Universidad Nacional De Rosario (UNR)
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Tesis de grado para Licenciatura en Ciencias de la Computación

Documentación de Estilos Arquitectónicos en Sistemas Web Colaborativos Sensibles al Contexto

Iván Rizzo
Legajo: R-2566/6

Director: Lic. Alejandro Sartorio

Miembros del jurado: Jurado 1
Jurado 2
Jurado 3

Tesis presentada en la FCEIA - UNR, en cumplimiento parcial de los requisitos para optar al título de:

Licenciado en Ciencias de la Computación

Junio, 2011

Resumen

En este trabajo se realiza una documentación sobre la arquitectura de un Framework Web Colaborativo Sensible al Contexto compuesto por la fusión de cuatro subsistemas. En el desarrollo de dicha documentación se describen como componentes de primera clase a los mecanismos de vinculación entre subsistemas representados en el diseño de la arquitectura. Además, se desarrolla un estudio sobre variantes de diseño de sistemas estratificados utilizado en la resolución de las problemáticas de diseño y documentación que surgieron en este dominio de aplicación. De esta manera, se proporciona como resultado final una contribución sobre la documentación arquitectónica del framework web colaborativo Sakai para su comunidad de diseñadores y desarrolladores.

Agradecimientos

Quiero agradecer la dirección y el apoyo del Lic Alejandro Sartorio, sin el cual esta tesina no se hubiese emprendido.

A mis padres que sacrificaron mucho para hacer esto posible. Por enseñarme de chico el valor de la educación y perseverancia. Por escucharme cuando necesitaba un consejo y acompañarme cuando lo necesitaba. A ellos más que a nadie.

A mis amigos, a todos. Porque fueron la fuente de distracción y apoyo que me acompaño en este largo camino.

A los profesores, que guiaron mis estudios y me formaron como profesional.

A todos aquellos que de manera directa o indirecta, influyeron, hicieron su aporte o ayudaron en este trabajo.

Índice general

Índice de figuras

CAPÍTULO 1

Introducción

La implementación de plataformas colaborativas constituyen unos de los medios más versátiles para el uso en actividades académicas. Como ejemplo de este tipo de aplicaciones se pueden mencionar: Sakai, WebCT, BlackBoard, e-ducativa, Plataforma Mediáfora, Dokeos, OfficeManager, Moodle, Nexus, ILIAS, Claroline.

Su constante evolución, crecimiento y adaptación permiten tener cada vez mejores prestaciones y servicios. El eficiente uso de estas plataformas implican tener sólidos conocimientos técnicos para su instalación, mantenimiento y desarrollo. Al mismo tiempo se debe contar con mínimas habilidades para la creación de los distintos espacios de trabajos y definir las metodologías de uso.

En el marco de los análisis efectuados y teniendo en cuenta experiencias del grupo de trabajo en el que esta tesis fue desarrollada, se sostiene que la incursión en proyectos científicos con desarrollos "open source" brindan propuestas consolidadas de diseño y desarrollo de entornos colaborativos Web para educación, orientado a herramientas que se implementan a través de servicios comunes (servicios bases). Por ejemplo, existen frameworks orientados a portales donde el servicio de edición de mensajes es utilizado en las herramientas Foro, Anuncio, Blog, etc. Más aún, otras de las características salientes es la versatilidad para su extensión y/o configuración. En efecto, es posible alterar ciertas configuraciones en tiempo de ejecución, por ejemplo, instrumentar una nueva funcionalidad en un servicio base determinado.

En la actualidad, teniendo en cuenta el contexto de nuestra región se evidencia la necesidad de promover el estudio de técnicas de Ingeniería de Software adaptadas a estos tipos de desarrollos, partiendo de los ámbitos académicos-científicos y su posterior transferencias a las industrias locales interesadas. En esta tesis se plantea la construcción

de una documentación de estilos arquitectónicos para un determinado tipo de sistema conformado por cuatro tipos diferentes de subsistemas.

Para el abordaje de este desafío planeado se propone articular investigaciones a partir de los resultados obtenidos en el manejo de sistemas colaborativos para la construcción y estudios de mejores técnicas en documentación de estilos arquitectónicos. Se tendrán en cuenta casos de estudios sobre aplicaciones Web, basados en un framework colaborativo con propiedades de sensibilidad al contexto [1].

1.1. Contexto

Este trabajo se enmarca dentro de un proyecto general de investigación denominado "Ingeniería de software para el diseño y desarrollo de sistemas Web colaborativos con propiedades context-aware, adaptativo, con coordinación de contratos basados en arquitecturas dinámicas" [10]. Cuyos objetivos generales pretenden iniciar investigaciones en el manejo de sistemas colaborativos en la construcción y estudios de mejores técnicas de Especificación, Diseño, Modelado, Testing, Formalización y Documentación, como aportes científicos en el campo de la Ingeniería de Software.

El proyecto se encuentra radicado en el Centro de Altos Estudios en Tecnologías Informática (CAETI sede Rosario) [a], pertenecientes a una de las 5 líneas de investigación [b] denominada "Nuevas Tecnologías para Internet". Las producciones de esta línea abordan la problemática del diseño y desarrollo partiendo del estudio de las nuevas tecnologías.

Para mayor información sobre el proyecto ingresar en la plataforma de investigación del CAETI Rosario a través del blog: http://caetirosario.blogspot.com/

1.1.1. Presentación del caso de estudio

Los avances en las principales comunidades científicas sobre desarrollos de herramientas Web colaborativas, permiten participación en varios niveles dentro de una activa comunidad de educadores, líderes institucionales y desarrolladores inspirados en las actividades de enseñanza, aprendizaje e investigación. Particularmente, los diseñadores y desarrolladores del proyecto Sakai [c] trabajan conjuntamente con docentes y estudiantes profesionales de universidades internacionales (ejemplos de algunas de ella involucradas en este proyecto: Indiana University, University of Michigan, Yale University, Stanford University, Universidad Politécnica de Valencia, Universidad del Valle de Guatemala), promoviendo el acortamiento de las distancias entre las necesidades del usuario final y

[a] `http://caeti.uai.edu.ar/Investigacion/`

[b] Líneas de investigación: Sociedad del Conocimiento y Tecnologías Aplicadas a la Educación. Algoritmos y software. Seguridad informática y telecomunicaciones. Nuevas tecnologías para Internet. Automatización y robótica.

[c] `http://sakaiproject.org`

el software. El mayor flujo de las actividades colaborativas se concentran en la lista de e-mails, wiki, foros, etc. Los miembros de estas comunidades presentan todo tipo de perfiles académicos e institucionales.

Las tareas diseñadas en este proyecto son propuestas como continuaciones de trabajos a partir de los avances en investigación y desarrollos efectuados por los miembros investigadores. Partiendo de experiencias en el dictado de cursos, dirección de trabajos y publicaciones sobre Ingeniería de Software y Análisis de Sistemas[15; 16]. También, se tienen en cuenta resultados y recorridos de experiencias conjuntas en los proyectos: "Técnicas De Ingeniería De Software Aplicadas Al Dispositivo Hipermedial Dinámico" (1ing252 - Resol C.S. 945/2008). (Cifasis: Conicet-Upcam-Unr) y "Obra Abierta: Dispositivos Hipermediales Dinámicos Para Educar e Investigar" (1ing253 - Resol C.S. 945/2008). (Cifasis: Conicet-Upcam-Unr) [d] sobre la inyección de propiedades de coordinación de contratos sensibles al contexto al framework colaborativo Sakai[17].

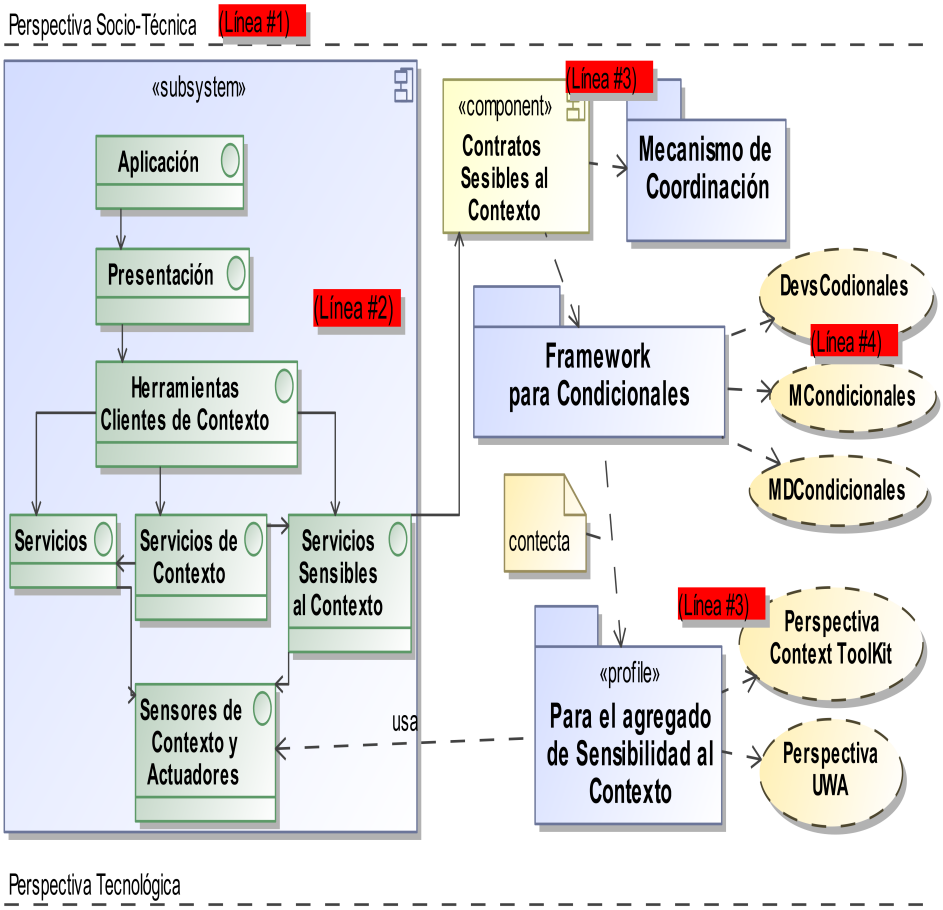


Figura 1.1: Arquitectura conceptual del FWCsc

La figura 1.1 presenta parte de la arquitectura conceptual de un Framework Web Colaborativo sensible al contexto (FWCsc). En el diagrama se muestra una composición de cuatro subsistemas.

[d] http://www.cifasis-conicet.gov.ar/index.php?sr=grupos/sanmartin.php

El primer subsistema representa la arquitectura independiente del framework web colaborativo a través de la estratificación de las siguientes capas:

El Framework Sakai está diseñado según una arquitectura de cuatro capas: La capa de aplicación, presentación, herramientas y servicios. Nuestra propuesta consiste en envolver los servicios del núcleo Sakai mediante un mecanismo de coordinación de contratos. De esta manera se altera el diseño original del framework, agregando y modificando capas que nos permitan la inyección de información de contexto y el agregado de una nueva pieza de software (un contrato) con propiedades de sensibilidad al contexto. Con este propósito, fue modificado el diseño de la capa de servicios original, mediante una división en tres partes:

- Servicios Originales: Pertenecientes al núcleo del framework original, no afectados con el agregado del mecanismo de coordinación de contrato.
- Servicios de Contexto: Permite a clientes el acceso a entidades, asignar, obtener y subscribir cambios en la información de contexto de las entidades.
- Servicios con coordinación de contratos (Servicios CSC): Servicios base del núcleo del framework Sakai modificados para poder efectuar la envoltura en los mecanismos de coordinación.

Mediante la división del estrato servicios se puede interpretar a los Servicios CSC como una nueva arquitectura basada en sistemas estratificados[19]. Entonces, esta composición se efectúa por el estrato de coordinación de contratos y el estrato de cómputo. La capa de cómputo estará compuesta por módulos de implementación (ej., servicios Sakai previos a la incorporación de contratos). Mientras que la capa de coordinación estará compuesta por módulos específicos de coordinación, patrones tipo proxy y contratos.

La implementación del los Servicios CSC se realizarán utilizando un patrón de diseño de coordinación de contratos ("Coordination Contracts Design Pattern") tomando como referencia la propuesta de Fiadeiro[13; 18]. Este patrón esta basado en el patrón de diseño "proxy" (o "Surrogate")[8]. Por un lado provee una interfaz específica ("SubjectInterface"), como una clase abstracta, para cada componente. Esta interfaz esta conectada al programa real ("SubjectBody") a través de un proxy dinámico reconfigurable. Por otra parte, soporta la reconfiguración dinámica del código ejecutado por medio de solicitud de operaciones a través del "proxy".

El segundo subsistema está compuesto por una componente contrato y su correspondiente mecanismo de coordinación. En este caso la coordinación de contrato se define como:

En términos generales, la coordinación de contratos es una conexión establecida entre un grupo de objetos (en nuestras consideraciones los participantes serían un objeto cliente y un determinado servicio), donde reglas, normas y restricciones (RNR) son superpuestas entre los actores participantes, estableciendo con un determinado grado de control las formas de interrelación (o interacción).

El tipo de interacciones establecidas entre las partes es más satisfactoria que las que se pueden lograr con UML o lenguajes similares (orientados a objetos) debido a que éstas contienen un mecanismo de superposición donde se toman como argumento los contextos. Cuando un objeto cliente efectúa una llamada a un objeto suministro, el contrato "intercepta" la llamada y establece una nueva relación teniendo en cuenta el contexto del objeto cliente, el objeto servidor e información relevante (respecto de la relación) adquirida y representada como contexto del entorno. Como condición necesaria, la implementación de los contratos no debe alterar el diseño y funcionalidad en la implementación de los objetos.

El tercer subsistema corresponde a un framework implementativo "context-awareness" que permite integrarse con algunas de las componentes del primer subsistema para la recolección del censado de información de contexto. Luego, dicha información es procesada a través de mecanismos que permitirán incorporarles propiedades de sensibilidad al contexto. Su configuración fue resuelta a partir de las ideas fundadoras del trabajo de Dey sobre el Context ToolKit[43] y el proyecto UWA[2].

El cuarto subsistema lo compone un nuevo modelo pensado para el diseño e implementación de condicionales que puedan ser utilizados en la composición de reglas de contratos. La principal idea de esta propuestas es estandarizar soluciones y brindar información necesaria en la creación de condicionales, donde su valores de verdad deban ser calculados a través de sistemas externos (por ejemplo, el tercer subsistema de la figura1.1). En este sentido, los tipos de condicionales serán abstraídos en modelos que comprendan cálculos a partir de métricas, estructuras y simulación de eventos discretos. También se puede ver a este subsistema como integrador (conector) entre el subsistema de coordinación de contrato (segundo subsistema) y el sensible al contexto (tercer subsistema).

1.2. Motivación

Este trabajo fue desarrollado con el objetivo de aportar a la comunidad de desarrolladores y diseñadores del proyecto Sakai, documentación de la arquitectura del framework tecnológico que se implementa.

A medida que el avance en la investigación y desarrollo de entornos colaborativos brindan mejoras e innovaciones en herramientas (videoconferencias, porfolios, wikis, workshops, etc.) y sus respectivos servicios, crece la cantidad de detalles arquitectónicos que deben ser documentados para facilitar y mejorar la comprensión del sistema. Por otro lado, quienes diseñan el sistema no necesariamente serán los que lo implementen y sus decisiones deberán estar disponibles para que los diversos interesados del sistema puedan revisarlas y comprenderlas. En consecuencia, documentar o describir de manera apropiada el diseño es tan importante como el diseño mismo. En [21] David Parnas introduce esta noción del diseño a través de la documentación ("*design through documentation*")

Principio de Diseño(de Documentación). *Diseñar es documentar; un diseño sin documentación carece de utilidad práctica.*

La documentación de diseño se compone de documentos denominados vistas y documentos que relacionan una o más vistas. Cada vista describe el diseño del sistema desde un punto de vista particular. Cuantas más vistas se documenten, más completa estará la documentación.

Las vistas se clasifican en tres grandes categorías:

- **Vistas de módulos**. Los elementos de estas vistas son módulos o unidades de implementación. Los módulos representan una forma basada en el código de considerar al sistema. Cada módulo tiene asignada y es responsable de llevar adelante una función. Estas vistas hacen referencias a Vistas de módulos, Especificación de Interfaces, Estructura de Módulos, Guía de Módulos, Estructura de Herencia y Estructura de Uso.
- **Vistas dinámicas**. En estas vistas los elementos son componentes presentes en tiempo de ejecución y los conectores que permiten que los componentes interactúen entre sí. A estas vistas se las llama de componentes y conectores, y hacen referencia a Estructura de Procesos, Estructura de Objetos, Diagrama de Interacciones.
- **Vistas con referencias externas**. Estas vistas muestran la relación entre las partes en que fue descompuesto el sistema y elementos externos (tales como archivos, procesadores, personas, etc.). A estas vistas se las llama allocation views.

No siempre es necesario documentar todas las vistas. Se sugiere documentar al menos una vista de cada categoría y siempre documentar la especificación de interfaces [14; 22]

Los detalles arquitectónicos mencionados anteriormente hacen referencia a estas vistas. En esta tesis se documentara los estilos Arquitectónicos utilizados sobre Sakai y usaremos la propuesta de incorporación de propiedades de adaptación dinámicas, reconfiguración e inteligencia (AdRI) a los servicios bases del framework Sakai mencionadas en la tesis doctoral de Alejandro Sartorio[12], con el objetivo de proponer una nueva Arquitectura Sakai que incluya contratos sensibles al contexto (CSC).

1.3. Solución propuesta y contribución

La propuesta de solución que forma parte de la principal contribución de esta tesis, consiste en documentar los estilos arquitectónicos del framework colaborativo web Sakai con contratos sensibles al contexto [1].

En este sentido, primero, se proporcionan diferentes formas de interpretar la arquitectura Sakai con el propósito de construir una primera descripción del subsistema (identificado con la etiqueta Línea 2) de la figura 1.1, con el objetivo de representar aquellos aspectos relevantes en los SCW (Sistemas Colaborativos Web). La documentación se realizará utilizando conceptos teóricos del libro *Documenting Software Architecture*[14]

mediante el lenguaje de documentación 2MIL utilizado en la cátedra de Ingeniería de Software de la Licenciatura en Ciencias de la Computación (FCEI-UNR) [15].

Luego, se realiza la documentación del diseño para inyección de contratos sensibles al contexto en FWCsc desde una visión arquitectónica, donde se propone un nuevo diseño de arquitectura Sakai que respeta mejor las nociones de estilos arquitectónicos para el agregado de CSC.

Por último, se describen un conjunto de variantes de diseño de sistemas estratificados para la representación arquitectónica de las propiedades de coordinación de contratos sensibles al contexto en el marco de los FWCsc. Brindando de esta manera una propuesta de diseño para la composición dinámica de servicios Sakai y servicios externos a nuestro framework. De esta manera se pretende establecer con el propósito de unificar servicios provenientes de diferentes nodos de una red.

