados del flujo, `CargarDatosClass`. Cada método, con un nombre que responda a su objetivo y devolviendo un resultado que es ubicado en el ámbito en uso para ser accesible desde las páginas. Evidentemente esta clase está mapeada en el contexto de la aplicación y, en caso de ser los resultados objetos o grupos de objetos, las clases a las que pertenecen, implementan la interfaz `Serializable`.

Es importante mapear el controlador del flujo `flowController` en el *servlet* [18]ya que el mismo, auxiliándose de sus propiedades, localiza, activa y le da continuación al flujo especificado. La propiedad fundamental que requiere este controlador es la especificación del ejecutor del flujo quien recibe un registro del flujo que contiene la dirección física del mismo y le especificado el tipo de repositorio que usarán los flujos de la aplicación, en este caso el de continuaciones y el número máximo de continuaciones a soportar, 40, para evitar sobrecarga de datos en memoria.

Se cuenta con una clase que hereda de `Multiaction` cuyos métodos se usan básicamente para mover de ámbitos los parámetros que vengan en la petición previniendo el acceso denegado a los mismos en próximos procedimientos. A través de esta clase se realiza el ingreso del individuo obteniendo todos los elementos necesarios del objeto del `RequestContext` que entra como parámetro al método.

[18] Sigep-controlpenal-ingresosegresos-servlet.xml

Todas las páginas involucradas en un flujo tienen los parámetros requeridos para darle funcionalidad al mismo, ya sea de iniciación o continuación además de la especificación del objeto que representa la clase de respaldo del formulario.

En el caso de dar inicio al flujo, en el menú se solicita la petición que accede al `flowController`, pasando como parámetro el `_flowId` y como valor del mismo, el nombre del fichero que contiene el flujo que se desea activar. Las páginas que conforman el flujo requieren del `_flowExecutionKey` que es generado para cada jsp una vez comenzada la ejecución del flujo y que el controlador reconoce como una orientación de continuidad, por tanto es necesaria su presencia en todas las peticiones al servidor dentro del flujo.

Los eventos están definidos con un identificador que sugiere su objetivo en el flujo y es el evento especificado en la transición correspondiente del flujo, clave esto para que resulte satisfactoria la estrategia de macheado del evento activado en la página con la transición. La etiqueta `spring:bind` es especificada para todos los campos de los formularios que tengan un equivalente en la clase del objeto del formulario, para de esta forma, mantener los datos en caso de acceder a una página llenada anteriormente.

Es válida la aclaración de que sólo se puede acceder a los datos del objeto de respaldo del formulario a través del ámbito que use el flujo, en cambio, los parámetros que se encuentren en las páginas y lleguen al flujo en la petición requieren de una acción adicional si se van a usar posteriormente. Dicha acción se traduce en un método en el `MultiactionIngreso` que obtiene de la petición el parámetro que se requiere y lo incorpora en el ámbito que use el flujo, todo esto a través del objeto `RequestContext` que es la entrada del método.

3.3. Requerimientos de software

Inicialmente, los ficheros de tipo xml que se usen para plasmar la definición de un flujo, acceden al fichero spring-SWF-1.0.xsd, que provee las etiquetas necesarias para conformar el flujo así como también los ficheros de los beans de la aplicación requieren de spring-SWF-config-1.0.xsd para la configuración de los recursos que hacen funcional la plataforma. Para ello es necesario agregar el xsd vía el catálogo de xml de Preferencias, en una ubicación física de la máquina para evitar configurar el Proxy y el acceso a Internet que puede retardar el trabajo .Es necesario habilitar la validación de dichos xml en las propiedades de la aplicación para verificar la estructuración del mismo.

Estos ficheros, una vez creados, se agregan en la opción SWF así como también se agregan los demás xml de la aplicación, pero estos, en la opción Spring. Ambas opciones ubicadas en las propiedades del proyecto. Con el objetivo de completar código en caso de acceder a un método de un bean determinado en la elaboración del flujo, se enlazan los xml que contienen los beans de la aplicación a los ficheros xml contenedores de la definición de los flujos, esto a través del botón editar de la opción SWF en las propiedades del proyecto.

Con estos detalles ya es posible comenzar la elaboración de lo que será el flujo de ingreso.

Si se cuenta con los *plugins* de SWF, el *graph* y el *core*, una vez terminada la definición del flujo se puede visualizar gráficamente la secuencia de pasos del mismo aumentando así el entendimiento de todo el proceso. Dichos *plugins* están actualmente incluidos dentro del *SpringIde* de Spring, a medida que se descubren nuevas funcionalidades de esta poderosa plataforma la integración se hace más necesaria.

3.4. Módulo “Ingreso de un nuevo individuo.”

El flujo comienza con una página de presentación para darle entrada al ingreso, por tanto el estado correspondiente a este paso es un estado de vista que no recibe parámetros de entrada pues es sólo información estática para el usuario. El siguiente estado es de vista también con la información de los cupos asignados. La carga de cupos en este caso se realiza a través de una tabla dinámica que accede, mediante una url determinada que se predefine en su creación, al controlador que devuelve los recursos que espera, dicho controlador hereda de `AbstractLoadTableController` que contiene los métodos para llevar a cabo la carga de datos. Luego de seleccionar un cupo y mediante un método del `MultiactionIngreso`, con el identificador del cupo seleccionado que se ubica en un campo oculto de la página, se recuperan nombres y apellidos de quien representa al cupo para posteriormente asignárselos al objeto del formulario que se obtiene del ámbito de conversación a través del objeto `RequestContext`. Todo esto para lograr que al llegar al próximo estado de entrada de datos personales del individuo, después de crear el objeto del formulario, se ubiquen en sus respectivos campos la información asignada con anterioridad.

El estado de datos personales realiza la carga inicial de algunos elementos como son el estado, los motivos de ingreso, las instancias de tribunal y los tipos de tribunales. Los demás campos de esta pantalla se llenan dependiendo de la selección de los anteriores. Por ejemplo, los circuitos dependen del estado, la extensión de los circuitos y el número de tribunal del tipo de tribunal y los circuitos respectivamente.

Los métodos que llenan elementos desde un inicio, se ubican como acciones de entrada y posicionan en la petición el resultado con un identificador específico que es ubicado también en los controles que reciben los datos en la vista.

```
<render-actions>
      <bean-action bean="cargarDatos" method="motivoIngreso">
            <method-result name="motivos"/>
      </bean-action>
</render-actions>
```

Inmediatamente después que se active el evento siguiente, se validan los datos del formulario con el `bindAndValidate` del `formActionIngreso`, que usa el método señalado en el flujo, perteneciente este al validador especificado en las propiedades de esta clase. Antes de terminar esta transición, se accede al método requisitoria del `MultiactionIngreso` que verifica el motivo de ingreso seleccionado y en dependencia de este asigna un valor de falso o verdadero al atributo `existeRequisitoria` de la clase del objeto del formulario.

```
<transition on="siguiente" to="decisionView">
      <action bean="formActionIngreso" method="bindAndValidate">
            <attributename="validateMethod"
value="validatePage2"/>
      </action>
      <action bean="multiactionIngreso" method="requisitoria"/>
      </transition>
```

```
<transition on="success" to="decisionView"/>
```

Acto seguido, en caso de resultar satisfactoria la verificación anterior, se alcanza un estado de decisión que verifica el valor del atributo `existeRequisitoria` y determina el próximo paso en el flujo, que puede ser o el estado de vista de entrada de documentos asociados del individuo o la entrada de la requisitoria.

```
<decision-state id="decisionView">
      <if test="conversationScope.ingreso.existeRequisitoria"
      then="requisitoriaView" else="documentosView" />
</decision-state>
```

En el estado de vista de los documentos del individuo, estos se insertan en una tabla en el cliente, no se requiere validación ya que los datos agregados de la tabla son los que se envían al servidor, no los campos, que en este caso, cumplen con la tarea de completar información de los documentos. El presente estado también carga datos en el inicio, usando el mismo procedimiento de la vista de los datos personales.
Si el estado de vista de la requisitoria es alcanzado, se crea el objeto del formulario y cuando se activa el evento terminar se validan los datos de la misma forma que en la pantalla de los datos personales del individuo y se lleva a cabo el mismo procedimiento para ingresar el individuo.
Estos dos estados anteriores contienen la misma navegabilidad, tanto hacia atrás como hacia delante, ya que ocupan la misma posición en el flujo independientemente de los motivos de ingreso.
En caso de dispararse el evento terminar, si es la pantalla de documentos, se machea la transición hacia un estado de subflujo que levanta un popup luego de pasar los documentos asociados de la petición para la conversación para su posterior uso y de realizar el ingreso del individuo con toda la información almacenada en el objeto del formulario y el ámbito de la conversación, ambas acciones pertenecientes al `MultiactionIngreso`. En caso de ser el estado de vista de requisitoria, al levantarse este evento, se validan y machean los datos para posteriormente realizar el ingreso y remitirnos al mismo estado de subflujo del popup.
Como parámetro de entrada al subflujo se mapean el expediente y el nombre del ya ingresado que son situados, en el término del método donde se ingresa, en el ámbito conversacional para ser recibidos de la manera siguiente en el subflujo ubicándolos su respectivo ámbito de flujo.

```
<subflow-state  id="popupSubFlow" flow="popupFlow">
      <attribute-mapper>
            <input-mapper>
            <mapping source="conversationScope.name"
            target="name"/>
            <mapping source="conversationScope.expediente"
            target="expediente"/>
            </input-mapper>
```

```
      </attribute-mapper>
      <transition on="*" to="notificacionesView"/>
</subflow-state>
```

El nombre de los atributos que se especifican dentro del subflujo coincide con el `target` correspondiente a cada uno de los elementos de entrada declarados en el estado de subflujo del flujo ingreso.

```
<input-mapper>
<input-attribute name="name"/>
      <input-attribute name="expediente"/>
</input-mapper>
```

Dentro de este flujo que se encarga de levantar el popup, luego del estado de inicio, se entra a un estado de vista con la página que se muestra en el popup y al cerrar la misma se activa un evento que nos remite al estado final del subflujo mediante la transición correspondiente.

```
<inline-flow id="popupFlow">
      <flow>
            <input-mapper>
            <input-attribute name="name"/>
            <input-attribute name="expediente"/>
            </input-mapper>
      <start-state idref="startPopup"/>
       <view-state id="startPopup" view="expedientecreado">
      <transition on="cerrar" to="end"/>
      </view-state>
      <end-state id="end"/>
      </flow>
</inline-flow>
```

Al finalizar este subflujo, ubicado en el cuerpo del xml contenedor de la definición del flujo ingreso, se reanuda la ejecución de este flujo detenida temporalmente, accediendo a la transición descrita dentro del estado de subflujo. Esta nos remite al estado final del flujo de ingreso que selecciona una vista de notificaciones para culminar su ejecución.

A continuación se presenta el gráfico de la definición del flujo (Anexo 15), generado por el Editor de WebFlow.