1.3.1. Primera noción de los elementos intervinientes en la solución

En esta sección se presentan los elementos influyentes en los componentes de diseño de la arquitectura que conforman la solución propuesta. El diagrama comienza con la interpretación del primer componente mas abstracto denominado Contrato.

Un contrato que siga las ideas de Meyer contiene toda la información sobre los servicios que utilizarán los clientes. Para incorporar sensibilidad al contexto los contratos deberán tener referencias sobre algún tipo de información de contexto que posibilite su utilización. En el diagrama de relaciones entre entidades analizado en la figura 1.3.1 se presenta una primera descripción de los elementos que componen el concepto de contrato sensible al contexto.

La configuración de este diagrama determina los niveles de diseño e implementación que abarcan los diferentes modelos de integración de subsistemas.

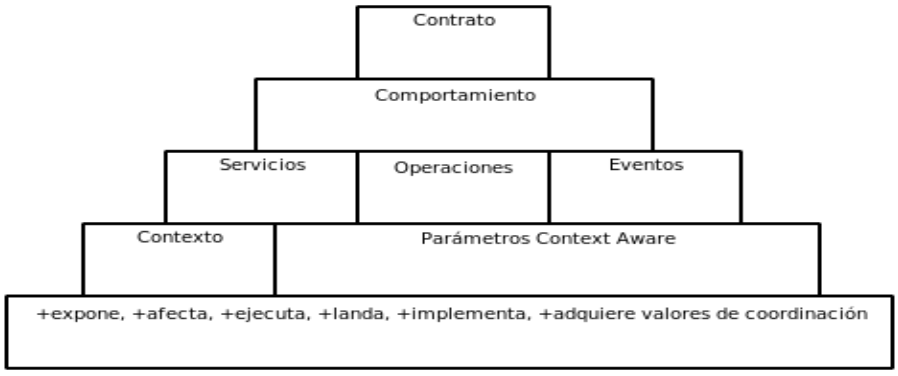


Figura 1.2: PiramideDHD: Primera interpretación de contratos

En la figura 1.3.1 se muestran los elementos para la instrumentación de la noción de contrato. Cada uno de los bloques apilados (desde el pilar 1 al pilar 5) en forma de

piramidal pueden representar componentes abstractas o concretas.

En el bloque base se encuentran las relaciones que intervienen entre las componentes superiores (pilar 1). Luego aparecen los bloques de contexto y parámetros context-aware (pilar 2), que permitirán configurar relaciones que posibilitan resolver requerimientos de adaptación. Seguidamente se encuentran los bloques que identifican componentes concretas que integran las herramientas colaborativas (pilar 3) que están sujetas a modificaciones para establecer las relaciones del bloque base para que se incorporen los elementos intermedios (pilar 2). Los últimos dos pilares tienen que ver con el comportamiento (pilar 4) que implementan los pilares inferiores, el contrato (pilar 5) representa la pieza de software que permitirá el control externo.

A continuación se presenta la descripción individual de las componentes más importantes:

- **Servicios**: En esta componente se representan los elementos necesarios para la identificación de un servicio y clasificación de los servicios que pueden formar parte de las acciones de los contratos. Por ejemplo, nombre del servicio, identificadores, alcance, propósito, etc. Para más detalles consultar[3]. El comportamiento funcional de cada servicio se expone a través de la componente Comportamiento.
- **Comportamiento**: El comportamiento de un servicio se logra a partir de combinar operaciones y eventos que son representados por las componentes Operaciones y Eventos.
- **Parámetros Context-Aware**: Se denominan parámetros context-aware a la representación de la información de contexto que forma parte de los parámetros de entrada de las funciones y métodos exportados por los servicios, estableciendo de esta manera una relación entre el componente Servicios y el componente Parámetros contex-aware. La influencia de estos parámetros en el comportamiento funcional de los servicios es representada a través de la relación entre los componentes Parámetros context-aware y Comportamiento.
- **Contexto**: Este componente representa el contexto o información de contexto definido anteriormente en esta sección. Para el modelo planteado en esta tesis, este tipo de información es utilizada de dos maneras diferentes: 1. para la asignación de los valores que toman los Parámetros context-aware; 2. esta información puede ser utilizada para definir los invariantes que se representan en los contratos.

Otro de los elementos importantes que se tendrán en cuenta en la propuesta de solución es el tipo de representación que se determina para el entorno. Para este propósito se observará que ligado al entorno se provea una infraestructura que permita distribuir el contexto producido por las fuentes. Esta infraestructura fue denominada capa de contexto, y sirve de intermediaria entre las fuentes contextuales y los elementos que permitirán la inyección de propiedades de sensibilidad al contexto.

1.4. Limitaciones

Las principales limitaciones que se evidencian en el aporte de esta tesis se relacionan con la formalización de varias de las especificaciones propuestas para la inclusión de contratos sensibles al contexto a la arquitectura Sakai.

La documentación no es completa, solo describe aquellos puntos considerados necesarios para la descripción de la arquitectura. La propuesta arquitectónica de integración de CSC y CA a Sakai, se centra sólo en algunos aspectos de la documentación que reflejan la integración entre subsistemas.

1.5. Organización del documento

Capítulo 1: En este capítulo se exponen la motivación para desarrollar este tema, la propuesta de solución y las limitaciones existentes del tema.

Capítulo 2: Este capítulo está dedicado a la exposición de los sistemas y estructuras que fueron utilizados como punto de partida hacia la construcción de una propuesta de solución que determina la documentación del Framework Sakai y una nueva arquitectura que sea compuesta por dicho framework y contratos sensibles al contexto.

Capítulo 3: En este capítulo se documenta en detalle los estilos arquitectónicos MVC, Cliente/Servidor y Estratificado que fueron localizados en el framework Sakai. Cada uno de estos Estilos incluirán Diagramas Canónicos, Estructura Física, Módulos 2MIL, Guía de módulos, etc.

Capítulo 4: En este capítulo documentaremos y propondremos una nueva arquitectura Sakai que integre conceptos de CSC. También explicaremos el patrón de diseño utilizado para su posible implementación.

Capítulo 5: En este capítulo propondremos variantes de diseño y soluciones alternativas para la incorporación de dinamismo al framework Sakai.

Capítulo 6: En este capítulo se brindaran las conclusiones y menciones de trabajos futuros.

CAPÍTULO 2

Estado del Arte

2.1. Introducción

Este capítulo está dedicado a la exposición de los sistemas y estructuras que fueron utilizados como punto de partida hacia la construcción de una propuesta de solución que determina la documentación del Framework Sakai y una nueva arquitectura compuesta por dicho framework y contratos sensibles al contexto (CSC). De esta manera se presenta parte del recorrido sobre el estudio del arte partiendo de la teoría de contratos de Meyer, estilos arquitectónicos en el ámbito del software, sistemas sensibles al contexto y el proyecto Sakai.

El conjunto de estos sistemas y estructuras pueden derivar en una nueva Arquitectura Sakai donde el concepto de contratos sensibles al contexto no se considere como un agregado externo, si no como parte de la Arquitectura.

2.2. Estilos Arquitectónicos

Un estilo arquitectónico define una familia de arquitecturas que satisfacen limitaciones. Un estilo permite a uno aplicar conocimiento especializado de diseño a una clase particular de un sistema y sostener esa clase de diseño de sistemas con herramientas de estilo especificas, análisis e implementación.

Algunos estilos son aplicables en cualquier sistemas de software, por ejemplo de-

scomposición, usos, implementación, y asignaciones de trabajos. Otros estilos sólo se producen en sistemas que fueron explícitamente elegidos y diseñado por un arquitecto, por ejemplo estratificado, cliente/servidor, procesos de comunicación (communicating-process).

Elegir un estilo, requiere obligaciones de documentación para registrar especializaciones y limitaciones de un sistema. Denominamos esta pieza de documentación guía del estilo. La obligación de documentar un estilo usualmente puede realizarse citando una descripción del estilo.

Ningún sistema es construido a partir de un único estilo. Por lo contrario, todo sistema puede verse como una combinación de varios estilos. Estas combinaciones pueden realizarse de distintas maneras:

- Diferentes áreas de un sistema pueden exhibir diferentes estilos. Por ejemplo, un sistema puede utilizar tubos y filtros para procesar datos de entrada pero enviar el resultado a una base de datos que es accesada por diferentes elementos. Este sistema es una combinación de tubos y filtros y datos compartido (shared-data). La documentación de este estilo deberá incluir la documentación pertinente de cada estilo que describe una parte del sistema. En este caso uno o mas elementos deben coincidir en ambas vistas y deben tener propiedades de ambos tipos de elementos. Estos elementos (Puentes) proveen la continuidad para comprender una vista y las siguientes. Usualmente estos elementos poseen múltiples interfaces y cada una de ellas provee mecanismos para permitir a elemento trabajar con otros elementos de diferentes vistas a la cual pertenece.
- Un elemento que forma parte de un estilo puede en si, ser compuesto por elementos de otros estilos. Por ejemplo, un servidor en un estilo cliente/servidor puede ser implementado por tubos y filtros sin el conocimiento del cliente. La documentación de este estilo deben incluir una vista de cliente/servidor, así como también una vista de tubos y filtros que documente el servidor.
- Un sistema con repositorios de base de datos puede ser visto como cliente/servidor o como datos compartidos. Si el cliente son procesos independientes, el sistema puede ser visto como procesos de comunicación (communicating-process). *La forma de pararse frente a un sistema, determinara lo que usted vea*.

Estos 3 casos dejan en claro la necesidad de documentar un sistema en partes, utilizando vistas y estilos. Es decir, una vista no debe intentar mostrar todo el sistema.

2.3. Sistemas sensibles al contexto

Podemos entender a un sistema colaborativos sensible al contexto como una aplicación provista de mecanismos que permiten una mejor adaptación de los servicios, a partir del contexto de los usuarios y del entorno. En ambientes colaborativos para educación, los servicios forman parte de las funcionalidades observables desde las perspectivas de

los usuarios (ej.:alumno, docente, etc.), que proveen las herramientas del dispositivo (ej., wiki, foros,mensajería glosario, recursos, etc.). Los elementos que componen la caracterización del contexto deben pertenecer a un dominio bien definido, manteniendo ciertos niveles de concordancia con los mecanismos encargados de manipularlos y la toma de decisiones en base a ellos.

A través de un simple diagrama podemos observar algunas de las características fundamentales que determinan un sistema colaborativos sensibles al contexto, donde se encuentran remarcadas aquellas particularidades (netamente tecnológicas y conceptuales) que se vinculan con la perspectiva de los DHD. En la figura 2.1, describimos en una arquitectura, básica y genérica, tres tipos de niveles, cada uno agrega nuevos rasgos y comportamientos que condicionan los servicios del dispositivo hipermedial hacia un nuevo modelo.

La figura 2.1 está dividida en tres bloques fundamentales, en el primero (Sistema Colaborativos Web Convencional) se describen los principales componentes y relaciones; fundamentalmente un alumno puede interactuar con las herramientas y servicios del sistema y comunicarse con un docente o con sus pares.

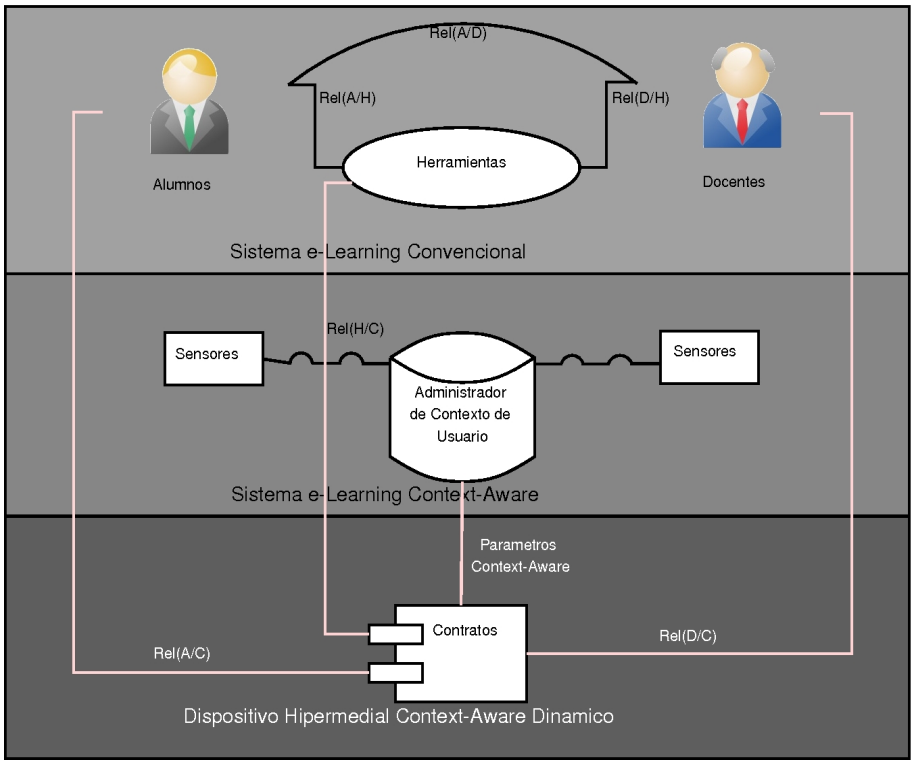


Figura 2.1: Arquitectura Propuesta Evolutiva

El segundo bloque de la figura 2.1 (Sistema Colaborativo Sensible al Contexto) contiene el agregado de dos componentes esenciales, sensores y un administrador de contexto. Los sensores capturan información del usuario y del entorno, son partes fundamentales, harán la recolección de la información; si los sensores juegan un papel secundario, los podemos enmarcar bajo el concepto de artefactos virtuales. El segundo componente hace referencia a la administración del contexto, donde se destacan las siguientes funcionalidades: recolección, abstracción, interpretación, comunicación y al-

macenamiento. Para la implementación de comportamientos context-aware los diseños de software deben completarse con diferentes modelos, para el caso de los denominados sistemas e-learning, las soluciones se focalizan en la tecnología de comunicación e información que utilizan los usuarios. MatchBase (Gross et al., 2006) es un proyecto de referencia en cuanto al uso de estos tipos de diseño y tecnología conformando una herramienta para el desarrollo de comunicaciones context-aware. Fue trasladado como experiencia a proyectos de investigación de sistemas e-learning context-aware (Schmidt, 2005). Pensar en aplicaciones para educación e investigación sensibles al contexto dentro del marco teórico context-awareness conforma un nuevo paradigma, donde las relaciones (rotuladas con "Rel (fuente/destino)" en la figura 2.1) cobran un mayor protagonismo en la concepción de los marcos conceptuales propuestos como soluciones a nuevos requerimientos académicos para el contexto físico-virtual.

Perspectivas como la del DHD, también centran los requerimientos sobre las relaciones y comunicaciones entre los principales componentes del dispositivo, focalizados en las relaciones entre los actores (Recursos Humanos en formación, grupos responsables) y los servicios brindados por las herramientas anteriormente mencionadas, a las que se les integra el potencial de comunicación hipermedial como por ejemplo, vídeo conferencias, edición musical, etc.

Retomando la evolución de los sistemas colaborativos referida a la adaptación de modelos para requerimientos educativos complejos, el ultimo bloque de la figura 2.1 (Dispositivos hipermediales sensibles al ContextoDHD Dinámico) representa un modo de interpretación de una nueva configuración de la arquitectura de un sistema, debido a la interposición de la componente contratos (referenciada con el dibujo característico para componentes de software en UML). Al agregar este nuevo elemento que permite una mejor articulación de las relaciones, resulta una nueva teoría que proporciona conceptualmente un marco innovador para las prácticas de diseño, desarrollo, uso de dispositivos y aplicaciones en dicho campo.

El contrato debe ser visto como una alternativa de abstracción de las relaciones mencionadas anteriormente, determina un nuevo tipo de relación que mantiene las características de los anteriores bloques de la figura y además, reduce la complejidad de los modelos de los SSC en la adaptación de los servicios que interfieren en la relación entre usuarios y comunicaciones. Los contratos se nutren con información del contexto por medio de sus parámetros; no intervienen en la recolección, en la abstracción, ni en la distribución del contexto y en principio pueden ser adaptados a cualquier modelo de sensibilidad al contexto similar, bajo la perspectiva propuesta por Dey. et. al (2001).

Como observamos en la figura 2.1, el bloque intermedio se configura como articulador entre los sistemas colaborativos tradicionales y las nuevas posibilidades tecnológicas que permiten la concreción más efectiva de Dispositivos Hipermediales Dinámicos agregando la componente Contrato.

Esta propuesta evolutiva y la forma de representación del contexto mantiene los principios que definen la teoría de los SCC.

El contexto puede definirse de manera general como:

Cualquier información que puede usarse para caracterizar la situación de una entidad. Donde una entidad es una persona, lugar u objeto que es considerado relevante para la interacción entre un usuario y una aplicación, incluyendo al usuario mismo y la aplicación.

(Dey y Abowd, 2000)

Las aplicaciones sensibles al contexto se definen como:

Aquellas que usan el contexto para proveer información y/o servicios relevantes al usuario, donde la relevancia depende de la tarea del usuario.

(Dey y Abowd, 2000)

Según un análisis realizado por Brown et al., (2000), entre las aplicaciones sensibles al contexto, la recuperación de información juega un papel central, y tales aplicaciones parecen confirmarse como la prospectiva que requiere el computo consiente de contexto.

Por lo tanto, en las aportaciones de los autores de los trabajos que presentaremos a continuación, tomaremos el problema planteado en el marco de la recuperación de información consistente de contexto.

2.4. Proyecto Sakai

El proyecto Sakai es una comunidad basada en el desarrollo open source, que se esfuerza por diseñar, construir y desplegar un nuevo Entorno de Colaboración y Aprendizaje (CLE) para la educación superior. El proyecto comenzó en enero de 2004.

El primer objetivo de dicho proyecto es proporcionar el entorno de aplicación de Sakai y sus herramientas y componentes CMS (Content Management System) asociados, diseñados para funcionar juntos. Estos componentes sirven para la gestión de cursos y, como aumento del modelo CMS original, soportan colaboración en la investigación. El software está siendo diseñado para ser competitivo con el mejor CMS disponible.

Las herramientas están siendo creadas por diseñadores, arquitectos de software y desarrolladores en diferentes instituciones, usando una variación experimental de un modelo de desarrollo de código libre llamado "modelo de código comunitario".

Para proveer un sistema de soporte para instituciones que quiera estar involucrado en el proyecto Sakai, ya sea adoptando las herramientas de Sakai o por medio del desarrollo de herramientas para la portabilidad interinstitucional, el proyecto Sakai ha creado además el Programa de Socios Educativos de Sakai (SEPP) y el Programa de Afiliados Comerciales de Sakai.