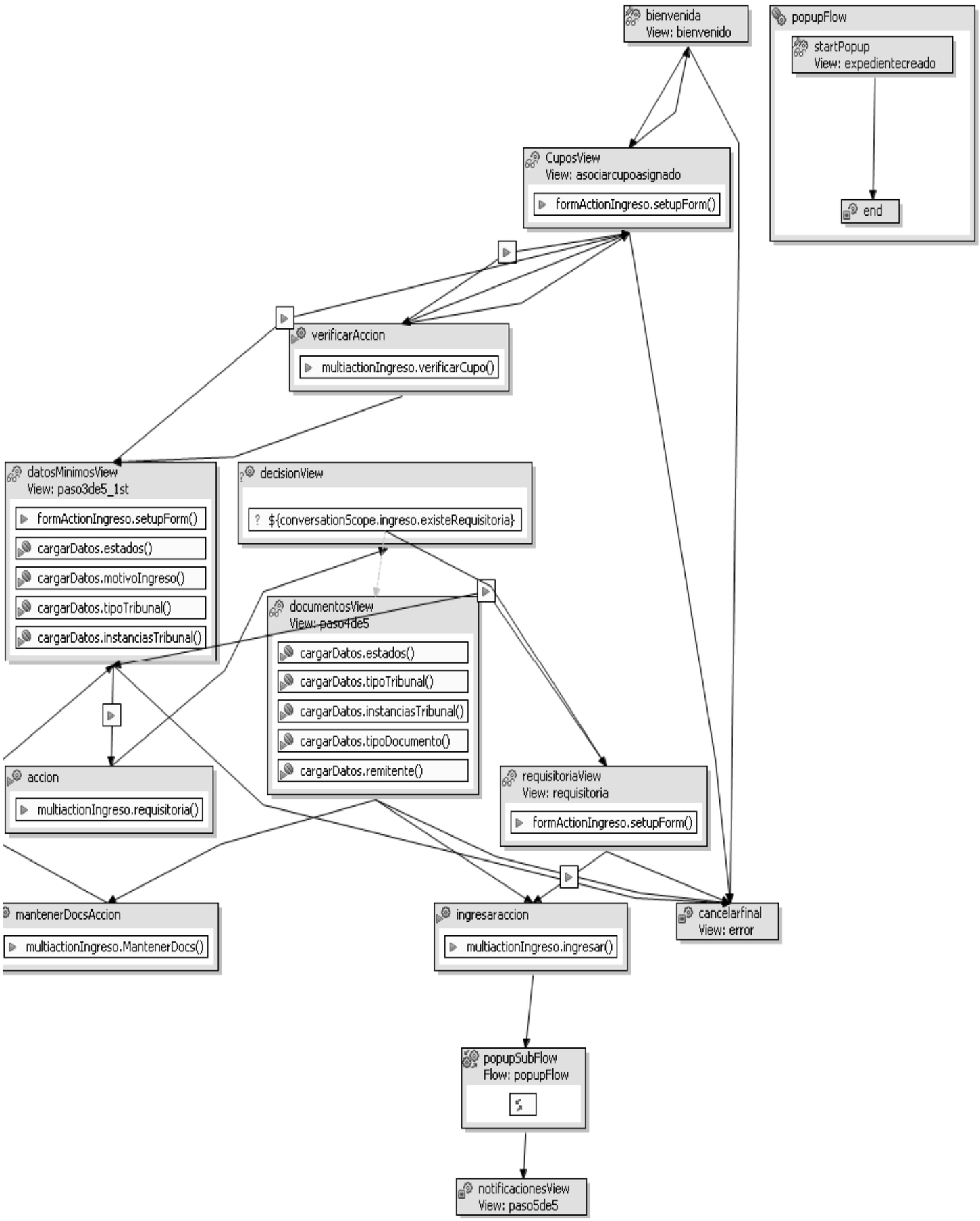

Figura 24 Flujo generado por el Editor de WebFlow.

3.5. Consideraciones

Pese a que SWF ha madurado considerablemente desde su primera versión no se puede categorizar como un *framework* del todo maduro que soporta una inmensa cantidad de funcionalidades, aunque las que provee cubren las necesidades que nos atañen en el proyecto. A medida que avanza, surgen nuevas funcionalidades que no contempla y que deben ser agregadas, existen varias versiones desde el 2005 que fue cuando surgió, todas corrigiendo los posibles fallos de las anteriores, que por no incluir ciertas facilidades, no quiere decir que su capacidad y funcionalidad se vean del todo afectadas. Por ejemplo, al configurar los recursos que hacen funcional esta plataforma, se tiene la posibilidad de asignar un número máximo de continuaciones por conversación para así evitar la denegación de servicio cuando hay varias copias de estados en memoria. En el escenario de entrada a un estado final y la respectiva publicación de la vista, si se retorna por el navegador a un punto anterior, se levantará una excepción del tipo `ContinuationNotFoundException`, ya que el repositorio basado en continuaciones que usamos, invalida la conversación estando en el estado final del flujo, por tanto, cuando el navegador trata de acceder a la url solicitada, que incluye la clave de la copia del estado, esta no se va a encontrar en el repositorio pues fue borrada automáticamente cuando se accede al estado final. Esto es manejable en esta versión 1.0.3. Haciendo uso de la que se tiene en la aplicación, con la clase `SpringSWFExceptionResolver` que hereda de `SimpleMappingExceptionResolver` se manejan todas las excepciones que pueda levantar la aplicación, incluyendo la anteriormente explicada. Requiere la entrada del conjunto de los posibles tipos de excepciones a manejar, la vista que se mostrará en caso de levantarse alguna y la definición del registro de flujo donde está la ubicación del flujo cuyas excepciones serán tratadas.

Todos los estados de vista usan la etiqueta `<render-actions>` en vez de `<entry-action>` para representar las acciones de entrada por una simple razón. SWF tiene una propiedad llamada "`alwaysRedirectOnPause`" que es verdadera por defecto, lo que significa que el estado de vista sólo va a dibujar la página en pantalla después de una redirección que obviamente refresca la ejecución del flujo. Este proceso de refrescar ejecuta las `<render-actions>` para preparar la visualización del jsp, en cambio, las `<entry-action>` no se llevan a cabo cuando se refresca el estado ya que se producen cuando ya se está en el estado. O sea, si se refresca una página, las `<render-actions>` siempre se van a ejecutar al contrario de las `<entry-action>` que no reconocen el proceso de refrescar como una orientación de ejecución ya que no se ha cambiado de estado.

Debido a esto, como nuestras acciones iniciales se traducen en la asignación de valores iniciales a los parámetros de entrada de la vista son usadas las `<render-actions>`, aunque pudiese darse el caso de asignarle información a algún estado en particular, entonces se podría utilizar la otra variante.

Por otro lado, uno de los factores fundamentales que se ha tenido en cuenta para sugerir un giro en la política de interfaz de usuario de un proyecto de la envergadura de SIGEP, y seguir el curso de la presente investigación, ha sido la reducción del tiempo de desarrollo que posibilita la utilización del *framework* de interfaz de usuario SWF.

Además de la ventaja que implica reducir el tiempo de aprendizaje de un ambiente o una herramienta determinada, es mucho más confiable un entorno que además de ser fácil de entender, permita un desarrollo ágil, consistente, y sobre todo que propicie esta agilidad por medio de su capacidad de crear componentes reutilizables que faciliten incluso el proceso de soporte y mantenimiento del producto final.

El escenario del caso de uso al cual se está tratando de dar solución como objeto demostrativo de las grandes funcionalidades de este joven *framework* y que ya se ha analizado en detalle en capítulos anteriores, como ya se sabe está realizado integrando ambos *frameworks*: Spring MVC y SWF.

Sin embargo en vistas de que el proyecto esta en proceso de desarrollo, este mismo escenario del caso de uso “Ingreso de un individuo al sistema”, ha sido desarrollado usando Spring MVC puro.

Ha resultado algo complicada la tarea de demostrar esta característica de reducción del tiempo de trabajo de SWF, puesto que el desarrollo en el marco del equipo de Proyecto SIGEP, consta de un repositorio SVN que posibilita el funcionamiento simultáneo de todos los integrantes del mismo, sin embargo, a la hora de realizar la versión con que se cuenta desarrollada con SWF, no era posible utilizar el repositorio para estos fines, pues se pondría en juego la integridad de los datos que almacena el mismo para el proyecto. O sea, las condiciones de trabajo no eran las mismas en ambos casos, ahora el trabajo se centraliza en una sola PC para mantener la organización.