Este proyecto tiene su origen en la Universidad de Michigan y en la Universidad de Indiana. Ambas universidades comenzaron independientemente sus esfuerzos en el ámbito del código libre para duplicar y aumentar la funcionalidad de sus CMS existentes. Poco tiempo después, el MIT y la Universidad de Stanford se unieron y, junto a la Iniciativa de conocimiento abierto (OKI) al consorcio uPortal (portal institucional abierto para educación superior) y a una generosa ayuda de la Fundación Mellon, formaron el proyecto Sakai.

El proyecto sigue, como ya expresamos, el modelo de código comunitario, que es una extensión del ya exitoso y, económicamente viable, movimiento de código libre forjado por proyectos como Apache, Linux y Mozilla. Dicho modelo se basa en el objetivo de dirigir las necesidades comunes y únicas de múltiples instituciones, sosteniéndose en mayor medida sobre roles definidos, responsabilidades y compromisos fundados por miembros de la comunidad, que sobre ciertos modelos de desarrollo de código libre.

En el futuro, Sakai proyecta continuar con sus esfuerzos por construir un entorno de trabajo interutilizable, desarrollando varias nuevas herramientas y extensiones. Las características funcionales de Sakai, permiten que cada participante posea, además de las prestaciones educativas similares a las brindadas por Moodle, un espacio para la investigación y desarrollo de proyectos.

En una forma muy diferente, los usuarios eligen entre las diversas herramientas de Sakai para crear su propio sitio de trabajo que sea apropiado para cursos, proyectos y colaboración en investigación.

Desde el punto de vista tecnológico, la principal variable que se pondera se concretiza en el desarrollo de Sakai implementado en Java2, lo que brinda una compatibilidad casi universal.

Sobre la problemática de comunicación hipermedial sincrónica y asincrónica, Sakai posee una herramienta para conferencia, (http://agora.lancs.ac.uk/), que se constituye en un desarrollo open source con excelentes prestaciones para desarrollar metodologías de taller afines a la perspectiva constructivista. Si bien es un proyecto que cuenta con pocos años de desarrollo y de muy reciente inserción en Latinoamérica, capitaliza desde su origen las aportaciones de los principales investigación referentes, y con desarrollo participación de altísimo de comunidades prestigio de académico universitario y aportes económicos significativos que dan factibilidad y proyección al desarrollo.

Sakai está en vías de ser traducido totalmente al español. La política de desarrollo que llevará adelante en los próximos años y el demo de la próxima versión que incorpora todo lo referido a las redes sociales y al paradigma de desarrollo enfocado en servicios, se encuentran publicados en http://sakaiproject.org/future-directions.

En dichos avances también se desarrolla la interoperabilidad con DSpace, este software de código abierto permite gestionar repositorios de archivos digitales en variados formatos, facilitando su depósito, organización, asignación de metadatos y difusión en buscadores (google, google académico, yahoo, etc.) o agregadores (OAIster, ROAR, etc).

CAPÍTULO 3

Documentación

3.1. Arquitectura Sakai

Antes de comentar la Arquitectura de Sakai y su respectivo framework, introduzcamos una definición de Arquitectura de Software.

La arquitectura de software, tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad, y disponibilidad.

Kruchten, Philippe.

Sakai fue desarrollado teniendo en cuenta que cada una de estas propiedades mencionadas por Kruchten Philippe sean cumplidas de la mejor forma posible, gracias a la utilización de patrones de diseño y estilos arquitectónicos se asegura un código de alto nivel que facilita la escalabilidad y mantenimiento del software, proveyendo portabilidad a sus herramientas.

El framework Java Sakai es una implementación particular de la Arquitectura Abstracta Sakai, que hace foco en proveer soporte a Herramientas Sakai TPP (Tool Portability Profile) escritas en java y operando en un Web Browser. El objetivo principal de Sakai TPP y el framework Java Sakai es proveer un ambiente donde las herramientas y servicios para soportar dichas herramientas, puedan ser depositadas como unidades de

expansión o bloques de construcción, para permitir que una organización ensamblé distintos componentes con el fin de resolver un problema en particular.

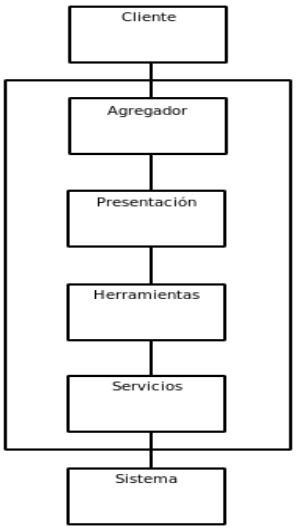


Figura 3.1: Arquitectura Abstracta de Sakai

El framework Java Sakai puede ser pensado como un contenedor de herramientas Sakai TPP y una asociación de servicios. Además de proveer soporte para Herramientas Sakai TPP, el framework soporta integración de portales, integración de Herramientas no TPP, y otros.

El objetivo de las herramientas Sakai TPP es definir interacciones entre herramientas Sakai TPP y el framework Sakai. Una herramienta TPP es una unidad funcional bien formada (similar a un plug-in), que puede ser agregada al framework junto con otras herramientas para proporcionar las funcionalidad general de las aplicaciones. Las herramientas Sakai TPP son por lo general mas restrictivas que aplicaciones java, particularmente porque los aspectos de presentación están acotados en las herramientas TPP debido a que utiliza la presentación provista por el framework.

Un elemento clave del contrato de Sakai TPP es que especifica completamente todas las interacciones que la herramienta hará, incluyendo la interfaz de la capa de presentación, servicios de aplicación y servicios del framework. Sakai TPP forma una cascara alrededor de las herramientas así pueden realmente ser agregadas en cualquier ambiente que cumpla con Sakai TPP.

Hay diferentes enfoques para la integración de funcionalidad a las aplicaciones en el framework Java Sakai.

- Las herramientas de Sakai TPP y servicios son una aproximación a la construcción de unidades de extensión Sakai, que están bien coordinadas con otras herramientas de Sakai. El entorno de Sakai TPP proporciona un conjunto específico de widgets JSF para asegurar la coherencia entre estas herramientas.
- Para las herramientas existentes escritas en Java, o herramientas que deben operar tanto dentro como fuera de Sakai, hay una forma más simple de integración

Servlet donde una aplicación puede acceder a la API de Sakai sin convertirla completamente a una herramienta Sakai. Todavía es posible, aunque más difícil, hacer coincidir la interfaz de usuario Sakai y garantizar la consistencia con las otras herramientas.

- El framework provee un Wrapper para permitir la introducción de herramientas CHEF al ambiente sin necesidad de realizar ninguna modificación. El ambiente Sakai provee servicios para estas herramientas como un remplazo completo del portal JetSpeed que era utilizado en CHEF

Existen casos donde Sakai TPP no es apropiado para integrar funcionalidad a Sakai incluyendo:

- Una aplicación necesita operar en conjunto con Sakai e independiente de Sakai.
- Una aplicación usando tecnología de presentación distinta a JSF, o utilizan un JSF que no es compatible.

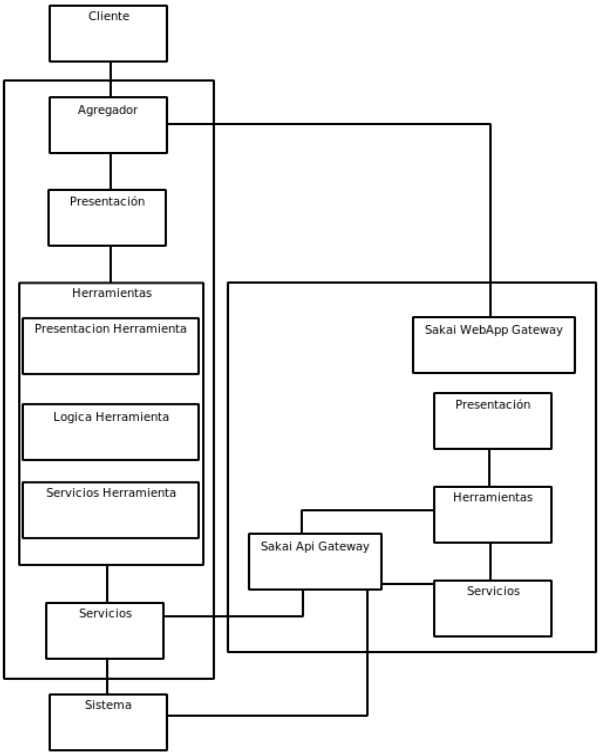


Figura 3.2: Integración de Aplicaciones externas a Sakai

Sakai provee métodos para integrar estas aplicaciones sin la necesidad de reescribir los aspectos de presentación de la herramienta. Herramientas que son integradas de esta forma tiene acceso ilimitado al framework Sakai y a las API's de las aplicaciones. Con una correcta estructura de código, es bastante natural mantener herramientas Sakai y herramientas stand-alone.

La funcionalidad base del ambiente colaborativo Sakai 1.0 fue provista por un numero de herramientas Legacy que evolucionaron de las herramientas CHEF. Estas herramientas

fueron incorporadas a Sakai generando una capa que provee implementaciones básicas de las API's de Velocity y Jetspeed (Tecnologías utilizadas en las herramientas CHEF), pero interactúan con el framework Sakai en vez de con JetSpeed. Las versiones posteriores a Sakai 1.0 extendieron su funcionalidad, manteniendo también las herramientas Legacy y su API's correspondientes.

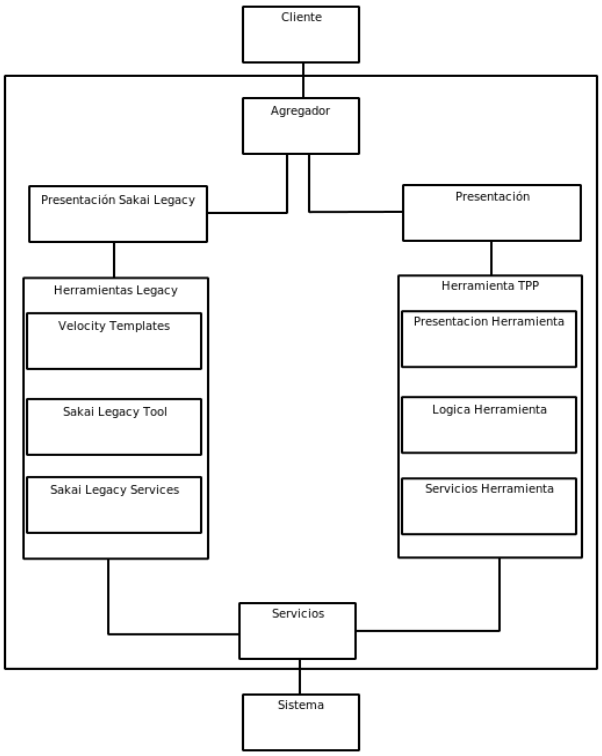


Figura 3.3: Integración de Herramientas Legacy a Sakai.

Se desarrollaron e incluyeron conjuntos de servicios, que en colaboración del framework *"Legacy"* simplificaban el mantenimiento de estas herramientas. Estos dos framework operan juntos en el mismo ambiente. El agregador puede mostrar herramientas Sakai Legacy o herramientas Sakai TPP al mismo tiempo.

El objetivo de este capitulo es mostrar aquellos estilos Arquitectónicos utilizados sobre framework Sakai en orden de proveer escalabilidad y portabilidad de Herramientas. Dentro de la arquitectura Sakai podemos distinguir con facilidad 3 estilos Arquitectónicos, **Cliente Servidor**, **Estratificado** y **Model-View-Controller**. Cada uno de estos estilos se encuentran representados en la figura 3.4.

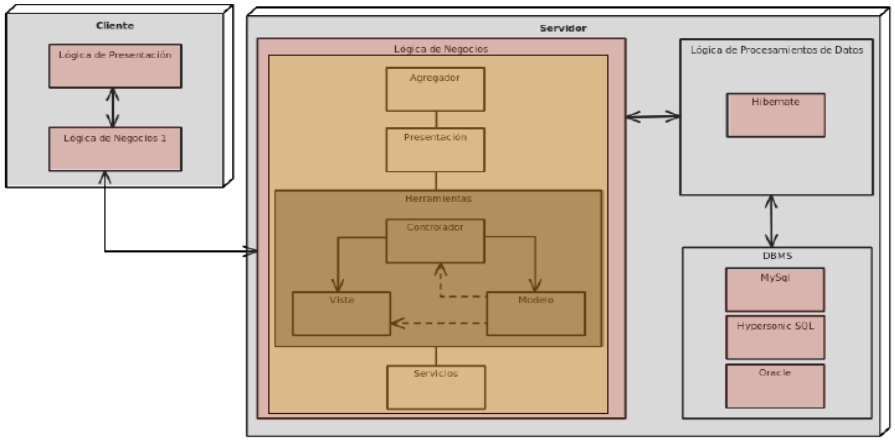


Figura 3.4: Arquitectura general del framework Sakai.

A continuación documentaremos el estilo *Cliente Servido* teniendo en cuenta los métodos de documentación explicados y desarrollados en "Catalogo incompleto de Estilos Arquitectónicos" [35] y "Documenting Software Architectures" [14].

3.1.1. Estilo Cliente Servidor en Sakai

En esta sección se brindaran conceptos básicos del estilo Cliente/Servidor, y de esta forma facilitar el entendimiento de la documentación que sera desarrollada a continuación.

Propósito:

Descomponer el procesamiento y almacenamiento de los datos procesados por grandes sistemas cooperativos, de forma tal que se verifiquen las siguientes cualidades:

- **Integración** de datos y aplicaciones. Unidades corporativas diferentes que desarrollaron sus propios sistemas de procesamiento de datos deben unificar esos sistemas; empresas diferentes con diferentes sistemas se fusionan y por lo tanto lo mismo debe ocurrir con sus sistemas
- **Modificabilidad** de aplicaciones, de representación de los datos, de ubicación física de los componentes. Las reglas de negocio cambian constantemente lo que implica cambios en las aplicaciones y en la representación de los datos; la organización crece y su ubicación física se expande por lo que es necesario que las aplicaciones y los datos migren.
- **Escalabilidad** para permitir que el sistema acompañe el crecimiento de la organización de forma transparente para las unidades que ya están en producción.
- **Aumentar el desempeño**, así de esta manera más usuarios puedan utilizar el sistema. Todo esto en un contexto de grandes cantidades de datos y transacciones. Otras cualidades que se buscan son: tolerancia a fallas, continuidad de las operaciones, alta redundancia y seguridad.

Componentes:

Existen dos tipos de componentes: clientes y servidores. Los clientes solicitan servicios a otros componentes; los servidores proveen servicios, que van desde subrutinas a bases de datos completas, a otros componentes. Es común que un servidor sea cliente de otro servidor.

Clientes y servidores suelen agruparse en capas, también llamadas capas lógicas, cada una de las cuales, muy posiblemente, ejecuta en una plataforma diferente. Cada capa ofrece o solicita un conjunto de servicios y datos (que es la unión de los servicios y datos que ofrecen los servidores dentro de esa capa o los servicios y datos que solicitan los clientes en esa capa). Normalmente una capa es un gran sistema de software (que incluye aplicaciones y datos) por lo que debe ser descompuesto. Cada componente de una capa puede ser un sistema, sub-sistema, un TAD o un módulo. Usualmente se utilizan tres capas.

Las capas no deben confundirse con los estratos de los Sistemas Estratificados[14]. Las capas no suelen diseñarse pensando en máquinas abstractas sino que el criterio se basa en escalabilidad, desempeño, tolerancia a fallas, uso inteligente del ancho de banda, etc. Los SE no suelen ser distribuidos en tanto que esa es la principal característica de los sistemas basados en Cliente/Servidor de Tres Capas en los cuales, como fue mencionado en el párrafo anterior, cada capa ejecuta en una plataforma diferente.

En un sistema CS3C los servidores pueden cumplir funciones muy diversas. Podemos clasificarlos en dos categorías: de negocio y de infraestructura, también llamados de soporte. Dentro de la primera categoría se encuentran las aplicaciones que implementan un requisito funcional específico del negocio; en la segunda categoría están los servidores que cumplen funciones más generales que usualmente abarcan más de un dominio de aplicación. Por ejemplo, en un sistema de gestión corporativa un servidor de negocio provee servicios para procesar documentos contables en tanto que un servidor de infraestructura puede ser un RDBMS [a] o un servidor Web.

Conectores:

- Protocolos cliente/servidor. Por ejemplo: FTP, HTTP, SQL remoto, RPC, NFS, two-phase commit protocols, Distributed Transaction Processing (DTP) de IBM, Webservices, etc.
- Llamada a procedimiento.
- Llamada a procedimiento remota (RPC, RMI, etc.).
- Tubos
- Memoria compartida (aunque deberá utilizarse sólo en casos muy especiales)

Usualmente la comunicación entre componentes es de a pares y la inicia un cliente;

[a] Relational database management system (RDBMS)

al pedido de un servicio de un cliente le corresponde la respuesta del servidor respectivo. En otras palabras la comunicación entre cliente y servidor es, por lo general, asimétrica.

Normalmente la invocación de servicios es sincrónica: el solicitante queda bloqueado hasta que el servicio solicitado completa su tarea y retorna, posiblemente con un resultado.

Algunos protocolos deben ser capaces de proveer integridad de las transacciones que llevan a cabo cooperativamente clientes y servidores.

Otras propiedades de los conectores pueden ser: estrategia para el manejo de errores, cómo se inicia y termina una interacción entre un cliente y un servidor, existencia de sesiones, estrategia para la localización de los servidores, etc.

Documentación

Estructura Física:

Existen varias alternativas para la distribución de los datos, consideramos que la mas utilizada es la que se muestra en la figura 3.5.

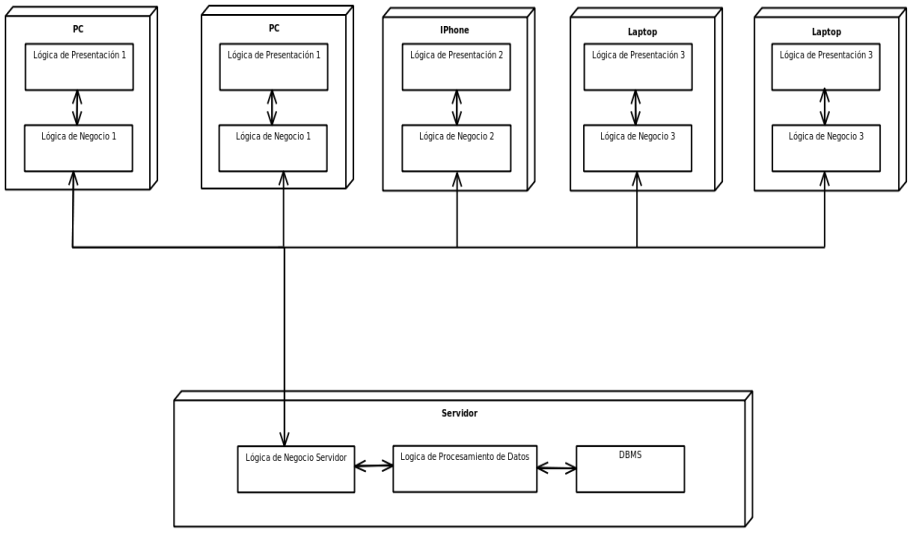


Figura 3.5: Estructura Física posible del framework Sakai.

El framework Sakai estará dispuesto en el servidor, mientras que el cliente normalmente utilizara un web browser para acceder a las herramientas y servicios de Sakai.

El cliente estará compuesto por *lógica de presentación* (LP) y *lógica de negocios* (LN). La *LP* se encargara de visualizar la plataforma en un web browser, mientras que la *LN* realizara chequeos previos al envío de datos, garantizando así la correctitud de los mismos. Parte de la *LN* perteneciente al cliente es implementada tecnológicamente con JavaScripts. En este caso resuelve cada validación de datos y ejecuta el envió de datos

hacia otras capas.

El servidor estará compuesto por *Lógica de Negocios*, *Lógica de procesamiento de datos* (LPD) y la *base de datos* (DBMS) misma, por lo que el framework Sakai se encontrara en su totalidad en el servidor. La *LN* se compone por web Services provistos por Sakai a desarrolladores y a Sistemas externos. Los web services que provee son en su mayoría para la creación y manejo de usuarios, sitios y grupos. La lógica de Herramientas constituye una parte de la *LN*. La *LPD* de los datos se implementa mediante Hibernate. Permitiendo a la aplicación manipular información de la base de datos operando sobre objetos, con todas las características de la POO. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos.

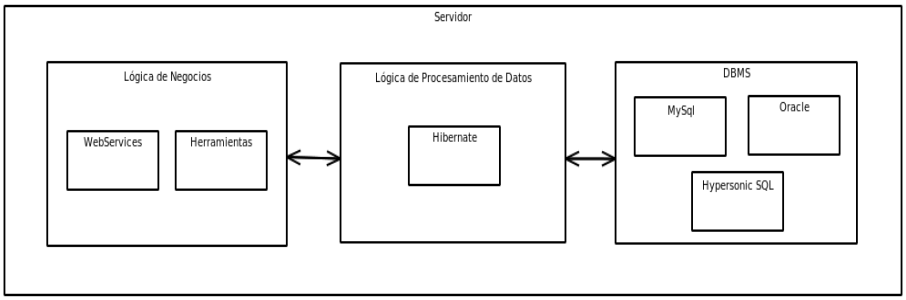


Figura 3.6: Composición lógica del Servidor

Sakai utiliza bases de datos para manejar la información generada por sus herramientas. Las versiones release de Sakai permiten la utilización de una variedad de base de datos, entre las cuales podemos mencionar:

- Hypersonic SQL
- MySQL
- Oracle

Especificación de Interfaces:

En esta sección brindaremos los Módulos 2MIL encargados de la descripción en detalle cada módulo y submódulo [b] pertenecientes al servidor del estilo Cliente/Servidor de Tres Capas(CS3C), estos módulos son descriptos para mostrar las interfaces principales, así como también aspectos arquitectónicos estudiados en esta tesis.

El cliente no sera documentado, esta decisión fue tomada considerando al FWC Sakai como parte del servidor, y el objetivo de este trabajo es realizar su documentación.

[b] Consideramos submódulo a aquel modulo que forma parte de un módulo Mayor. Ejemplo, dado un modulo *A* formado por *B*, *C* y *D*, decimos que *B*, *C* y *D* son submódulos de *A*

También se tuvo en cuenta que la lógica de presentación del cliente es irrelevante, y la lógica de negocio en el cliente utiliza JavaScripts para realizar verificaciones menores.

		MG	DP
Module	**Lógica de Negocios Servidor**		
comprises	Webservices, Herramientas		

		MG	DP
Module	**Webservices**		
comprises	SakaiLogin, SakaiPortalLogin, SakaiScript, SakaiSession, SakaiSigning, SakaiSite		

		MG	DP
Module	**SakaiLogin**		
exportsproc	login()		
	logout()		

		MG	DP
Module	**SakaiPortalLogin**		
exportsproc	login()		
	loginAndCreate()		
	UsageSessionServiceloginDirect()		

		MG	DP
Module	**SakaiSession**		
exportsproc	checkSession		
	getSessionUser		

		MG	DP
Module	**SakaiSigning**		
exportsproc	establishSession		
	testsign		
	verifysign		
	getsession		
	touchsession		

		MG	DP
Module	**SakaiSite**		
exportsproc	establishSession		
	getUserSite		
	getSiteList		
	joinAllSites		
	getSitesDom		
	getToolsDom		

		MG	DP
Module	**SakaiScript**		
exportsproc	checkSession()		
	addNewUser()		
	removeUser()		
	changeUserInfo()		
	changeUserName()		
	changeUserEmail()		
	changeUserType()		
	changeUserPassword()		
	getUserEmail()		
	getUserDisplayName()		
	addNewAuthzGroup()		
	removeAuthzGroup()		
	addNewRoleToAuthzGroup()		
	removeAllRolesFromAuthzGroup()		
	removeRoleFromAuthzGroup()		
	allowFunctionForRole()		
	disallowAllFunctionsForRole()		
	setRoleDescription()		
	addMemberToAuthzGroupWithRole()		
	removeMemberFromAuthzGroup()		
	removeAllMembersFromAuthzGroup()		
	setRoleForAuthzGroupMaintenance()		
	addMemberToSiteWithRole()		
	addNewSite()		
	removeSite()		
	copySite()		
	addNewPageToSite()		
	removePageFromSite()		
	addNewToolToPage()		
	addConfigPropertyToTool()		
	checkForUser()		
	checkForSite()		
	checkForMemberInAuthzGroupWithRole()		
	getSitesUserCanAccess()		

	MG | DP
Module	**Logica de Procesamiento de Datos**
comprises	HibernateAPI

	MG | DP
Module	**HibernateAPI**
comprises	Hibernate, Hibernate.cfg, Hibernate.classic, Hibernate.criterion, Hibernate.metadata, Hibernate.usertype

	MG | DP
Module	**Hibernate**
comprises	Criteria, Filter, Interceptor, Query, ScrollableResults, Session, SessionFactory, SQLQuery, StatelessSession

	MG | DP
Module	**Criteria**
exportsproc	add() addOrder() createAlias() createCriteria() getAlias() list() scroll() scroll() setCacheable() setCacheMode() setCacheRegion() setComment() setFetchMode() setFetchSize() setFirstResult() setFlushMode() setLockMode() setLockMode() setMaxResults() setProjection() setResultTransformer() setTimeout() uniqueResult()

		MG	DP
Module	**Filter**		
exportsproc	getFilterDefinition()		
	getName()		
	setParameter()		
	setParameterList()		
	validate()		

		MG	DP
Module	**Interceptor**		
exportsproc	afterTransactionBegin()		
	afterTransactionCompletion()		
	beforeTransactionCompletion()		
	findDirty()		
	getEntity()		
	getEntityName()		
	instantiate()		
	isTransient()		
	onCollectionRecreate()		
	onCollectionRemove()		
	onCollectionUpdate()		
	onDelete()		
	onFlushDirty()		
	onLoad()		
	onPrepareStatement()		
	onSave()		
	postFlush()		
	preFlush()		

		MG	DP
Module	**SessionFactory**		
exportsproc	close()		
	evict()		
	evictCollection()		
	evictEntity()		
	evictQueries()		
	getAllClassMetadata()		
	getAllCollectionMetadata()		
	getClassMetadata()		
	getCollectionMetadata()		
	getCurrentSession()		
	getDefinedFilterNames()		
	getFilterDefinition()		
	getStatistics()		
	isClosed()		
	openSession()		
	openStatelessSession()		

		MG	DP
Module	**Query**		
exportsproc	executeUpdate()		
	getNamedParameters()		
	getQueryString()		
	getReturnAliases()		
	getReturnTypes()		
	iterate()		
	list()		
	scroll()		
	setBigDecimal()		
	setBigInteger()		
	setBinary()		
	setBoolean()		
	setByte()		
	setCacheable()		
	setCacheMode()		
	setCacheRegion()		
	setCalendar()		
	setCalendarDate()		
	setCharacter()		
	setComment()		
	setDate()		
	setDouble()		
	setEntity()		
	setFetchSize()		
	setFirstResult()		
	setFloat()		
	setFlushMode()		
	setInteger()		
	setLocale()		
	setLockMode()		
	setLong()		
	setMaxResults()		
	setParameter()		
	setParameterList()		
	setParameters()		
	setProperties()		
	setReadOnly()		
	setResultTransformer()		
	setSerializable()		
	setShort()		
	setString()		
	setText()		
	setTime()		
	setTimeout()		
	setTimestamp()		
	uniqueResult()		

		MG	DP
Module	**ScrollableResults**		
exportsproc	afterLast()		
	beforeFirst()		
	close()		
	first()		
	get()		
	getBigDecimal()		
	getBigInteger()		
	getBinary()		
	getBlob()		
	getBoolean()		
	getByte()		
	getCalendar()		
	getCharacter()		
	getClob()		
	getDate()		
	getDouble()		
	getFloat()		
	getInteger()		
	getLocale()		
	getLong()		
	getRowNumber()		
	getShort()		
	getString()		
	getText()		
	getTimeZone()		
	getType()		
	isFirst()		
	isLast()		
	last()		
	next()		
	previous()		
	scroll()		
	setRowNumber()		

		MG	DP
Module	**SQLQuery**		
exportsproc	addEntity()		
	addJoin()		
	addScalar()		
	addSynchronizedEntityClass()		
	addSynchronizedEntityName()		
	addSynchronizedQuerySpace()		
	setResultSetMapping()		

		MG	DP
Module	**Session**		
exportsproc	beginTransaction()		
	cancelQuery()		
	clear()		
	close()		
	connection()		
	contains()		
	createCriteria(()		
	createFilter()		
	createQuery()		
	createSQLQuery()		
	delete()		
	disableFilter()		
	disconnect()		
	enableFilter()		
	evict()		
	flush()		
	get()		
	getCacheMode()		
	getCurrentLockMode()		
	getEnabledFilter()		
	getEntityMode()		
	getEntityName()		
	getFlushMode()		
	getIdentifier()		
	getNamedQuery()		
	getSession()		
	getSessionFactory()		
	getStatistics()		
	getTransaction()		
	isConnected()		
	isDirty()		
	isOpen()		
	load()		
	lock()		
	merge()		
	persist()		
	reconnect()		
	refresh()		
	replicate()		
	save()		
	saveOrUpdate()		
	setCacheMode()		
	setFlushMode()		
	setReadOnly()		
	update()		

		MG	DP
Module	**StatelessSession**		
exportsproc	beginTransaction()		
	close()		
	connection()		
	createCriteria()		
	createQuery()		
	createSQLQuery()		
	delete()		
	get()		
	getNamedQuery()		
	getTransaction()		
	insert()		
	refresh()		
	update()		

		MG	DP
Module	**Transaction**		
exportsproc	begin()		
	commit()		
	isActive()		
	registerSynchronization()		
	rollback()		
	setTimeout()		
	wasCommitted()		
	wasRolledBack()		

Guía de Módulos:

Dado que la Guía de Módulos toma la forma de un documento con capítulos, secciones, subsecciones, etc. Es muy difícil documentarla dentro de un documento que ya tiene esas divisiones. Por lo tanto optamos por documentarla en letra mas pequeña y con títulos, subtítulos, etc. Centrados y sin numeración.

La función de la guía de módulos es describir brevemente la función desempeñada por cada módulo, así como también el secreto que oculta a otros módulos. En otras palabras, cada módulo de la descomposición se caracteriza por su conocimiento de una decisión que oculta a los demás módulos; su interfaz se elige de manera tal de revelar lo menos posible sobre su maquina interna [36].

Module Lógica de Negocios Servidor

Este modulo lógico agrupa los módulos que deben ser modificados si se reemplaza la lógica de Negocios del Servidor. Los submódulos de este módulos proveen la lógica de Negocios necesaria para la implementación de Sakai. Los secretos ocultos en estos módulos son las diversas implementaciones de la lógicas de negocios.

Module Webservices

Este modulo contiene los módulos que deben ser modificados si se reemplaza algún Web Service de Sakai. Los submódulos de este módulos proveen los Web Services básicos. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos Web Services provistos.

Module SakaiLogin

En este modulo se encuentran los Web Services que necesitan Registrarse antes de poder llamar a otros servicios y ocultan su implementación.

Module SakaiPortalLogin

En este modulo ponemos encontrar Web Services que ayudan a las conexiones de Portales como Uportal y ocultan la implementación de sus servicios.

Module SakaiScript

Los Web Services de este modulo proveen formas de manipulación de usuarios, sitios, membresías y permisos de sitios y oculta la implementación de sus servicios.

Module SakaiSession

Los Servicios de este modulo devuelven información de la sesión. Su secreto es la implementación de sus servicios.

Module SakaiSigning

Los Servicios del modulo permiten a aplicaciones externas verificar un usuario. El secreto del modulo es la implementación de sus servicios.

Module SakaiSite

Este modulo provee servicios para manipular sitios. Los métodos con la palabra DOM devuelve una cadena con formato XML. El secreto del modulo es la implementación de los servicios.

Module Lógica de Procesamiento de Datos

Este modulo lógico agrupa los módulos que deben ser modificados si se reemplaza algo de la Lógica de Procesamiento de datos. Los submódulos de este módulo proveen los métodos básicos de procesamiento de datos. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintas lógicas de procesamiento de datos.

Module HibernateAPI

Este modulo contiene los módulos que deben ser modificados si se reemplaza algunas de las APIs centrales de Hibernate, APIs de configuración, compatibilidad con versiones anteriores de Hibernate, APIs para acceso de metamodelos de Hibernate, etc.

Module Hibernate

Este modulo contiene los módulos que deben ser modificados si se reemplaza algunas de las APIs centrales de Hibernate. Los submódulos de este módulo provee los métodos básicos de procesamiento de datos de Hibernate. Los secretos ocultos en estos módulos son las diversas implementaciones de las distintos métodos de Hibernate.

Module Criteria
Esta interfaz publica que provee métodos para recuperar entidades, componiendo Objetos por criterios. Las instancias de los criterios son usualmente obtenidas vía Factory Methods restringidos.

Module Filter
Es una interfaz publica que define tipos de filtros. Los filtros definen las vistas de usuarios en filtros dinámicos, permitiendo que ellos definan los parámetros del filtro.

Module Interceptor
Es una interfaz publica que permite al código de usuario inspeccionar y/o cambiar valores de propiedades. En vez de implementar directamente esta interfaz, usualmente es conveniente extenderla de EmptyInterceptor y sobreescribir los métodos de interés.

Module Query
Es una interfaz publica que provee una representación orientada a objetos de una consulta Hibernate.

Module ScrollableResults
Es una interfaz publica que provee métodos para moverse en un iterador dentro de un resultado por incrementos arbitrarios.

Module Session
Es la interfaz principal entre una aplicación Java y Hibernate. Esta API es la clase central que abstrae la noción de servicios de persistencia.

Module SessionFactory
Es una interfaz publica que crea Sesiones. Usualmente una aplicación tiene un único SessionFactory. El comportamiento de una SessionFactory es controlada por propiedades provistas en tiempo de configuración. estas propiedades son definidas en el ambiente.

Module SQLQuery
Esta interfaz permite a los usuarios declarar tipos y seleccionar los puntos de inyección de listas de todas las entidades devueltas por la consulta. También permite declarar alias a las columnas de cualquier resultado escalar de una consulta.

Module StatelessSession
Es una interfaz publica que provee métodos para realizar operaciones masivas sobre la base de datos.

Module Transaction
Es una interfaz publica que permite a las aplicaciones definir unidades de trabajo, mientras mantiene la abstracción de la implementación de transacción subyacente. Una transacción esta asociada con una sesión y usualmente es instanciada al llamar a Session.beginTransaction().

Protocolos:

En esta sección explicaremos como los Web Browser se comunican con el servidor para obtener paginas web y que protocolos utilizan.
Entre los protocolos utilizados por Sakai, podemos mencionar SOAP, REST y HTTP.

SOAP es un protocolo de mensajes XML que, en el caso de los sitios Sakai, se sitúa por encima de Hyper Text Transfer Protocol (HTTP). Sakai utiliza el framework Apache Axis, en el cual los desarrolladores deben configurar para aceptar llamadas SOAP, vía POST. SOAP envía un mensaje es un formato especifico XML con el tipo de contenido, también conocido como tipo MIME. Un programador no necesita conocer mucho mas que esto, ya que las librerías del cliente se hacen cargo de la mayoría de los detalles de bajo nivel.

REST es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. Para poder entenderlo es necesario comprender HTTP. El conjunto completo de HTTP es OPTIONS, GET, HEAD, POST, PUT, DELETE, and TRACE.

HEAD devuelve del servidor solo las cabeceras de la respuesta sin el contenido, y es útil para clientes que necesitan conocer si el contenido a cambiado desde el ultimo pedido. PUT solicita que el contenido del pedido sea almacenado en una ubicación partícular mencionada en la solicitud. DELETE es para eliminar una entidad.

REST utiliza la URL de la solicitud para rutear el recurso, y GET es utilizado para obtener un recurso, PUT para actualizar, DELETE para borrar, y POST para añadir un nuevo recurso. En general, POST=crea un ítem, PUT=Actualiza un ítem, DELETE=borra un ítem, y GET=devuelve información de un ítem.

A diferencia de SOAP, donde uno apunta directamente a un servicio que el cliente necesita o indirectamente vía una descripción de Web Service, en REST parte de la URL describe el recurso o recursos con los cual uno necesita trabajar. Los servicios de REST son mas intuitivos que los de SOAP y permiten cambiar fácilmente el formato HTML a JSON para alimentar los sitios dinámicos.

HTTP es el protocolo utilizado por Web Browser para obtener paginas web de un servidor. El cliente envía mensajes en formato XML a un servicio, incluyendo la información que el servicio necesita, y luego el servicio devuelve un mensaje con el resultado o el mensaje de error pertinente.

A continuación procederemos a documentar el estilo arquitectónico *Sistema estratificado* como un refinamiento del estilo Cliente/Servidor ya mencionado. El refinamiento es el proceso de revelar información gradualmente de una serie de descripciones, en este caso revelaremos mas información sobre el Servidor del estilo Cliente Servidor aplicado anteriormente.

3.1.2. Sistema Estratificado en Sakai

Es esta sección indagaremos en detalle la estructura que presenta el framework Sakai dispuesto en el Servidor ya mencionado. Aquí se introducen conceptos básicos de este estilo y su proceso de documentación. En la figura 3.7 podremos apreciar una vista general de Sakai y el Sistema Estratificado que procederemos a documentar.