Puesto que el entorno y las condiciones de trabajo no son ni siquiera similares, no se hubiese podido hacer una estimación satisfactoria del tiempo de desarrollo para ambos *frameworks* si las variables a evaluar no son las mismas.

Al finalizar la implementación del presente caso de estudio se comprendió el alcance real de esta nueva tecnología que emerge y se perfecciona constantemente. El uso de SWF con Spring MVC en este contexto, demostró en todos los sentidos la teoría que se ha estado investigando y enriqueciendo, en otras palabras, todos los problemas y dificultades planteadas fueron solucionados con este *framework* que además de su claridad y entendimiento resultó eficiente y sobre todo óptimo para llevar a cabo los flujos Web.

CONCLUSIONES

La utilización de un *framework* en el proceso de desarrollo de una aplicación Web ofrece al desarrollador innumerables ventajas; es por ello que al enfrentarse a una actividad de este tipo, la selección del *framework* a utilizar y sobre el cual se va a montar nuestra aplicación, constituye no sólo una de las fases más difíciles, sino también el punto de partida que determina la obtención de un resultado exitoso.

Al realizar este proceso de selección se debe tener en cuenta determinados factores, y sobre todo tener un conocimiento adecuado de los *frameworks* disponibles para así determinar cuáles se adaptan a las características de la aplicación en desarrollo.

En la presente investigación se realiza un recorrido por las características de los principales *frameworks* de programación de interfaz de usuario para desarrollo Web utilizando tecnología Java, profundizando en las características y funcionalidades del joven *framework* SWF.

A través de la implementación de un caso de estudio utilizando esta tecnología como complemento de Spring MVC, ha sido demostrado que SWF es capaz de resolver los problemas que presenta actualmente el resto de los *frameworks* Java con la captura del flujo Web.

La investigación realizada, así como el análisis detallado del caso de estudio, indican que la integración de Spring MVC con SWF reporta beneficios, tales como la rapidez de aprendizaje del entorno para el equipo de desarrollo, la creación de componentes reutilizables que favorezcan la inclusión de los mismos en otras partes de la aplicación en desarrollo, la creación de código relativamente fácil, legible y organizado, que favorezca el proceso de desarrollo y sobre todo el mantenimiento y soporte al producto final.

La utilización conjunta de estos dos *frameworks*, se ajusta a las necesidades y requerimientos de aplicaciones que consten de una secuencia de páginas organizada y dirigida, por lo que se considera que dada la inclinación que está tomando nuestra facultad hacia la tecnología Java, se debería prestar una especial atención a este *framework* por las razones que han quedado expuestas en la presente investigación.

RECOMENDACIONES

Se recomienda:

- Desarrollar el módulo de Ingreso de un nuevo individuo del proyecto Prisiones con la versión 1.0.4 de Spring WebFlow lanzada en los últimos tiempos para optimizar las funcionalidades de SWF.
- Usar el *framework* Spring WebFlow en proyectos que implementen módulos con más de una página por transacción del negocio.

BIBLIOGRAFÍA

1. Noticiasdot.com. [En línea] 25 de abril de 2007. http://www.noticiasdot.com/wp2/2007/04/25/java-es-el-lenguaje-de-programacion-mas-popular/.

2. **Apache.** Java en castellano. *El API Struts.* [En línea] http://www.programacion.net/java/tutorial/struts/1/.

3. **Sánchez González, Carlos.** ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras. [En línea] 28 de septiembre de 2004. http://oness.sourceforge.net/proyecto/html/ch03s02.html.

4. aNieto2k. [En línea] http://www.anieto2k.com/2007/01/13/usar-o-no-un-framework-para-nuestras-aplicaciones/.

5. **Gutiérrez;Javier J.** ¿Qué es un framework web? [En línea] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

6. JavaRanch. *JavaRanch Big Moose Saloon.* [En línea] 12 de abril de 2005. http://saloon.javaranch.com/cgi-bin/ubb/ultimatebb.cgi?ubb=next_topic&f=83&t=000149&go=older.

7. Stripes. [En línea] http://stripes.mc4j.org/confluence/display/stripes/Home;jsessionid=66408656DBC8D33C470C1CD5101B3BE1.

8. Wicket. [En línea] 22 de abril de 2007. http://wicket.sourceforge.net/Introduction.html.

9. Swing. [En línea] http://jungla.dit.upm.es/~santiago/docencia/apuntes/Swing/index.htm.

10. **Apache.** Tapestry. [En línea] http://tapestry.apache.org/.

11. —. The Apache Cocoon Project. [En línea] 24 de enero de 2007. http://cocoon.apache.org/.

12. InfoQ. *Keith Donald on Reuseable UI Flows with Spring WebFlow 1.0.* [En línea] 27 de octubre de 2006. http://www.infoq.com/news/spring-web-flow10.

13. **Ladd, Seth, y otros.** *Expert Spring MVC and Web Flows.* s.l. : Steve Anglin, 2006. 159059584X.

14. **Devijver, Steven.** Interface21. *Spring WebFlow.* [En línea] 12 de Octubre de 2005. http://www.nljug.org/pages/events/content/jfall_2005/sessions/00005/slides.pdf/.

15. **Donald, keith y Vervaet, Erwin.** The Server Side. *Spring WebFlow.* [En línea] mayo de 2005. http://www.theserverside.com/tt/articles/article.tss?l=SpringWebFlow.

16. **Begoli, Edmon.** DevX.com. *Spring WebFlow Sneak Preview: Reuse and Framework Abstraction.* [En línea] 05 de mayo de 2007. [Citado el: 16 de feb de 2007.] http://www.devx.com/webdev/Article/28041.

17. Soft-dev. *Continuations with Spring Web Flow.* [En línea] 24 de febrero de 2006. http://www.newinstance.net/blog/?p=49.

18. **Vervaet;Erwin.** Ervacon. [En línea] 25 de Octubre de 2006. http://www.ervacon.com/products/swf/tips/tip1.html.

19. Wikipedia. [En línea] [Citado el: 27 de mayo de 2007.] http://es.wikipedia.org/wiki/Programaci%C3%B3n_Orientada_a_Aspectos.

20. Wikipedia. [En línea] http://en.wikipedia.org/wiki/Finite_state_machine.

21. **Donald; Keith.** The Spring Experience. *Applied Design Patterns with Spring Web Flow.* [En línea] 2005. http://www.opensource.atlassian.com/confluence/spring/download/attachments/1090/spring-webflow-applied-patterns.ppt?version=1 .