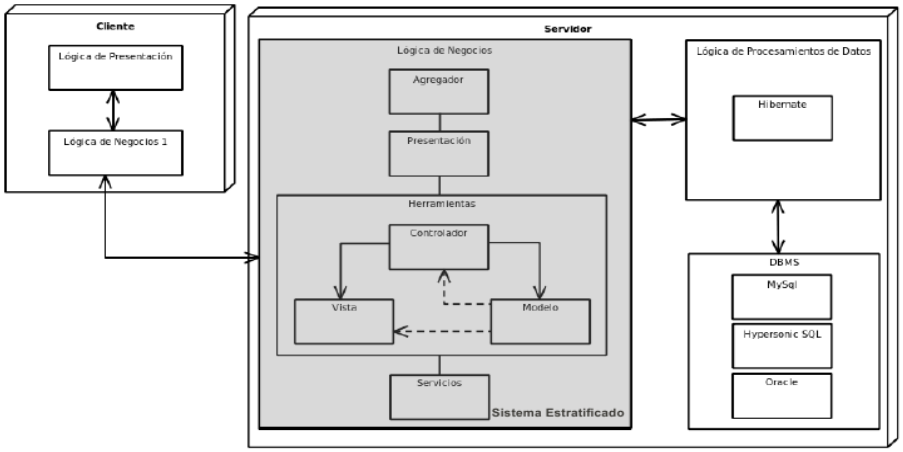


Figura 3.7: Vista General del Sistema Estratificado en Sakai.

Propósito:

Este estilo arquitectónico es muy útil para estructurar aplicaciones que pueden ser descompuestas en grupos de sub-tareas cada una de las cuales está a un nivel de abstracción particular. Los grupos están ordenados jerárquicamente según su nivel de abstracción.

Componentes:

Los componentes de este estilo se denominan estratos. Un estrato ofrece un conjunto de subrutinas en su interfaz y realiza llamadas a la interfaces de otros estratos. Cada estrato debe proveer una funcionalidad más abstracta o con una semántica más rica o más orientada al negocio del sistema, que la provista por aquellos estratos a los cuales invoca. Un estrato puede estar estructurado en módulos que en conjunto proveen los servicios del estrato. Estos módulos cooperan entre sí para proveer la funcionalidad del estrato.

Conectores:

El conector fundamental es llamada a procedimiento. Un estrato invoca los servicios de otro estrato por medio de llamada a procedimiento. Los módulos que componen un estrato pueden conectarse también por llamada a procedimiento. Es común también que se usen eventos para comunicar información desde los estratos inferiores hacia los superiores.

Documentación del estilo

Diagrama Canónico:

En esta sección utilizaremos sistemas estratificados para representar parte de la arquitectura Sakai. En este estilo podemos distinguir 4 estratos:

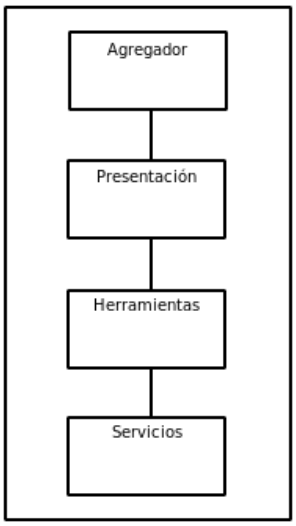


Figura 3.8: Sistema Estratificado Sakai.

- **Agregador:** Es un servidor de aplicaciones destinado a manejar pantallas y transacciones del usuario.
- **Presentación:** La capa de presentación combina datos de las herramientas Sakai y descripción de la interfaz de usuario para crear un fragmento que sera agregado antes de ser enviado al cliente.
- **Herramientas:** Es una aplicación que une lógica de presentación con lógica de negocios. Las herramientas proveen código para responder a solicitudes y eventos de la interfaz de usuario, que pueden (o no) modificar datos.
- **Servicios**: es una colección de clases que manejan datos a través de un conjunto de comportamientos definidos. Estos datos pueden o no, ser persistentes a lo largo de una sesión de usuario. Los servicios intentan ser modulares, reutilizables y portables para el ambiente Sakai y potencialmente para ambientes no Sakai.

Antes de ver en detalle cada estrato para ver su composición y funcionamiento, debemos mencionar que cada uno de estos estratos se encuentran ubicados sobre un servidor web llamado Tomcat. Tomcat es un servidor web que soporta Servlet y JSP, también contiene el compilador Jasper que se encarga de transformar JSP en Servlets. Tomcat nos sirve como piedra basal para el desarrollo de Sakai.

Veamos en mas detalle cada estrato para poder entender mas sobre su composición y funcionamiento.

El estrato **Agregador** esta compuesto por una gran variedad de portales, entre los portales que podemos mencionar se encuentran Charon, Mercury, Uportal, Astro, Pluto, Jetspeed y otros. Nosotros nos concentraremos en **Charon** y en **Mercury** dado que son los mas utilizados. La función de los portales es ensamblar herramientas, botones, etiquetas, etc y producir la interfaz final del usuario. El portal recibe y entrega peticiones luego de haber seteado cosas como el Contexto.

El estrato de **Presentación** debe proveer soporte a diferentes dispositivos tales como PDA, Iphones, Browsers, etc, pero esta capa debe ocultarle a las herramientas sobre que ambiente están siendo ejecutadas y así proveer abstracción y modularidad. También es capaz de producir markup para distintos tipos de agregadores/portales. La capa de Presentación puede ser implementada sobre diferentes tecnologías, entre ellas podemos mencionar JSF, Spring MVC, Struts, Velocity, RSF, etc.

El estrato de **Herramientas** se comentara en la siguiente sección, ya que ahí se documentara en detalle el estrato, explicando que estilo arquitectónico es utilizado para el desarrollo de las herramientas.

El estrato de **Servicios** esta dividido en diferentes capas de servicios:

- **Application Services**, servicios vinculados a un uso particular como Calendarios, Email, Novedades, Tareas, Chat, Recursos, Anuncios, Discusiones.
- **Educational Services**, servicios que soportan la educación como course management y gradebook.
- **Common Services**, servicios básicos utilizados como servicios de usuarios, de grupo, de sitio y de seguridad. Eventualmente se podrá incluir servicios como recursos, repositorios, contenido, etc.
- **framework Services**, núcleo de servicios que requieren persistencia, como entidades.
- **Kernel Services**, núcleo de servicios que no requieren persistencia, como componentes, sesiones, herramientas, etc.

Especificación de Interfaces:

En esta sección se describen en detalle los estratos "Agregador", y "Servicios". El estrato "Herramienta"sera documentado en la siguiente sección, donde se puede apreciar en detalle la aplicación del estilo MVC a las Herramientas Sakai.

En Sakai, el estrato "Presentación"suele ser implementado utilizando el framework JSF. En el framework JSF se incluyen los siguientes componentes:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entradas, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.

- Un conjunto por defecto de componentes para la interfaz de usuario.
- Dos bibliotecas de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP.
- Un modelo de eventos en el servidor.
- Administración de estados.
- Beans administrados.

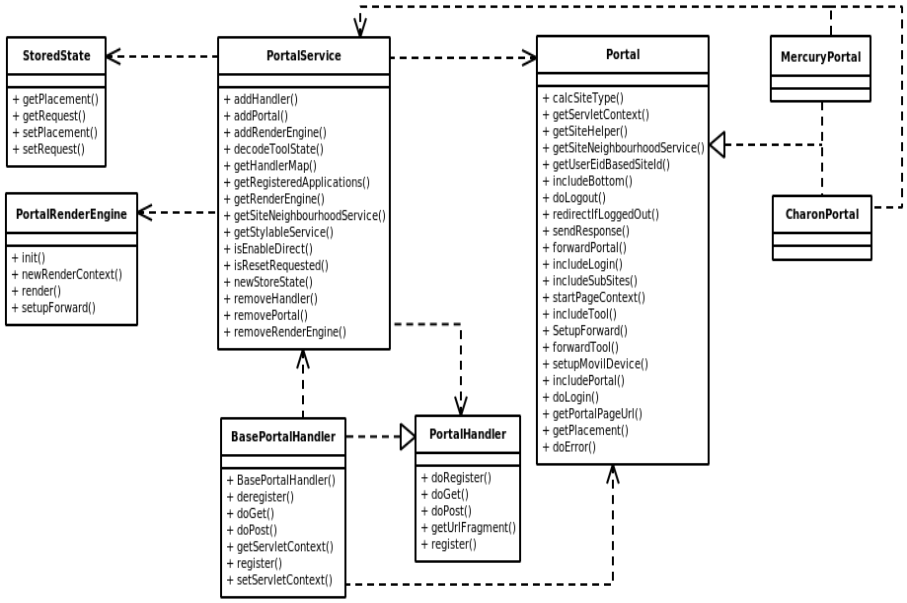


Figura 3.9: Diagrama UML del estrato Agregador

Dada la gran cantidad de servicios que provee Sakai, ApplicationServices, EducationalServices, CommonServices, FrameworkServices y KernelServices, se decidió ampliar solo uno de ellos (KernelServices) y mostrar algunos de los módulos que lo componen. De esta forma evitamos una exageración de módulos 2MIL que no aportaría nada a la tesis.

	MG | DP
Module	**Agregador**
comprises	Portal, PortalHandler, BasePortalHandler, PortalService, PortalRenderEngine

	MG | DP
Module	**Portal**
imports	PageFilter, PortalRenderContext
exportsproc	calcSiteType() getServletContext() getSiteHelper() getSiteNeighbourhoodService() getUserEidBasedSiteId() includeBottom() doLogout() redirectIfLoggedOut() sendResponse() forwardPortal() includeLogin() includeSubSites() startPageContext() includeTool() setupForward() forwardTool() setupMovileDevice() includePortal() doLogin() getPortalPageUrl() getPlacement() doError()

	MG | DP
Module	**CharonPortal inherits from** HttpServlet
imports	HttpServletRequest, HttpServletResponse, Session, String, ToolConfiguration, PortalService
exportsproc	allowTool() calcSiteType() destroy() doErrorReport() doGallery() doGet() doHelp() doLogin() doLogout() doNavLogin() doPage() doPost() doPresence() doSite() doThrowableError() doTitle() doTool() doWorksite() endResponse() forwardPortal() forwardTool() getPlacement() getPortalPageUrl() getScriptPath() getServletInfo() getUserEidBasedSiteId() includeBottom() includeGalleryLogin() includeGalleryNav() includeLogin() includeLogo() includePage() includePageNav() includeSiteNav() includeTabs() includeTitlev() includeTool() includeWorksite() indexOf() init() postLogin() sendPortalRedirect() setupForward() showSession() startResponse()

	MG | DP
Module	**MercuryPortal inherits from** HttpServlet
imports	ActiveTool, HttpServletRequest, HttpServletResponse, ServletConfig, Session, Placement, PortalService
exportsproc	destroy() doDisabled() doError() doGet() doHome() doLogin() doLoginx() doLogout() doPost() doThrowableError() doTool() forwardTool() getServletInfo() init() postHome() postLogin() postTool() printCategories() printConfiguration() printKeywords() showSession()

	MG | DP
Module	**PortalHandler**
exportsproc	doRegister() doGet() doPost() getUrlFragment() register()

	MG | DP
Module	**BasePortalHandler inherits from** PortalHandler
imports	Portal, PortalService
exportsproc	BasePortalHandler() doRegister() doGet() doPost() getServletContext() register() setServletContext()

	MG | DP
Module	**PortalService**
imports	Portal, PortalRenderEngine, StoredState
exportsproc	addHandler() addPortal() addRenderEngine() decodeToolState() getHandlerMap() getRegisteredApplications() getRenderEngine() getSiteNeighbourhoodService() getStylableService() isEnableDirect() isResetRequested() newStoredState() removeHandler() removePortal() removeRenderEngine()

	MG | DP
Module	**StoredState**
imports	
exportsproc	getPlacement() getRequest() setPlacement() setRequest()

	MG | DP
Module	**PortalRenderEngine**
imports	
exportsproc	init() newRenderContext() render() setupForward()

	MG | DP
Module	**Presentación**
comprises	Faces, FacesApplication, FacesComponent, FacesComponentHtml, FacesContext, FacesConvert, FacesEvent, FacesLifecycle, FacesModel, FacesRender, FacesValidator, FacesWebapp

	MG | DP
Module	**Faces**
comprises	FactoryFinder

	MG | DP
Module	**FacesApplication**
comprises	Application, ApplicationFactory, FacesMessage, FacesMessageSeverity, NavigationHandler, StateManagerWrapper, ViewHandler, ViewHandlerWrapper

	MG | DP
Module	**FacesComponent**
comprises	ActionSource, ActionSource2, ContextCallback, EditableValueHolder, NamingContainer, StateHolder, ValueHolder

	MG | DP
Module	**FacesComponentHtml**
comprises	HtmlColumn, HtmlCommandButton, HtmlCommandLink, HtmlDataTable, HtmlForm, HtmlGraphicImage, HtmlInputHidden, HtmlInputSecret, HtmlInputText, HtmlInputTextarea, HtmlMessage, HtmlMessages, HtmlOutputFormat, HtmlOutputLabel, HtmlOutputLink, HtmlOutputText, HtmlPanelGrid, HtmlPanelGroup, HtmlSelectBooleanCheckbox, HtmlSelectManyCheckbox, HtmlSelectManyListbox, HtmlSelectManyMenu, HtmlSelectOneListbox, HtmlSelectOneMenu, HtmlSelectOneRadio

	MG | DP
Module	**FacesContext**
comprises	ExternalContext, FacesContext, FacesContextFactory, ResponseStream, ResponseWriter, ResponseWriterWrapper

	MG | DP
Module	**FacesConvert**
comprises	Converter

	MG | DP
Module	**FacesEvent**
comprises	ActionListener, FacesListener, PhaseListener, ValueChangeListener, ActionEvent, FacesEvent, MethodExpressionActionListener, MethodExpressionValueChangeListener, PhaseEvent, PhaseId, ValueChangeEvent

	MG | DP
Module	**FacesLifecycle**
comprises	Lifecycle, LifecycleFactory

	MG | DP
Module	**FacesModel**
comprises	DataModelListener, ArrayDataModel, DataModel, DataModelEvent, ListDataModel, ResultDataModel, ResultSetDataModel, ScalarDataModel, SelectItem, SelectItemGroup

	MG | DP
Module	**FacesRender**
comprises	Renderer, RenderKit, RenderKitFactory, ResponseStateManager

	MG | DP
Module	**FacesValidator**
comprises	Validator, DoubleRangeValidator, LengthValidator, LongRangeValidator, MethodExpressionValidator

	MG | DP
Module	**FacesWebapp**
comprises	AttributeTag, ConverterELTag, ConverterTag, FacesServlet, FacetTag, UIComponentBodyTag, UIComponentClassicTagBase, UIComponentELTag, UIComponentTag, UIComponentTagBase, ValidatorELTag, ValidatorTag

		MG	DP
Module	**FactoryFinder**		
exportsproc	getFactory() releaseFactories() setFactory()		

		MG	DP
Module	**Lifecycle**		
exportsproc	addPhaseListener() execute() getPhaseListeners() removePhaseListener() render()		

		MG	DP
Module	**LifecycleFactory**		
exportsproc	addLifecycle() getLifecycle() getLifecycleIds()		

		MG	DP
Module	**ActionListener**		
exportsproc	processAction()		

		MG	DP
Module	**PhaseListener**		
exportsproc	afterPhase() beforePhase() getPhaseId()		

		MG	DP
Module	**ValueChangeListener**		
exportsproc	processValueChange()		

		MG	DP
Module	**ActionEvent**		
exportsproc	isAppropriateListener()		
	processListener()		

		MG	DP
Module	**FacesEvent**		
exportsproc	getComponent()		
	getPhaseId()		
	isAppropriateListener()		
	processListener()		
	queue()		
	setPhaseId()		

		MG	DP
Module	**MethodExpressionActionListener**		
exportsproc	isTransient()		
	processAction()		
	restoreState()		
	saveState()		
	setTransient()		

		MG	DP
Module	**MethodExpressionValueChangeListener**		
exportsproc	isTransient()		
	processValueChange()		
	restoreState()		
	saveState()		
	setTransient()		

		MG	DP
Module	**PhaseEvent**		
exportsproc	getFacesContext()		
	getPhaseId()		

		MG	DP
Module	**PhaseId**		
exportsproc	compareTo()		
	getOrdinal()		
	toString()		

		MG	DP
Module	**ValueChangeEvent**		
exportsproc	getNewValue()		
	getOldValue()		
	isAppropriateListener()		
	processListener()		

		MG	DP
Module	**Renderer**		
exportsproc	convertClientId()		
	decode()		
	encodeBegin()		
	encodeChildren()		
	encodeEnd()		
	getConvertedValue()		
	getRendersChildren()		

		MG	DP
Module	**RenderKit**		
exportsproc	addRenderer()		
	createResponseStream()		
	createResponseWriter()		
	getRenderer()		
	getResponseStateManager()		

		MG	DP
Module	**RenderKitFactory**		
exportsproc	addRenderKit()		
	getRenderKitIds()		

	MG | DP
Module	**ResponseStateManager**
exportsproc	getComponentStateToRestore() getState() getTreeStructureToRestore() isPostback() writeState()

	MG | DP
Module	**Services**
comprises	ApplicationServices, EducationalServices, CommonServices, FrameworkServices, KernelServices

	MG | DP
Module	**ApplicationService**
comprises	CalendarService, CourseManagementService, EmailService, NewsService, AssignmentService, ChatService, ResourceService, AnnoucementService, DiscussionService

	MG | DP
Module	**EducationalServices**
comprises	CourseManagementService, GradebookService

	MG | DP
Module	**CommonServices**
comprises	UserService, GroupService, SecurityService, SiteService

	MG | DP
Module	**FrameworkServices**
comprises	EntityService, UserService, SecurityService, ContentService, SiteService, AliasService

	MG | DP
Module	**KernelServices**
comprises	ComponentManager, SessionManager, ToolManager, RequestManager, FunctionManager

	MG | DP
Module	**ToolManagerService**
imports	Site, ToolConfiguration
exportsproc	register() getTool() findTools() getCurrentTool() getCurrentPlacement() setResourceBundle() isVisible()

	MG | DP
Module	**SessionManager**
imports	HttpServletRequest
exportsproc	getSession() makeSessionId() getSessions() startSession() getCurrentSession() getCurrentSessionUserId() getCurrentToolSession() setCurrentSession() setCurrentToolSession() getActiveUserCount()

Guía de Módulos:

Ahora se presenta la guía de módulos donde se describe brevemente la función desempeñada por cada módulo, así como también el secreto que oculta a otros módulos.

Agregador

Este módulo lógico agrupa los módulos que deben ser modificados si se reemplaza algún componente del agregador. Los submódulos de este módulos proveen los clases básicas de los Portales. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintas clases de portales.

Module Portal

Es un modulo abstracto que solo provee una interfaz para utilizar diferentes clases de

Portales.

Module MercuryPortal
Es una clase que provee la implementación de los métodos necesarios del portal Mercury.