22. **Kumar;Kishore.** JDJ. *The Promise of Handling Complex Page Navigations in Any Web Application.* [En línea] 4 de Diciembre de 2005. http://java.sys-con.com/read/131756_1.htm.

23. **Devijver;Steven .** JavaLobby. [En línea] http://www.javalobby.org/articles/spring-webflow/.

24. **Donald, Keith.** *Managing Web Conversations with Spring Web Flow.* [En línea] http://www.opensource.atlassian.com/confluence/spring/download/attachments/1090/spring-webflow-javazone.ppt .

25. **Jonson, Rod.** The Server Side. *Introduction to the Spring Framework.* [En línea] mayo de 2005. http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework.

26. Spring Framework. *Spring Web Flow 1.0.3 .* [En línea] http://static.springframework.org/spring-webflow/docs/1.0.x/api/.

27. **Fuller, Lennard y Holdorph, Cris J.** *Spring Web Flow.* [En línea] 2006. http://web.princeton.edu/sites/isapps/jasig/2006summerVancouver/Presentations/WebFlow-nojars.pdf.

28. Adictosaltrabajo. *¿Que ofrece Autentia?* [En línea] http://www.adictosaltrabajo.com/tutoriales/pdfs/struts_flow.pdf.

29. Blog and Share. [En línea] 4 de enero de 2006. http://blogandshare.blogspot.com/.

30. InfoQ. [En línea] http://www.infoq.com/news/migrating-struts2.

31. JavaHispano. [En línea] 31 de Octubre de 2004. http://weblogs.javahispano.org/page/mondelo/20041031.

32. JavaRanch. [En línea] http://www.javaranch.com/bunkhouse/JSP.jsp#159059228X.

33. Pepone's Blog. [En línea] http://peponeblog.blogspot.com/2005/12/notas-sobre-wicket.html.

34. Sourceforge. *Maverick.* [En línea] http://mav.sourceforge.net/.

35. Spring Web Flow. [En línea] Desi Massimiliano. http://www.anieto2k.com/2007/01/13/usar-o-no-un-framework-para-nuestras-aplicaciones/.

36. SpringFramework. *Spring MVC.* [En línea] http://www.springframework.org/docs/reference/mvc.html.

37. Sun Developer Network (SDN). [En línea] http://java.sun.com/j2ee/javaserverfaces/index.jsp.

38. **Apache.** The Apache Software Fundation. *Struts.* [En línea] http://struts.apache.org/.

39. **Devijver, Steven.** Spring Web Flow Examined [en línea]. *Spring Web Flow Examined [en línea].* [En línea] 05 de 2005. [Citado el: 04 de 02 de 2007.] http://www.javalobby.org/articles/spring-webflow/ .

40. **Lewis Ship, Howard.** Tapestry. *Tapestry.* [En línea] 01 de 2007. [Citado el: 04 de 02 de 2007.] http://tapestry.apache.org.

41. **Willard, Wendy.** *Fundamentos de programación en HTML.* [ed.] Eder Mauricio Hernandez B. 2003. 958-41-0266-4.

42. **Zeldman, Jeffrey.** *Diseño con estándares Web.* [ed.] Juan Ignacio Luca de Tena. Madrid : Ediciones Anaya Multimedia, 2004. 84-415-1608-1.

43. **Dessi, Massimiliano.** Java User Group Sardegna Onlus. *Spring WebFlow.* [En línea] http://www.jugsardegna.org/vqwiki/jsp/Wiki?SpringWebFlow.

44. **Carrasco Montenegro., Alberto.** Interface21. *Spring WebFlow.* [En línea] http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=springWebFlow.

45. **Donald, Keith y Vervaet, Erwin.** TheServerSide.com. *Spring Web Flow.* [En línea] 05 de 2005. http://www.theserverside.com/tt/articles/article.tss?l=SpringWebFlow.

ANEXOS

Anexo 1

```
<action-state id="ejecutarBusqueda">
        <bean-action bean="ServicioBusqueda" method="ejecutarBusqueda">
                <method-arguments>
                    <argument expression="${flowScope.criterio}"/>
                </method-arguments>
                <method-result name="resultados"/>
          </bean-action>
            <transition on="yes" to="entrarDetallesCompra"/>
            <transition on="no" to="ordenCompra"/>
      </action-state>
```

Anexo 2

```
<action-state id="subirFichero">
       <action bean="accionSubida" method="subirFichero"/>
<transition on="success" to="seleccionarFichero">
       <set attribute="ficheroSubido" scope="flash" value="true"/>
 </transition>
</action-state>
```

Anexo 3

```
<view-state id="desplegar" view="formulario">
               <render-actions>
                <action bean="formAction" method="setupForm"/>
            <action bean="formAction" method="loadFormReferenceData"/>
         </render-actions>
        <transition on="submit" to="salvarDatos">
               <actionbean="formAction" method="bindAndValidate"/>
        </transition>
 </view-state>
```

Anexo 4

```
<decision-state id="requiereDatos">
<if    test="${flowScope.orden.necesitaDatos}"    then="entrarDetalles"
else="enviarDatos"/>
```

```
</decision-state>
```

Anexo 5

```
<subflow-state id="detalles" flow="detallesFlow">
 <attribute-mapper>
   <input-mapper>
    <mapping source="flowScope.orden.idElemento" target="shipping"/>
   </input-mapper
 </attribute-mapper>
 <transition on="terminar" to="listaObjetos"/>
 </subflow-state>
```

Anexo 6

Estado final con atributos de salida.

```
<end-state id="terminar">
     <output-mapper>
         <output-attribute name="resultado"/>
     </output-mapper>
</end-state>
```

Estado final que redirecciona a la ejecución de flujo

```
<end-state id="terminar" view="flowRedirect:busquedaFlow"/>
```

Estado final que redirecciona a una URL externa

```
<end-state id="terminar"
view="externalRedirect:/ordenes/${flowScope.order.id}"/>
```

Anexo 7

```
<global-transitions>
            <transition on="eventoGlobal1" to="estado1"/>
            <transition on=" eventoGlobal2" to="estado2"/>
</global-transitions>
```

Anexo 8

```
FlowDefinition definicion = ...
FlowExecutionFactory fabrica = ...
FlowExecution execution = fabrica.createFlowExecution(definicion);
```

Anexo 9

```
<bean name="/controlador.htm"
class="org.springframework.Webflow.executor.mvc.FlowController">
<property name="flowExecutor" ref="flowExecutor"/>
</bean>
```

Anexo 10

Comenzando una ejecución de flujo mediante formulario.

```
<a href="controlador.htm?_flowId=flujo">Iniciar flujo</a>
```

Comenzando una ejecución de flujo mediante parámetros.

```
<form action=" controlador.htm" method="post">
<input type="submit" value="Iniciar flujo"/>
<input type="hidden" name="_flowId" value="flujo">
</form>
```

Reactivando una ejecución de flujo mediante parámetros.

```
<a
href="controlador.htm?_flowExecutionKey=${flowExecutionKey}&_eventId=su
bmit">
Submit
</a>
```

Reactivando una ejecución de flujo mediante formulario.

```
<form action="controlador.htm" method="post">
<input type="hidden" name="_flowExecutionKey"
value="${flowExecutionKey}">
<input type="hidden" name="_eventId" value="submit"/>
<input type="submit" class="button" value="Submit">
</form>
```

Reactivando una ejecución de flujo mediante múltiples botones en el formulario.

```
<form action="controlador.htm" method="post">
<input type="hidden" name="_flowExecutionKey"
value="${flowExecutionKey}">
<input type="submit" class="button" name="_eventId_submit"
 value="Submit">
<input type="submit" class="button" name="_eventId_cancel"
value="Cancel">
```

```
</form>
```

Refrescando la ejecución de un flujo

```
<a
href="flowController.htm?_flowExecutionKey=${flowExecutionKey}">Refresc
ar</a>
```

Anexo 11

```
public interface FlowExecutor {
ResponseInstruction launch(String flowDefinitionId, ExternalContext
context);
ResponseInstruction resume(String flowExecutionKey, String eventId,
ExternalContext context);
ResponseInstruction refresh(String flowExecutionKey, ExternalContext
context);
}
```

Anexo 12

```
<flow:executor id="flowExecutor" registry-ref="flowRegistry"
repository-type=”simple” />
<flow:registry id="flowRegistry">
<flow:location path="/WEB-INF/flows/**/*-flow.xml"/>
</flow:registry>
```

Anexo 13

```
<flow:executor id="flowExecutor" registry-ref="flowRegistry">
<flow:repository type="continuation" max-conversations="5" max-
continuations="30" conversation-manager-ref="conversationManager"/>
 </flow:executor>

<bean id="conversationManager"
class="ejemplo.ManejadorEstadoConversacion"/>
```

Anexo 15

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http/www.springframework.org/schema/webflow"
xmlnssi="http/www.w3.org/2001/XMLSchema-instance"
xsichemaLocation="http/www.springframework.org/schema/webflow
http/www.springframework.org/schema/webflow/spring-webflow-
1.0.xsd">

<start-state idref="bienvenida" />
```

```
<view-state id="bienvenida" view="bienvenido">
<transition on="siguiente" to="CuposView" />
<transition on="cancelar" to="cancelarfinal" />
</view-state>

<view-state id="CuposView" view="asociarcupoasignado">
<render-actions>
<action bean="formActionIngreso" method="setupForm" />
</render-actions>
<transition on="siguiente" to="verificarAccion" />
<transition on="atras" to="bienvenida" />
<transition on="cancelar" to="cancelarfinal" />
</view-state>

  <action-state id="verificarAccion">
<action bean="multiactionIngreso" method="verificarCupo" />
<transition on="yes" to="datosMinimosView" />
<transition on="no" to="CuposView">
<action bean="multiactionIngreso" method="mensaje" />
</transition>
<transition on="success" to="CuposView" />
</action-state>

<view-state id="datosMinimosView" view="paso3de5_1st">
<render-actions>
<action bean="formActionIngreso" method="setupForm" />
<bean-action bean="cargarDatos" method="estados">
<method-result name="estados" />
</bean-action>
<bean-action bean="cargarDatos" method="motivoIngreso">
<method-result name="motivos" />
</bean-action>
<bean-action bean="cargarDatos" method="tipoTribunal">
<method-result name="tipoTribunal" />
</bean-action>
<bean-action bean="cargarDatos"
method="instanciasTribunal">
<method-result name="instancias" />
</bean-action>
</render-actions>
<transition on="siguiente" to="accion">
<action bean="formActionIngreso" method="bindAndValidate">
<attribute name="validatorMethod" value="validatePage2" />
</action>
</transition>
<transition on="atras" to="CuposView">
<action bean="formActionIngreso" method="bind" />
</transition>
<transition on="cancelar" to="cancelarfinal" />
</view-state>
```

```
<action-state id="accion">
<action bean="multiactionIngreso" method="requisitoria" />
<transition on="yes" to="decisionView" />
</action-state>

<decision-state id="decisionView">
<if test="${conversationScope.ingreso.existeRequisitoria}"
then="requisitoriaView" else="documentosView" />
</decision-state>

<view-state id="documentosView" view="paso4de5">
<render-actions>
<bean-action bean="cargarDatos" method="estados">
<method-result name="estados" />
</bean-action>
<bean-action bean="cargarDatos" method="tipoTribunal">
<method-result name="tipoTribunal" />
</bean-action>
<bean-action bean="cargarDatos"
method="instanciasTribunal">
<method-result name="instancias" />
</bean-action>
<bean-action bean="cargarDatos" method="tipoDocumento">
<method-result name="tipoDoc" />
</bean-action>
<bean-action bean="cargarDatos" method="remitente">
<method-result name="remitentes" />
</bean-action>
</render-actions>
<transition on="atras" to="mantenerDocsAccion" />
<transition on="terminar" to="ingresaraccion" />
<transition on="cancelar" to="cancelarfinal" />
</view-state>

<action-state id="mantenerDocsAccion">
<action bean="multiactionIngreso" method="MantenerDocs" />
<transition on="success" to="datosMinimosView" />
</action-state>

<view-state id="requisitoriaView" view="requisitoria">
<render-actions>
<action bean="formActionIngreso" method="setupForm" />
</render-actions>
<transition on="terminar" to="ingresaraccion">
<action bean="formActionIngreso" method="bind" />
</transition>
<transition on="atras" to="datosMinimosView">
<action bean="formActionIngreso" method="bind" />
</transition>
<transition on="cancelar" to="cancelarfinal" />
</view-state>
```

```
<action-state id="ingresaraccion">
<action bean="multiactionIngreso" method="ingresar" />
<transition on="yes" to="popupSubFlow" />
</action-state>

<subflow-state id="popupSubFlow" flow="popupFlow">
<attribute-mapper>
<input-mapper>
<mapping source="conversationScope.name" target="name" />
<mapping source="conversationScope.ingreso"
target="ingreso" />
<mapping source="conversationScope.expediente"
target="expediente" />
</input-mapper>
</attribute-mapper>
<transition on="*" to="notificacionesView" />
</subflow-state>