Module CharonPortal
Es una clase que provee la implementación de los métodos necesarios del portal Charon.

Module PortalHandler
Es una interfaz que debe ser implementada por aquellas herramientas que quieren agregar un Handler al espacio Url del portal. Una vez inyectada al portal, el portal invocara los métodos register() y deregister() como parte del ciclo de vida.

Module BasePortalHandler
Es una clase abstracta que contiene métodos bases para los PortalHandlers.

Module PortalService
Es una clase que actúa como un foco para todas las actividades básicas del Portal, las implementaciones del servicio deben actuar como un contenedor para permitir que varias aplicaciones web se comuniquen entre si.

Module PortalRenderEngine
Esta interfaz representa la API utilizada por el Portal para comunicarse con la implementación del RenderEngine.

Module Presentación
Este módulo lógico agrupa los módulos que deben ser modificados si se reemplaza algún componente del framework JSF. Los submódulos de este módulos proveen los componentes básicos del framework. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos componentes provistos por el framework.

Faces
Este módulo comprende las APIs de nivel mas alto de JavaServer Faces.

FacesApplication
Este módulo contiene las APIs que son utilizadas para conectar la lógica de negocios de las aplicaciones con JavaServer Faces.

FacesComponent
Este módulo posee APIs fundamentales para los componentes de la interfaz de usuario.

FacesComponentHtml
Este módulo contiene clases especializadas para componentes de la interfaz de usuario basadas en HTML.

FacesContext
Este módulo contiene las clases e interfaces necesarias para definir información de estado

de una solicitud.

FacesConvert
Este módulo contiene las clases e interfaces necesarias para definir una conversión.

FacesEvent
Este módulo posee las interfaces necesarias para describir eventos y escuchador de evento(event listeners), y clases concretas de implementación de eventos.

FacesLifecycle
Este módulo posee clases e interfaces necesarias para definir la administración del ciclo de vida(lifecycle) de las implementaciones para JavaServer Faces.

FacesModel
Este módulo contiene modelos estándar de data beans para JavaServer Faces.

FacesRender
Este módulo esta compuesto por clases e interfaces que definen modelos de interpretación

FacesValidator
Este módulo posee interfaces que definen modelos de validaciones, y las clases de implementación de las validaciones.

FacesWebapp
Este módulo contiene las clases requeridas para la integración de JavaServer Faces en aplicaciones web, incluyendo Servlets estándar, clases bases para etiquetas JSP, y implementaciones concretas de las etiquetas.

FactoryFinder
Este módulo implementa los algoritmos de búsqueda para todos los objetos fabricas (Factory Objects) de las APIs de JavaServer Faces. Dada el nombre de una clase fabrica, su clase de implementación es buscada utilizando estos algoritmos.

Lifecycle
Este módulo maneja el procesamiento del ciclo de vida de una solicitud JavaServer Faces en particular. Es responsable de ejecutar todas las fases que fueron definidas en la especificación de JavaServer Faces, en el orden especificado.

LifecycleFactory
Este módulo es una fabrica que crea (si es necesario) y devuelve instancias de Lifecycle. Las implementaciones de JavaServer Faces deben proveer al menos una implementación de Lifecycle que sea default.

ActionListener
Este módulo provee una interfaz para recibir ActionEvents. Una clase que le interese recibir estos eventos, debe implementar esta interfaz, y luego registrarse con el UIComponent de su interés, llamando a addActionListener().

PhaseListener
Este módulo provee una interfaz que debe implementarse por aquellos objetos que deseen ser notificados al principio y fin de cada fase del ciclo de vida.

ValueChangeListener
Este módulo posee una interfaz que debe implementar aquella clase que este interesada de recibir ValueChangeEvents. Además debe registrarse con el UIComponent de su interés, llamando a addValueChangeListener().

ActionEvent
Este módulo representa la activación de una componente de usuario (así como UICommand).

FacesEvent
Este módulo es la clase base para la interfaz de usuario y eventos de aplicación, que puede ser disparadas por UIComponents.

MethodExpressionActionListener
Este módulo representa una ActionListener que envuelve a MethodExpression. Cuando recibe un ActionEvent, ejecuta un método sobre el objeto identificado por el MethodExpression.

MethodExpressionValueChangeListener
Este módulo representa un ValueChangeListener que envuelve un MethodExpression. Cuando recibe un ValueChangeEvent, ejecuta un método sobre el objeto identificado por el MethodExpression.

PhaseEvent
Este módulo representa el principio y el fin del proceso de una fase en particular del ciclo de vida de una solicitud.

PhaseId
Este módulo representa la numeración que puede ser devuelta al llamar a método getPhaseId() sobre la interfaz FacesEvent.

ValueChangeEvent
Este módulo representa una notificación, que indica la modificación del valor local del componente origen, como resultado de la actividad del usuario.

Renderer
Este módulo convierte la representación interna de un UIComponent en un flujo de salida asociado con el pedido particular.

RenderKit
Este módulo representa una colección de instancias Renderer que, juntas, saben como entregar instancias de JavaServer Faces UIComponent a clientes específicos.

RenderKitFactory
Este módulo es una fabrica de objetos que registra y devuelve instancias de RenderKit.

ResponseStateManager
Este módulo es una clase de ayuda para un StateManager que conoce la tecnología de presentación utilizada para generar la respuesta.

Module Services
Este modulo contiene los módulos que deben ser modificados si se reemplaza algún servicio de Sakai. Los submódulos de este módulos proveen los servicios básicos. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos servicios provistos.

Module ApplicationService
Este modulo contiene los módulos que deben ser modificados si se reemplaza algún servicio de aplicación. Los submódulos de este módulos proveen los servicios básicos de aplicaciones como Calendario, News, Email y otros. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos servicios provistos.

Module EducationalServices
Este modulo contiene los módulos que deben ser modificados si se reemplaza algún servicio de Educación. Los submódulos de este módulos proveen los servicios básicos de Educación como Gradebook y Course Management. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos servicios provistos.

Module CommonServices
Este modulo contiene los módulos que deben ser modificados si se reemplaza algunos de los servicios comunes. Los submódulos de este módulos proveen los servicios comunes de Usuarios, Grupos, Seguridad y otros. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos servicios provistos.

Module FrameworkServices
Este modulo contiene los módulos que deben ser modificados si se reemplaza algunos de los servicios del framework. Los submódulos de este módulos proveen los servicios de Entidades, Contenidos y otros. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos servicios provistos.

Module KernelServices
Este modulo contiene los módulos que deben ser modificados si se reemplaza algunos de los servicios del Kernel. Los submódulos de este módulos proveen los servicios Manejo de Solicitudes, Manejo de Sesiones, Manejo de Funciones y otros. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos servicios provistos.

A continuación procederemos a documentar el estilo arquitectónico *MVC* como un refinamiento del estilo Sistema Estratificado ya mencionado. En este caso el refinamiento se realizara sobre el estrato Herramientas, y de esta forma indagar la estructura, características y funcionamiento del estrato.

3.1.3. Model-View-Controller en Herramientas Sakai

En esta sección brindaremos una introducción a MVC y luego describiremos como es utilizado este estilo en el framework colaborativo web Sakai. Dada el amplio soporte a diferentes tecnologías que provee Sakai, es posible implementar sus herramientas utilizando diferentes tecnologías respetando el estilo arquitectónico. Dentro de las tecnologías de presentación que se encuentran, podemos nombrar:

- **JSF**
- **Spring MVC**
- Struts
- Velocity
- RSF
- Otras

En esta sección daremos la descripción de las 2 primeras, JSF y Spring MVC, ya que actualmente son las mas utilizadas.

En esta sección nos concentraremos en las herramientas Sakai. Pondremos la lupa sobre el estrato "Herramientas" mencionado en el estilo Estratificado ya comentado. En la figura 3.10 se muestra una ubicación general del estilo en el framework. En esta sección también se incluyen conceptos básicos y los propósitos de este estilo.

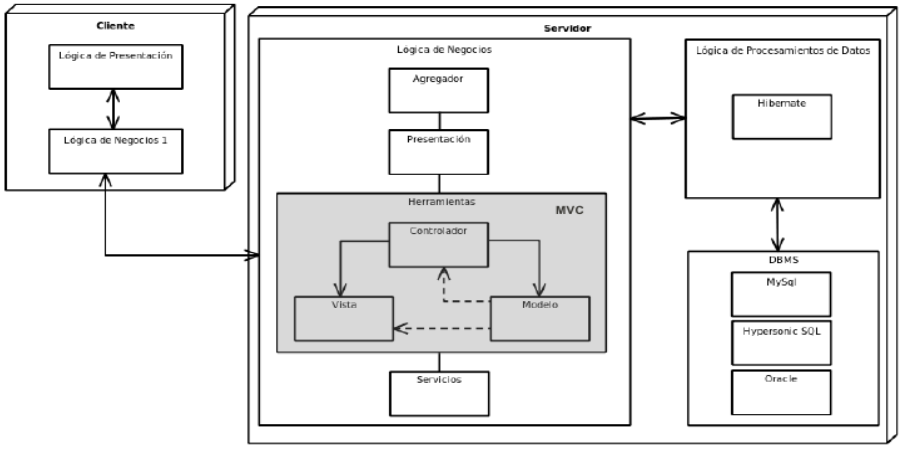


Figura 3.10: Vista General del Estilo MVC en Sakai.

Propósito:

El estilo nos permite separar la lógica del dominio de la aplicación, la lógica de presentación, y la iteración de la aplicación con el usuario en tres clases separadas llamadas respectivamente Modelo, Vista y Controlador. Como consecuencia de esta separación entre la interfaz de usuario y la lógica de negocio permite [37]:

- Aislar los cambios en la interfaz de usuario previniendo de que estos impliquen cambios en el modelo de datos y vice versa.
- Soportar interfaces de usuarios múltiples y sincronizadas del mismo modelo de datos.
- Permitir que el modelo sea construido y validado independientemente de su representación visual.

Componentes:

Todo sistema diseñado bajo este estilo se descompone en tres clases de componentes distintos denominadas Modelo, Vista y Controlador [38].

Modelo: Encapsula la lógica del dominio de la aplicación. Esto es, los datos y la funcionalidad que se corresponde con los requerimientos funcionales de la aplicación. El Modelo presenta una API a través de la cuál los componentes Vista y Controlador acceden al modelo de datos.

Vista: Constituyen el input y el output de la aplicación, esto es, la iteración del usuario con la aplicación. Desde el punto de vista del input, las Vistas presentan al usuario componentes visuales a través de los cuáles el usuario ingresa información y/o emite ordenes al sistema. Desde el punto de vista del output, las Vistas obtienen datos desde el Modelo y representan esa información de una manera particular. Múltiples Vistas pueden representar un mismo elemento de datos de forma diferente.

Controlador:

Gestiona el input del sistema, esto es, las órdenes de entrada desde las Vistas como pueden ser señales del mouse o del teclado. El Controlador recibe el input y decide la acción a tomar para dar respuesta a esa entrada. Esta acción puede ser una invocación a otra Vista o al Modelo para ejecutar la orden de o entrada. Por cada petición que el usuario pueda efectuar desde la Vista, existe o un Controlador encargado de atender dicha petición.

Conectores:

Llamada a procedimiento: Son utilizados por el Controlador para llamar a las Vistas y al Modelo. También se utilizan por las Vistas para invocar al Controlador cuando ocurre un input del usuario y también para invocar al Modelo para obtener la información necesaria para generar un output.

Eventos: Se utilizan por el Modelo para informar a las Vistas y Controladores activos de los cambios en los datos del modelo.

Existen varias implementaciones posibles para el desarrollo del estilo MVC sobre Sakai, nosotros haremos foco sobre aquellos que utilizan JSF y Spring MVC. A continuación describiremos la estructura y comportamiento del estilo utilizando JSF para

luego describirlo haciendo uso de la tecnología Spring. En esta sección documentaremos el estilo, utilizando Módulos 2MIL y siguiendo la metodología puesta en practica en capítulos anteriores.

Documentación del estilo(Utilizando JSF)

Diagrama Canónico:

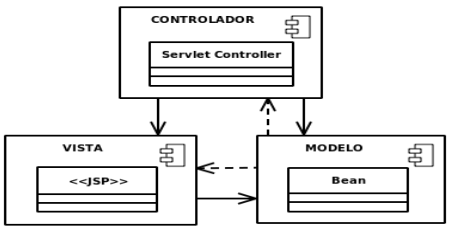


Figura 3.11: Diagrama Canónico para MVC utilizando JSF.

Los siguientes pasos describen el comportamiento usual dinámico del sistema MVC utilizando JSF:

- El Controlador recibe una petición de usuario realizada a través de la vista relacionada y ejecuta el método manejador de la petición. Normalmente este método traduce la petición de usuario a una llamada a un procedimiento del modelo.
- El modelo ejecuta el procedimiento invocado por el controlador que puede resultar en un cambio de datos del mismo.
- En caso de ocurrir un cambio de datos, el modelo notifica dichos cambios a todas las vistas y controladores registrados en el mecanismo de notificación de cambios.
- Cada vista involucrada con los cambios invoca algún método de la API del modelo para obtener los datos modificados y actualiza su contenido con la nueva información.
- Cada controlador afectado por los cambios ejecuta su procedimiento correspondiente para atender dicho evento de cambio.
- El controlador original recupera el control de ejecución en su método manejador de la petición de usuario y una ves finalizada la ejecución del método puede eventualmente invocar a una vista de output encargada de mostrar al usuario el resultado a su petición.

Especificación de Interfaces:

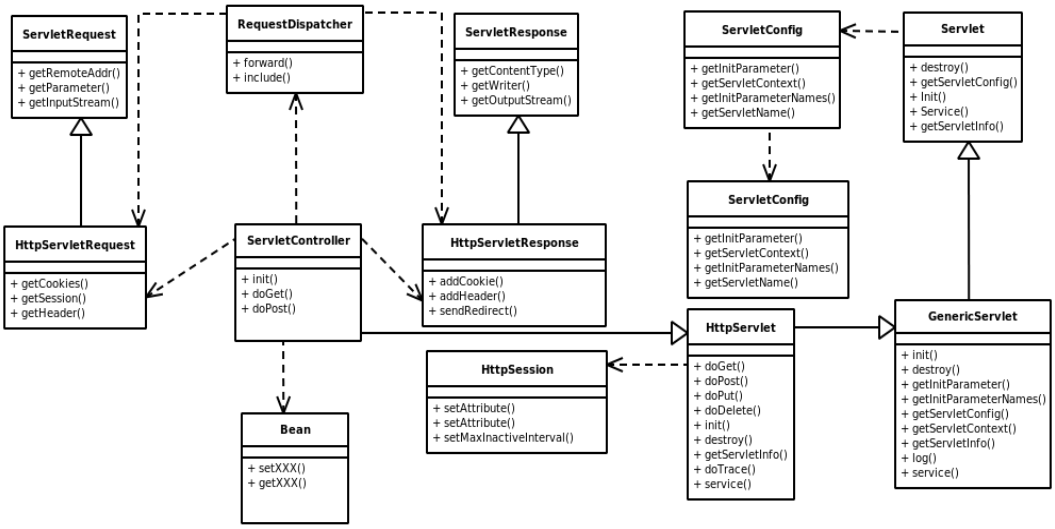


Figura 3.12: Representación UML de las clases MVC utilizando JSF

	MG
Module	**ServletController inherits from** HttpServlet
imports	Servlet, RequestDispatcher, HttpServletRequest, HttpServletResponse
exportsproc	init() dopost(HttpServletRequest req , HttpServletResponse res) doget(HttpServletRequest req , HttpServletResponse res)

	MG | DP
Module	**RequestDispatcher**
imports	HttpServletRequest, HttpServletResponse
exportsproc	forward(HttpServletRequest req , HttpServletResponse res) include(HttpServletRequest req , HttpServletResponse res)

	MG | DP
Module	**Bean**
exportsproc	setXXX() getXXX()

	MG
Module	**HttpServlet inherits from** GenericServlet
imports	HttpServletRequest, HttpServletResponse
exportsproc	doGet(HttpServletRequest req , HttpServletResponse res) doPost(HttpServletRequest req , HttpServletResponse res) doPut() doDelete() destroy() getServletInfo() doTrace() service()
comments	doGet(HttpServletRequest req, HttpServletResponse resp), es llamado para procesar información que haya sido enviado con el método GET. doPost(HttpServletRequest req, HttpServletResponse resp), ídem al anterior pero para el método POST.

	MG
Module	**GenericServlet inherits from** Servlet
imports	String
exportsproc	init() destroy() getInitParameter() getInitParameterNames() getServletConfig() getServletContext() getServletInfo() log() service(String msg)
comments	log(String msg), Escribe en la consola del servidor el mensaje junto con el nombre del servlet.

	MG | DP
Module	**Servlet**
imports	ServletConfig, HttpServletRequest, HttpServletResponse
exportsproc	Init(ServletConfig Config) destroy() getServletConfig() getServletInfo() service()
comments	init(ServletConfig Config), método utilizado para crear una nueva instancia del servlet. GetServletConfig(), Retorna la configuración dada para la inicialización del servlet. Service(), método utilizado cuando se recibe una petición de un cliente y en su implementación normal para HTTP verifica el tipo de solicitud GET, POST, etc. y la redirige a los métodos respectivos. Destroy(), es llamado por el servidor para indicar que el servlet será destruido.

	MG | DP
Module	**ServletConfig**
imports	ServletContext, String
exportsproc	getInitParameter(String name) getServletContext() getInitParameterNames() getServletName()
comments	getInitParameter(String name), retorna el valor del parámetro dado en name. GetServletContext(), Que retorna un objeto ServletContext que guarda la información referente al servidor.

	MG | DP
Module	**ServletContext**
imports	String
exportsproc	getMimeType(String file)
comments	getMimeType(String file), Retorna el tipo MIME (Multipurpose Internet Mail Extensions) definido en el servidor para el archivo dado.

	MG | DP
Module	**ServletRequest**
imports	String
exportsproc	getRemoteAddr() getParameter(String name) getInputStream()
comments	getRemoteAddr(): Retorna la IP del cliente. getParameter(String name): Retorna el valor del parámetro name dado por el cliente. getInputStream(): Sirve para crear un canal de comunicación para obtener datos binarios.

	MG | DP
Module	**ServletResponse**
imports	String
exportsproc	getContentType(String type) getWriter() getOutputStream()
comments	setContentType(String type): Permite definir el tipo de respuesta que se le dará al cliente. getWriter(): Retorna un objeto Writer para poder enviar respuestas de texto. getOutputStream(): Retorna un objeto ServletOutputStream que permite enviar respuestas binarias al cliente.

	MG
Module	**HttpServletRequest inherits from** ServetRequest
imports	String
exportsproc	getHeader(String name) getCookies() getSession()
comments	getHeader(String name): Permite obtener el valor de los Headers de HTTP con que fue llamado el servlet. getCookies(): Retorna un arreglo que contiene todas las cookies que el cliente envía al servlet. getSession(): Retorna la sesión en la cual se encuentra el cliente.