<end-state id="cancelarfinal" view="error" />

<end-state id="notificacionesView" view="paso5de5" />

<importresource="sigep-controlpenal-ingresosegresos-beans-
ingreso-flow.xml" />

<inline-flow id="popupFlow">
<flow>
<input-mapper>
<input-attribute name="name" />
<input-attribute name="expediente" />
<input-attribute name="ingreso" />
</input-mapper>
<start-state idref="startPopup" />

<view-state id="startPopup" view="expedientecreado">
<transition on="final" to="end" />
</view-state>
<end-state id="end" />
</flow>
</inline-flow>
</flow>
```

Tablas

Tabla 1

Nombre de la propiedad	Descripción	Cardinalidad	Valor por defecto
`Id` (identificador)	Identificador de la definición del flujo, único para los demás flujos de una aplicación.	1	
`Attributes` (Atributos)	Atributos adicionales sobre el flujo.	0..*	Ninguno
`States` (Estados)	Pasos en el flujo	1..*	
`startState` (Estado de inicio)	Punto de partida del flujo	1	
`Variables`	Conjunto de variables del flujo para una vez iniciado el flujo, crear una ejecución del mismo.	0..*	Vacío
`inputMapper` (Servicio de entrada)	Servicio responsable de mapear las entradas suministradas por el flujo cada vez que la ejecución del flujo es iniciada.	0..1	Null
`startActions` (Acciones de inicio)	Lista de acciones a ejecutar una vez	0..*	Vacío

	iniciado el flujo.		
`endActions` Acciones finales	Lista de acciones a ejecutar cada vez que el flujo culmina.	0..*	Vacío
`outputMapper` (Servicio de salida)	Servicio responsable de mapear la salida del flujo una vez que este termina su ejecución.	0..1	Null
`globalTransitions` (Transiciones globales)	Grupo de transiciones compartidas por todos los estados en el flujo.	0..*	Vacío
`exceptionHandlers` (Manejadores de excepciones)	Ordenado grupo de manejadores a ser aplicados cuando es lanzada una excepción dentro de un estado del flujo.	0..*	Vacío
`inlineFlows` (Flujo en línea)	Conjunto de flujos interiores que serán llamados como subflujos.	0..*	Vacío

Tabla 2

Nombre de la propiedad	Descripción	Cardinalidad	Valor por defecto
`Id` (identificador)	Identificador del estado, único de la definición del flujo.	1	

`owner` (Dueño)	Definición de flujo contenedor.	1	
`attributes` (Atributos)	Atributos adicionales de estado.	0..*	Ninguno
`entryActions` (Acciones de entrada)	Lista de acciones a ejecutar cada vez que se inicia el estado.	0..*	Vacío
`exceptionHandlers` (Manejadores de eventos)	Conjunto de manejadores a ser invocados cuando una excepción es lanzada dentro del estado.	0..*	Vacío

Tabla 3

Inicio de un flujo	`<start-actions/>`
Una vez que un estado inicia	`<entry-actions/>`
En una transición	`<action/>`
Final de un estado de transición	`<exit-actions/>`
Cuando se selecciona una vista	`<render-actions/>`
Fin de sesión de flujo	`<end-actions/>`

Tabla 4

Capa de ejecución del core[19].

Es la capa más baja de SWF, define y ejecuta el flujo central. Es altamente estable sin dependencias con otra capa.

Nombre del subsistema	Descripción	Paquetes	Interfaces del subsistema	Dependencias internas con subsistemas
Core (Centro)	Es fundacional, común a muchos y usado por otros subsistemas. Contiene el analizador gramatical de expresiones por defecto (OGNL-based).	core, core.collection	Ninguna	Ninguna
Util	Utilidades de bajo nivel usadas por todas las partes del sistema.	util	Ninguna	Ninguna

[19] Centro. Núcleo.

FlowDefinition (Definición de flujo)	Abstracciones centrales para el modelado de las definiciones de los flujos. Incluyen FlowDefinition, StateDefinition que forman el dominio de lenguaje para describir los flujos	definition	FlowDefinition	Core
Flow Definition Registry (Registro de definición de flujo)	Brinda soporte para trabajar con registros de definición de flujo.	definition.registry	FlowDefinition Registry FlowDefinition Locator	Core, FlowDefinition
External Context (Contexto Externo)	Suministra acceso normalizado a un ambiente de cliente que ha llamado dentro de SWF.	context, context.servlet, context.portlet	ExternalContext	Core
Conversation (Conversación)	Maneja la creación y la limpieza de estado conversacional.	conversation, conversation.impl	ConversationManager	Core, Util,

				External Context
Flow Execution (Ejecución de flujo)	Abstracciones estables en tiempo de ejecución que definen el modelo de ejecución de la definición del flujo.	execution, execution.support, execution.factory	FlowExecution	Core, External Context, FlowDefinition
Flow Execution repository (Repositorio de ejecución de flujo)	Persiste ejecuciones de flujo en pausa además de una única petición dentro del servidor.	execution.repository, execution.repository.support, execution.repository.continuation	FlowExecutionRepository	Core, FlowDefinition, Conversation ,Flow Execution
Action (Acción)	Implementaciones de acciones reusables	action, action.portlet	Ninguna	Core, Util, FlowDefinition, External Context, Flow Execution
Capa del motor de ejecución				

Define una implementación del centro de una interface de programa de aplicación(API) de la ejecución de un flujo, formando las bases de la máquina de estado o implementación. Se considera más volátil y contiene implementaciones especificas de abstracciones de ejecución estables. Depende de Centro de ejecución

Nombre del subsistema	Descripción	Paquetes	Interfaces del subsistema	Dependencias internas
`Engine Implementation` (Implementación de la máquina)	La implementación de la máquina de ejecución del flujo basado en una máquina de estado finita.	`engine`, `engine.support`, `engine.impl`	Ninguna	Ninguna
`Flow Definition Builder` (Constructor de la definición del flujo)	Abstracciones usadas en el momento de configuración para construir y ensamblar el ejecutable de definición de flujo a través de la implementación de la máquina. Los flujos son generalmente definidos en ficheros XML.	`engine.builder`, `engine.builder.xml`	`FlowBuilder`	Implementación de la máquina, Spring Beans 1.2.7, Spring Context 1.2.7, `builder.xml` requires JDK 1.5 or Xerces for XSD support

Capa de Prueba

Contiene artefactos del flujo para pruebas de unidad y ejecuciones de flujo para pruebas de sistema. Depende de la máquina de ejecución y del Centro de ejecución.

Nombre del subsistema	Descripción	Paquetes	Interfaces del subsistema	Dependencias internas
`Engine Artifact Unit Test Support` (Soporte para pruebas de unidad a artefactos)	Soporte para implementaciones para pruebas de unidad tales como acciones en aislamiento.	`test`	Ninguna	JUnit 3.8.1