	MG
Module	**HttpServletResponse inherits from** ServetResponse
exportsproc	addCookie(Cookie cookie) setHeader(String name, String value) sendRedirect(String location)
comments	addCookie(Cookie cookie): Define nuevas cookies en el cliente. setHeader(String name, String value): Define un header HTTP a enviar al cliente. sendRedirect(String location): Envía un mensaje al cliente para redireccionar la respuesta a la dirección señalada.

	MG | DP
Module	**HttpSession**
exportsproc	setAttribute(String name, Object value) setAttribute(name) setMaxInactiveInterval(int interval)
comments	setAttribute(String name, Object value), permite compartir un objeto cualquiera entre distintos servlets para el mismo usuario. setAttribute(name) se utiliza para obtener el objeto. setMaxInactiveInterval(int interval): Permite definir un tiempo máximo para el cual la sesión será válida.

Guía de Módulos:

La función de la guía de módulos es describir brevemente la función desempeñada por cada módulo, así como también el secreto que oculta a otros módulos.

Module Servlet
Es una interfaz abstracta que todos los servlets deben implementan directamente o extendiendo una clase que lo implemente como HttpServlet.

Module ServletConfig
Es una interfaz abstracta que contiene los parámetros que entrega el servidor al servlet para ser inicializado que pueden ser dados por el administrador a través de un archivo de configuración.

Module ServletContext
Es una interfaz abstracta que contiene métodos que sirven para comunicar un servlet con el servidor que lo contiene.

Module GenericServlet
Es una clase abstracta que implementa Servlet y/o ServletConfig. Define un servlet genérico independiente del protocolo. Además de implementar alguno de los métodos

de las interfaz.

Module HttpServlet
Es una clase abstracta que hereda de GenericServlet. Es la clase de la cual se debe extender para crear un servlet HTTP.

Module ServletRequest
Es una interfaz abstracta que permite obtener información del cliente que no depende del protocolo.

Module ServletResponse
Es una interfaz abstracta que define un objeto para permitir a un servlet enviar una respuesta al cliente.

Module HttpServetResponse
Es una interfaz abstracta que hereda de ServletResponse y permite enviar al cliente respuestas específicas del protocolo HTTP.

Module Bean
Es una interfaz abstracta que provee los métodos principales Set y Get del modelo que esta siendo representado.

Module RequestDispatcher
Es una interfaz diseñada para envolver los servlets, así de esta forma un contenedor de servlets pueden crear objetos RequestDispatcher para envolver cualquier tipo de recurso. Define un objeto que recibe pedidos de un cliente y lo envía a cualquier recurso (tal como servlet, archivos HTML, o archivos JSP) en el servidor.

Module ServletController
Posee la lógica de negocios necesaria para acceder y reenviar la información obtenida de los Beans.

Module HttpSession
Es una interfaz abstracta que permite identificar al mismo usuario a través de distintos servlets.En general se implementa guardando una cookie en el cliente la cual es recuperada por el servidor para reasignar su sesión.

Documentación del estilo(Utilizando Spring MVC)

Diagrama Canónico:

Los siguientes pasos describen el comportamiento usual dinámico del sistema MVC utilizando Spring:

- El DispatcherServlet recibe una petición de usuario y este debe delegar a otro

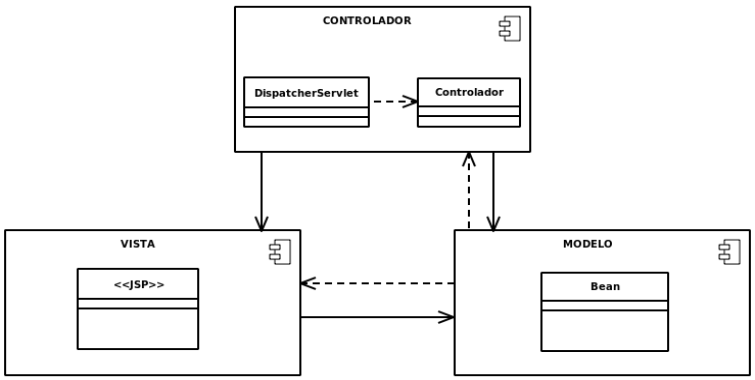


Figura 3.13: Diagrama Canónico para MVC usando Spring.

componente el procesamiento del pedido. Utiliza uno o mas HandlerMapping para determinar que controlador debe recibir el request enviado por el usuario.

- El HandlerMapping devuelve al DispatcherServlet el nombre del controlador que deberá hacerse cargo del pedido del usuario.
- Una vez que el controlador recibe el pedido, crea un objeto que se denomina ModelAndView que tiene el fin de entregar un nombre lógico a la vista que realizara el despliegue del Modelo, entregar un nombre lógico al Modelo e Inyectar el objeto Modelo el cual tiene los datos que serán desplegados en la Vista.
- Luego que el objeto ModelAndView es regresado al DispatcherServlet, este delega la responsabilidad de la mapping del nombre lógico de la vista, con el componente ViewResolver.
- El ViewResolver es el encargado de realizar el mapping entre el nombre lógico de la vista y el componente.
- Luego que la vista realiza el procesamiento, el DispatcherServlet envía el request de retorno al usuario.

Especificación de Interfaces:

En esta sección brindaremos los módulos 2MIL encargados de la descripción en detalle cada módulo y submódulo pertenecientes al estilo MVC, estos módulos son descriptos para mostrar las interfaces principales, así como también aspectos arquitectónicos estudiados en esta tesis.

		MG	DP
Module	**Bean**		
exportsproc	setXXX()		
	getXXX()		

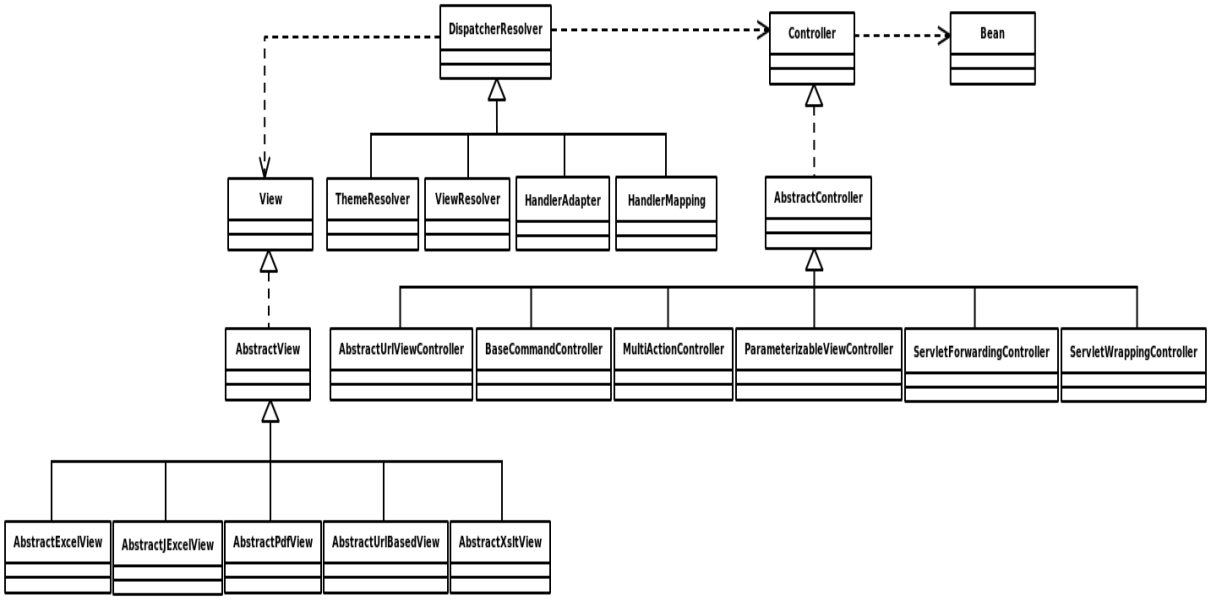


Figura 3.14: Representación UML de las clases MVC utilando Spring MVC

	MG | DP
Module	**ThemeResolver**
imports	String, HttpServletRequest, HttpServetResponse
exportsproc	resolveThemeName() setThemeName()

	MG | DP
Module	**HandlerAdapter**
imports	Object, HttpServletRequest, HttpServletResponse
exportsproc	getLastModified() handle() supports()

	MG | DP
Module	**Controller**
imports	HttpServletResponse, HttpServletRequest, ModelAndView
exportsproc	handleRequest() handleRequestInternal() isSynchronizeOnSession() setSynchronizeOnSession()

	MG | DP
Module	**AbstractController inherits from** WebContentGenerator
imports	HttpServletRequest, HttpServetResponse
exportsproc	handleRequest()
	handleRequestInternal()
	isSynchronizeOnSession()
	setSynchronizeOnSession()

	MG | DP
Module	**MultiActionController inherits from** AbstractController
imports	HttpServletRequest, HttpServletResponse
exportsproc	bind()
	createBinder()
	getCommandName()
	getExceptionHandler()
	getLastModified()
	getMethodNameResolver()
	getValidators()
	getWebBindingInitializer()
	handleNoSuchRequestHandlingMethod()
	handleRequestInternal()
	initBinder()
	initBinder()
	invokeNamedMethod()
	newCommandObject()
	setDelegate()
	setMethodNameResolver()
	setValidators()
	setWebBindingInitializer()

	MG | DP
Module	**ViewResolver**
imports	String, Locale, View
exportsproc	resolveViewName()

	MG | DP
Module	**HandlerMapping**
imports	HttpServletRequest
exportsproc	getHandler()
comments	Devuelve un handler y cualquier interceptors para esta petición.

	MG
Module	**DispatcherServlet inherits from** FrameworkServlet
imports	HttpServlet, ApplicationContext, Object, ModelAndView, HandlerMapping, ViewResolver, MultipartResolver, ApplicationContext
exportsproc	buildLocaleContext() checkMultipart() cleanupMultipart() createDefaultStrategy() doDispatch() doService() getDefaultStrategies() getDefaultStrategy() getDefaultViewName() getHandler() getHandlerAdapter() getLastModified() getMultipartResolver() getThemeSource() initStrategies() noHandlerFound() onRefresh() processHandlerException() render()

	MG | DP
Module	**View**
imports	HttpServletRequest, HttpServletResponse
exportsproc	render()

	MG DP
Module	**AbstractView inherits from** WebApplicationObjectSupport
exportsproc	addStaticAttribute()
	createRequestContext()
	createTemporaryOutputStream()
	exposeModelAsRequestAttributes()
	generatesDownloadContent()
	getAttributesMap()
	getBeanName()
	getContentType()
	getRequestContextAttribute()
	getStaticAttributes()
	prepareResponse()
	render()
	renderMergedOutputModel()
	setAttributes()
	setAttributesCSV()
	setAttributesMap()
	setBeanName()
	setContentType()
	setRequestContextAttribute()
	toString()

	MG
Module	**ModelAndView inherits from** Object
imports	View, Object, Map, String
exportsproc	addAllObjects()
	addObject()
	addObject()
	clear()
	getModel()
	getModelInternal()
	getModelMap()
	getView()
	getViewName()
	hasView()
	isEmpty()
	isReference()
	setView()
	setViewName()
	toString()
	wasCleared()

Guía de Módulos:

Module Bean

Es una interfaz abstracta que provee los métodos principales Set y Get del modelo que esta siendo representado.

Module Controller
Es una interfaz que provee métodos para procesar la solicitud recibida del DispatcherServlet y devuelve un ModelAndView apropiado.

Module AbstractController
Es una superclase para implementaciones de controladores. El AbstractController es uno de los controlador base más importante, proporciona funciones básicas tales como la generación de cabeceras de almacenamiento en caché y la activación o desactivación de los métodos de apoyo (GET / POST).

Module MultiActionController
Es una implementación de controller que permite múltiples solicitudes, que pueden ser manejadas por la misma clase. Subclases de esta clase pueden manejar diferentes tipos de solicitudes con métodos de la forma:
public (ModelAndView / Map / String / void) actionName(HttpServletRequest request, HttpServletResponse response); Esta clase es capaz de mapear solicitudes con nombres de métodos y luego invocar el método correcto para manejar una solicitud particular.

Module ViewResolver
Es el encargado de realizar el mapping entre el nombre lógico de la vista y el componente(Modelo). Esta interfaz debe ser implementada por aquellos objetos que pueden resolver una vista por su nombre.

Module ThemeResolver
Es una interfaz que posee estrategias de resolución de temas basados en Web, permite la resolución de un tema vía solicitud o la modificación de temas vía solicitud y respuesta.

Module HandlerAdapter
Es una interfaz que debe ser implementada por cada tipo de Handler para manejar cada request. Esta interfaz es utilizada para permitir al DispatcherServlet ser extensible.

Module HandlerMapping
Analiza cada petición y determina el controlador que la gestiona. Podemos contar con varios manejadores, en función de las diversas estrategias de "mapeo" (basado en cookies, variables de sesión, etc.). En la mayor parte de los casos nos sirve el manejador por defecto de Spring.

Module View
Es responsable de representar el contenido, y exponer el modelo.

Module AbstractView
Esta clase proporciona soporte para los atributos estáticos, que se pondrá a disposición de la vista, con una variedad de maneras de especificar. Los atributos estáticos se fusionará con los atributos dados dinámico (el modelo que el controlador devuelve) para cada operación de procesamiento.

Module ModelAndView
Compone el modelo y la vista. Mantiene ambas para hacer posible al controlador, devolver el modelo y la vista en un único valor de retorno. Representa el modelo y la vista devuelta por un handler, y que luego pueda ser resuelta por el DispatcherServlet(FronController).

Module DispatcherServlet
Recibe y gestiona todas las peticiones (request), delegando a otro componente su procesamiento. Resulta oculto al programador y es instanciado por Spring.

CAPÍTULO 4

Sakai con Contratos

4.1. Sakai con Contratos ScC

El proyecto Sakai brinda una de las propuestas más consolidadas de diseño y desarrollo de entornos colaborativos e-learning para educación, orientado a herramientas que se implementan a través de servicios comunes (servicios bases). Por ejemplo, el servicio de edición de mensajes es utilizado en las herramientas Foro, Anuncio, Blog, PortFolio, etc. Más aun, otras de las características salientes de Sakai es la versatilidad para su extensión y/o configuración. En efecto, Sakai permite alterar ciertas configuraciones en tiempo de ejecución, por ejemplo, instrumentar una nueva funcionalidad en un servicio base.

Sin embargo, estas soluciones no pueden resolver aquellos procesos e-learning que involucren cambios en el comportamiento de la relaciones entre un componente (cliente) que ocasiona, a través de un pedido, la ejecución de un componente servidor (proveedor). Estos tipos de cambios refieren a la capacidad de adaptación dinámica del sistema extendiendo, personalizando y mejorando los servicios sin la necesidad de recompilar y/o reiniciar el sistema.

Tomando en cuenta la propuesta realizada por Alejandro Sartorio [7], donde propone la incorporación (agregado) de propiedades de adaptación dinámicas a los servicios bases del Framework Sakai, especialmente diseñadas para implementar procesos e-learning donde se requieren nuevos aspectos de adaptación en la perspectiva de los Dispositivos Hipermediales Dinámicos en el campo del e-learning con característica context-aware.

A continuación se inicia el recorrido de creación de un nuevo diseño de arquitectura

Sakai que contemple la incorporación de propiedades de adaptación dinámica a los servicios bases del Framework Sakai junto con CSC.

4.2. Hacia la documentación del estilo arquitectónico Sakai

En esta sección documentaremos el sistema estratificado sin tener en cuenta los métodos o servicios provistos por Sakai. Consideramos que presentar un vista abstracta es mas elegante, donde el simple reemplazo de cada request por un servicio concreto de una herramienta dada, llevaría a un ejemplo particular de la herramienta Sakai. (ej, reemplazar un request() por un editMessage() de la herramienta foro).

Diagrama Canónico:

El framework Sakai, como ya mencionamos, está diseñado según una arquitectura de cuatro capas: La capa de agregación, presentación, herramientas y servicios. La propuesta consiste en envolver los servicios del núcleo Sakai mediante la coordinación de contratos. Esta propuesta altera el diseño original del framework Sakai, ya que debemos agregar y modificar capas que nos permitan la inyección de información de contexto y el agregado de CSC. Para esto debimos modificar la capa de Servicios, dividiéndola en tres partes:

- Servicios Originales: Pertenecientes al núcleo del framework original, no afectados con el agregado del mecanismo de coordinación de contrato.
- Servicios de Contexto: Permite a clientes el acceso a entidades, asignar, obtener y subscribir cambios en la información de contexto de las entidades.
- Servicios con coordinación de contratos (Servicios CSC): Servicios base del núcleo del framework Sakai modicados para poder efectuar la envoltura de los mecanismos de coordinación.

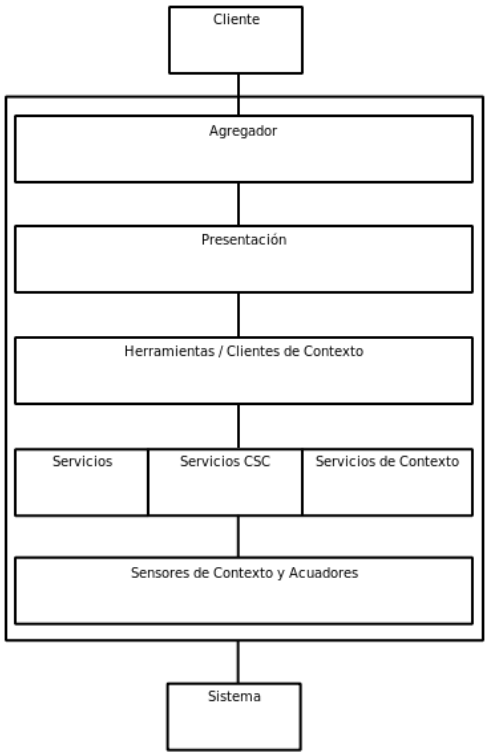


Figura 4.1: Framework Sakai con Contratos

Mediante la división del estrato servicios se puede interpretar a los servicios CSC como una nueva arquitectura basada en sistemas estraticados [19]. Entonces, esta composición se efectúa por el estrato de coordinación de contratos y el estrato de cómputo. La capa de cómputo estará compuesta por módulos de implementación (ej., servicios Sakai previos a la incorporación de contratos). Mientras que la capa de coordinación estará compuesta por módulos especícos de coordinación, patrones tipo proxy y contratos.