`Flow Execution Test Support` (Soporte de pruebas de flujo de ejecución)	Soporte para probar ejecución de flujos.	`test.execution`	Ninguna	Spring Beans 1.2.7, JUnit 3.8.1

Capa ejecutora

Capa de gran estabilidad para conducir y coordinar la ejecución de la definición de los flujos. Es desacoplada desde la capa más volátil, la Implementación de la máquina. Depende de: Centro de ejecución

Nombre del	Descripción	Paquetes	Interfaces del subsistema	Dependencias internas

subsistema				
Core	Capa estable, brinda abstracciones genéricas de la ejecución del flujo y soporte.	executor, executor.support	FlowExecutor	Ninguna
Spring MVC	La integración entre Spring MVC y SWF.	executor.mvc	Ninguna	Core, Spring Web MVC 1.2.7, Portlet MVC requires Spring 2.0
Struts	La integración entre Struts y SWF.	executor.struts	Ninguna	Core, Struts 1.1
Java Server Faces (JSF)	La integración entre Java Server Faces y SWF.	executor.jsf	Ninguna	Core, JSF 1.0

Capa de configuración del sistema

Es la capa más alta de la jerarquía para configurar todo el sistema de SWF con el objetivo de usarlo dentro de la aplicación. Como la capa más alta, depende de la mayoría, como son: Ejecución, Centro de Ejecución, Máquina de implementación.

Nombre del subsistema	Descripción	Paquetes	Interfaces del subsistema	Dependencias internas
`Spring Configuration Support` (Soporte de configuración de Spring)	Para configurar SWF usando Spring 1.x y 2.x	`config`	Ninguna	Spring Beans 1.2.7, `spring-Webflow-config-1.0` XSD support requires Spring 2.0

Tabla 5

Estado de Sesión	Descripción	Evento
Creado	La sesión de flujo ha sido creada pero aún no esta activa.	Creación de una nueva ejecución de flujo.
Activo	La sesión de flujo se está ejecutando.	Cuando se inicia el flujo.
En pausa	La sesión de flujo esta esperando entrada de datos del usuario.	Cuando se muestra la vista.
Suspendido	La sesión de flujo es aún válida pero no está activa.	Cuando se crea un nuevo subflujo, la sesión de flujo existente queda suspendida.
Terminado	La sesión de flujo ya no está activa.	Cuando se alcanza el estado final del flujo.

GLOSARIO DE TÉRMINOS.

Continuaciones: Una continuación es una salva de un estado específico de un programa, una fotografía instantánea de un estado ejecutable de un programa en cualquier momento dado. De manera que sea posible restablecer este estado y reiniciar la ejecución del programa a partir de ese punto.

Repositorio: Sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.

Servlets: Los servlets son objetos que corren dentro del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. También podrían correr dentro de un servidor de aplicaciones (ej: OC4J Oracle) que además de contenedor para servlet tendrá contenedor para objetos más avanzados como son los EJB (Tomcat sólo es un contenedor de servlets). El uso más común de los servlets es generar páginas Web de forma dinámica a partir de los parámetros de la petición que envíe el navegador Web.

Request: El objeto Request nos devuelve informaciones del usuario que han sido enviadas por medio de formularios, por URL o a partir de cookies (se verá de qué se tratan seguidamente). También nos informa sobre el estado de ciertas variables del sistema como pueden ser la lengua utilizada por el navegador, el número IP del cliente...

Sesión: Una sesión es el tiempo que transcurre entre el momento que el usuario se identifica en el sistema y, bien por falta de actividad, bien por desconexión voluntaria, sale del mismo.

Acoplamiento: El acoplamiento hace referencia a las relaciones que tienen los objetos entre sí dentro de un sistema.

Transiciones: Estructuras intermediarias entre estados de un flujo.

Framework: En general con este término se hace referencia a una estructura de software formada por componentes personalizables y además intercambiables, útiles en el desarrollo de una aplicación. Un *framework* se podría definir como una aplicación genérica incompleta y configurable a la cual se le pueden incluir las piezas finales para realizar una aplicación en concreto.

Framework Web: Un *framework* Web se define como un conjunto de componentes (entiéndase clases Java, descriptores, archivos de configuración en XML) que se unen para componer un diseño reutilizable para facilitar y agilizar la realización de sistemas WEB.

Beans: Es una clase de Java.

Widget: Es una pequeña aplicación presentada en archivos o ficheros pequeños ejecutados por un motor de widgets o Widgets Engine. Tiene entre sus objetivos dar acceso fácil a funciones frecuentemente usadas y sobre todo proveer información visual. Un widget puede hacer todo lo que la imaginación pueda crear e incluso interactuar con servicios y con información distribuida en Internet

Ingeniería inversa: Se denomina ingeniería inversa del software a la actividad que se ocupa de descubrir cómo funciona un programa, de cuyo código fuente no se dispone, hasta el punto de poder modificar ese código.

Aplicaciones Standalone: Ordenador independiente que funciona sin necesidad de equipo agregado y sin estar conectado a una red.

Programación Orientada a Aspectos: Es un paradigma de programación relativamente reciente con la intención de ofrecer una modularización adecuada en una aplicación y contribuir así a una óptima separación de conceptos (19)

DataBinder: Evalúa expresiones de datos enlazados en tiempo de ejecución.

BeanFactory: Es la interfaz raíz para acceder al contenedor de beans. Es la vista cliente básica de un contenedor de beans. Maneja los beans de cualquier naturaleza usando cualquier tipo de facilidad de almacenamiento.

Validator: Interfaz que se encarga de validar los datos una vez enviados al servidor.

Application Context: Se fundamenta en el BeanFactory y agrega funcionalidad como la integración a la Programación Orientada a Aspectos y propagación de eventos entre otras mejoras.

Dispatcher Servlet: Controlador frontal adonde se dirige inicialmente la petición en Spring MVC y que posteriormente es enviada hacia otros componentes para su ejecución.

Inversión de control: Patrón de diseño de programación que especifica una capa de abstracción entre componentes.

Inyección de dependencia: Se refiere a la separación entre la construcción de un objeto y su utilización.

Arquitectura: Una arquitectura puede ser descompuesta recursivamente en: partes que interactúan entre si por medio de interfaces, relaciones que conectan las partes, y restricciones para ensamblar las partes.