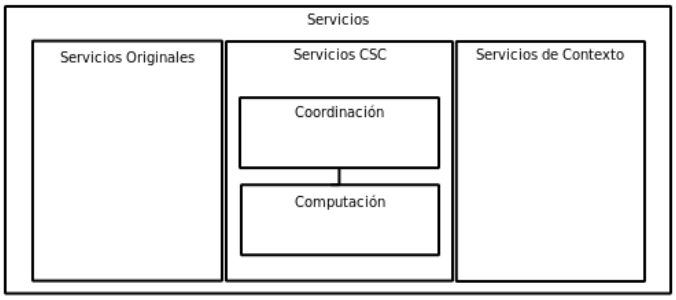


Figura 4.2: Estrato Servicios Sakai

De esta forma cuando se implemente la inyección de CSC en el código fuente de Sakai no se deberán realizar grandes modificaciones al código existente. Únicamente se incluye los módulos de coordinación, se realiza las comunicaciones entre ellos y los servicios bases existentes.

Los servicios CSC se implementaran utilizando un patrón de diseño de coordinación de contratos ("Coordination Contracts Design Pattern") tomando como referencia la propuesta de Fiadeiro [13; 18]. Este patrón esta basado en el patrón de diseño "proxy" (o "Surrogate")[8]. Por un lado provee una interfaz especíca ("SubjectInterface"), como una clase abstracta, para cada componente. Esta interfaz esta conectada al programa real ("SubjectBody") a través de un proxy dinámico reconfigurable. Por otra parte, soporta la reconguración dinámica del código ejecutado por medio de solicitud de operaciones a través del "proxy".

El ciclo de vida de una petición realizada por un cliente puede desarrollarse de dos formas distintas, con o sin coordinación de contratos.

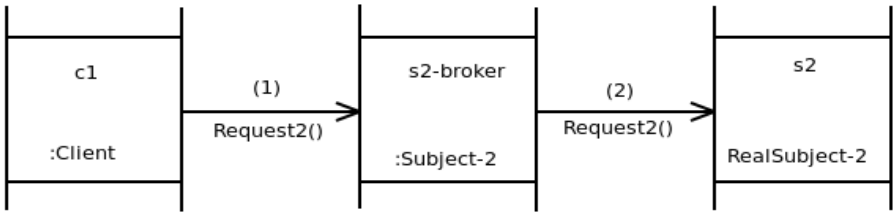


Figura 4.3: Proceso de Solicitud Sin coordinación de Contratos

En este escenario, donde el objeto *s2* no se encuentra baja coordinación, la única diferencia impuesta por el patrón, es una llamada extra del broker al objeto real. Introducir un contrato para coordinar la interacción con el objeto real *s2* solo implica realizar modificaciones sobre el objeto *s2-broker*, haciendo que su proxy se convierta en una referencia para el objeto *c2*(de tipo *PartnerConnector-2*). Haciendo esta modificación menor, el cliente ni los objetos reales *s2* necesitan ser modificados para adaptarse a este nuevo comportamiento establecido al agregar contratos.

El nuevo comportamiento introducido por contratos es descripto en la figura 4.4. En esta se muestra como un contrato introduce un nuevo comportamiento cuando una solicitud de tipo Request2() es invocada sobre un objeto real *s2*.

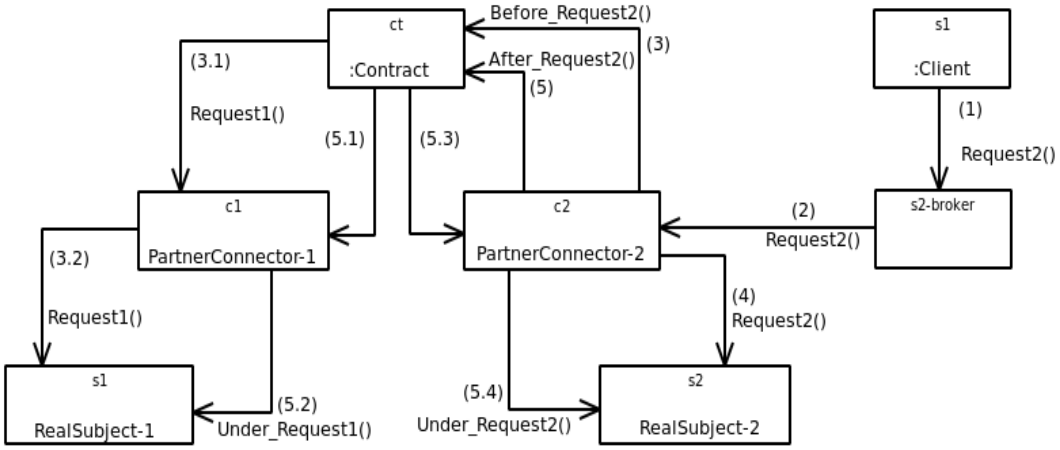


Figura 4.4: Proceso de Solicitud Con coordinación de Contratos

Una vez realizado el envío del Request2() al s2-broker, delega la ejecución a la referencia proxy(En este caso sobre c2 en vez de s2, como se vio en la figura 4.4.)

En c2 la implementación del servicio Subject tiene el siguiente formato:

```
(2):    Request2() is {
(3):        If ( contract.Before_Request2() = TRUE ) Then {
(4):            implementation.Request2();
(5):            If (contract.After_Request2() = FALSE )
                Then implementation.Undo_Request2();
            }
        }
```

Es decir, antes que PartnerConnector *c2* de permiso al objeto real *s2* que ejecute la petición, intercepta la solicitud y da derecho al contrato a decidir si la solicitud es valida y a realizar otras acciones. Esta intercepción nos permite imponer otras obligaciones contractuales a la interacción entre la persona que llama y el destinatario de la llamada. También permite al contrato realizar otras acciones antes y después que el objeto real ejecute el pedido. Solo si el contrato lo autoriza, el conector puede pedir a *s2* realizar la acción y el commit, o deshacer la ejecución por violación de una post condición establecida por el contrato.

Veamos ahora conceptos claves para comprender este nuevo Framework Context-Aware:

- **Servicios de Contexto(Context Services):** Un servicio recibe, maneja, almacena, y distribuye información de contexto para entidades. Varios servicios de contexto pueden cooperar en una forma peer-to-peer.
- **Entidades(Entities):** Una entidad modela algo que desea manejar para que contenga información de contexto. Un ejemplo de Entidad son personas, pacientes, un lugar, una TV, una PC, etc. Cada entidad tiene un contexto que esta formado por un conjunto de ítems de contexto y cada uno de ellos una relación con la entidad.
- **Clientes de Contexto(Context Client):** Clientes de contexto pueden presentar información de contexto y pueden escuchar cambios en la información de contexto de las entidades. Los clientes de contexto especializados en censar, resolver, y presentar información de contexto son denominados *Context Monitors*. Clientes de contexto especializados en utilizar información de contexto son llamados *Context Actuators*. Monitors y Actuators pueden ser registrados a uno o mas servicios de contexto y son notificados cuando necesitan proveer o utilizar información de contexto actualizada.

- **Eventos de Contexto(Context Event):**Un servicio de contexto permite clientes de contexto especiales(Entity Listener) para registrar interés en eventos de entidades especificas y recibir una notificación si ocurriesen dichos eventos.

Vale mencionar que el agregado del Contexto al Framework Sakai esta basado en los criterios y decisiones de diseño presentados en la infraestructura JCAF [23; 24]. Allí se presenta una Arquitectura JCAF diseñada en tres capas:

- Context Client
- Context Service
- Context Sensor and Actuator

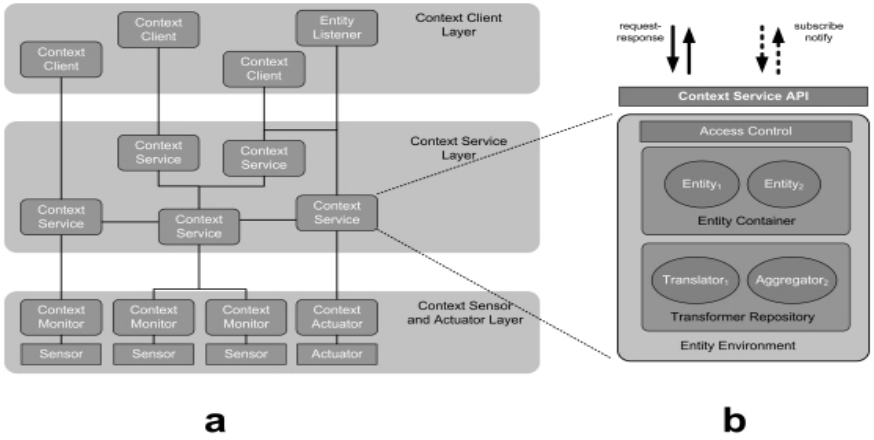


Figura 4.5: Arquitectura del Framework JCAF.

- **a**- Ejemplo de una situación de despliegue de un conjunto de monitores de contexto, actuadores, y Servicios de Contexto Cooperativos.
- **b**- Detalles de un Servicio de Contexto.

Cada una de estas capas han sido incorporadas o compuestas con los estratos nativos de Sakai con el propósito de incorporar JCAF al sistema general que compone el FWCsc. De esta manera agregando de esta forma información de contexto de una forma segura y ya estudiada.

JCAF incorpora conceptos de contribuciones previas sobre context-aware [25; 26; 27; 28; 29; 30; 31] para obtener un framework distintivo en al menos 3 formas:

- La infraestructura JCAFs orientada a servicios es un sistemas distribuido basado en la idea de dividir la adquisición, manejo, y distribución en una red cooperativa de servicios de contexto.
- JCAF es una infraestructura robusta, modificable, basada en eventos, que presenta una arquitectura segura.
- JCAF es genérica, extensible, y un modelo expresivo de programación Java para el despliegue y desarrollo de aplicaciones context-aware y modelos de contexto.

Siendo JCAF una infraestructura genérica, extensible, robusta y modificable, es posible establecer una correcta integración con Sakai. Además, existe homogeneidad desde el punto de vista tecnológicos.

La incorporación de las APIs JCAF al entorno Sakai permite a los programadores, crear aplicaciones sensibles al contexto integradas en Sakai. La figura 4.7 muestra el diagrama de clases e interfaces de las APIs pertenecientes a JCAF.

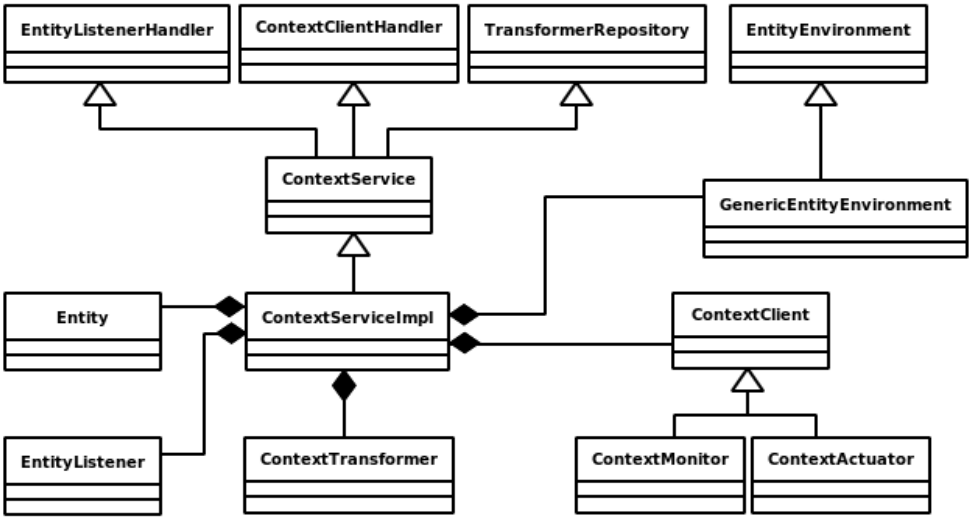


Figura 4.6: Diagrama de clases del Framework JCAF

Los *ContextService* manejan los objetos *Entity*, *EntityListeners*, y *ContextClients*. Dos ejemplos de clientes de contexto son *ContextMonitor* y *ContextActuators*. Cada uno de los servicios de contexto poseen un *EntityEnvironment*, esta interfaz contiene métodos para añadir y obtener atributos, acceder a información local de servicios de contexto, y acceder a repositorios de transformación. Veamos en detalles algunas de estas clases e interfaces.

Context Service y Entity Environment.

Un servicio de contexto (Context Service) permite a clientes acceder a entidades para

setear, obtener y subscribirse a cambios en la información de contexto de las entidades. La interfaz contextservice es presentada a continuación en la sección "Especificación de Interfaces". Esta interfaz provee métodos para agregar, remover, obtener y setear entidades. El método *getEntity()* devuelve una copia del servicio de contexto del objeto entidad, mientras que el método *lookupEntity()* contacta otro servicio de contexto conocido tratando de localizar el objeto entidad(Entity). El método *lookupEntity()* toma como argumento un ID de la entidad que se desea buscar. El método notifica al *EntityListener* si se encontró la entidad buscada.

El ContextService hereda de tres interfaces:

- **TransformerRepository:** Contiene *métodos para agregar* y obtener transformadores. Por ejemplo, un cliente puede utilizar el *método getContextTransformer()* para obtener un transformador, este puede transformar un array de ítems de contexto en otro ítem de contexto.
- **ContextClientHandler:** Contiene *métodos* para añadir y autentificar un cliente de contexto, que puede ser un ContextMonitor o un ContextActuator. Por ejemplo, addContextClient() Agrega un cliente de contexto, mientras que authenticate() lo autentifica
- **EntityListenerHandler:** Contiene *métodos* para añadir, remover y acceder a escuchadores de entidades(EntityListener). Por Ejemplo, *addEntityListener()* y vremoveEntityListener(), agregan y eliminan *EntityListeners*.

Entity y Context

El concepto de Entidad(Entity) es el mas básico dentro de la infraestructura JCAF, este posee un contexto con un conjunto de ítems de contexto(*ContextItem*). Estos son mostrados en la figura 4.7. Una entidad, un contexto y un ítem de contexto son todas interfaces Java, que los desarrolladores de las aplicaciones Context-Aware deben implementar.

Los *ContextItems* son agregados al contexto de entidades típicamente por monitores de contexto(*ContextMonitor*) u otros clientes.

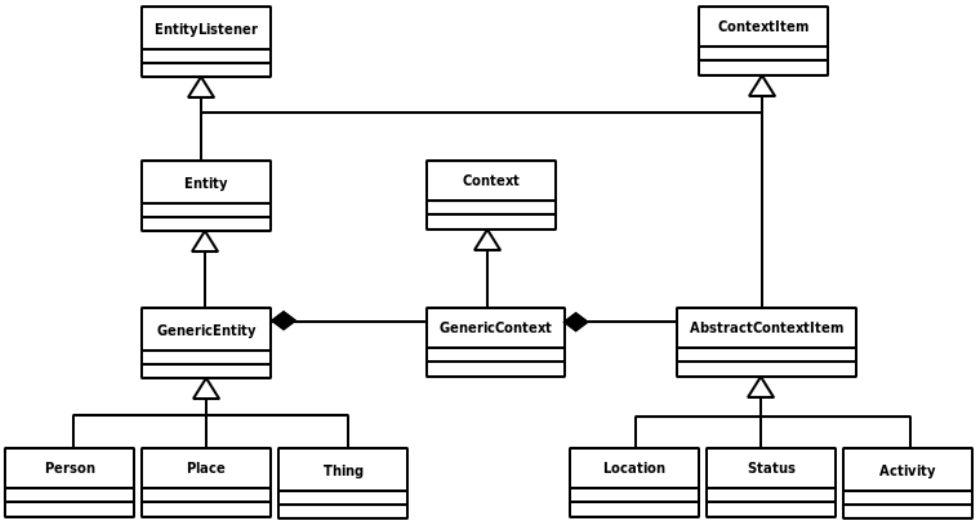


Figura 4.7: Diagrama de clases de una entidad con contexto.

Es importante poder juzgar la calidad de un ítem de contexto [31]. El método *getAccuracy()* de la interfaz *ContextItem* es utilizado para este propósito. La implementación de ítems de contexto devuelve una probabilidad entre cero y uno. El método *isSecure()* es utilizado para establecer si la información de contexto es original de un *ContextMonitor* confiable y autentificado.

EntityListener y ContextEvent

La infraestructura JCAF esta basada en eventos, que son propagados sobre cambios en las entidades de contexto hacia los clientes interesados. Los *EntityListener* son manejados por los *ContextServices*, que heredan de la interfaz *EntityListenerHandler*. Esta interfaz contiene métodos para agregar y remover *EntityListeners*.

Las entidades son conscientes de cambios en su contexto al implementar esta interfaz(EntityListener). La parte principal del procesado de una entidad es por lo tanto el método *contextChanged()*. Este método se garantiza que sera llamado por el contenedor de entidades cuando el contexto de esta entidad cambie.

El objeto ContextEvent es un estándar *java.util.EventObject* que brinda acceso a entidades e ítems de contexto, que causan el cambio.

Context Client - Monitors y Actuators

El framework JCAF puede manejar la adquisición y transformación de información de contexto de dos formas posibles, síncronamente y asincrónicamente. Un *ContextMonitor* puede solicitar continuamente información de contexto a una entidad utilizando el método *setContextItem()* en la interfaz de *ContextService*. La forma asincrónica es el modo

frecuente, sin embargo algunas aplicaciones de contexto pueden querer tener actualizado la información de contexto. Por lo tanto el framework JCAF provee un modo sincrónico, donde los clientes pueden solicitar información de contexto de una entidad, y la entidad pide a su contexto que se actualice.

Un monitor puede registrarse a un *ContextMonitorHandler* utilizando el método *addContextMonitor()*. Cuando un cliente solicita información de contexto, utilizando el método *getContext()*, los *ContextMonitors* registrados son llamados para adquirir la información de contexto llamando a su método *getContextItem()*. Cuando el *ContextMonitor* comienza a informar, el cliente es notificado utilizando el método *contextChanged()* en la interfaz *EntityListener*.

Context Transformers

Esta interfaz posee métodos como *getInType()* que dice que tipo de *ContextItem* este transformador puede tomar como entrada y el *getOutType()* dice que tipo de *ContextItem* devuelve. El método *traslate()* toma un array de ítems de contexto como entrada y devuelve un ítem de contexto traducido. Si se encuentra solo un ítem de contexto en el array de entrada, el transformador trabaja como un traslator(Traductor) traduciendo de un ítem de contexto a otro. Si se encuentran mas de un ítem, el transformador trabaja como un agregador(aggregator), agregando distintos ítems de contexto en uno.

Un Repositorio de Transformadores contiene un rango de transformadores. Un cliente puede utilizar el método *getContextTransformer()* para obtener un transformador, el cual puede transformar una arreglo de ítem de contexto en otro ítem de contexto.

Especificación de Interfaces:

En este apartado mostraremos los módulos 2MIL del estrato servicios, sensores de contexto y actuadores de nuestra arquitectura, donde solo especificaremos las clases y métodos del modulo contratos. Esto se debe a que la mayoría de los modulo servicios fueron explicado en la arquitectura nativa de Sakai.

		MG	DP
Module	**Servicios y Servicios de Contextos**		
comprises	Servicios, Servicios CSC, Servicios de Contexto		

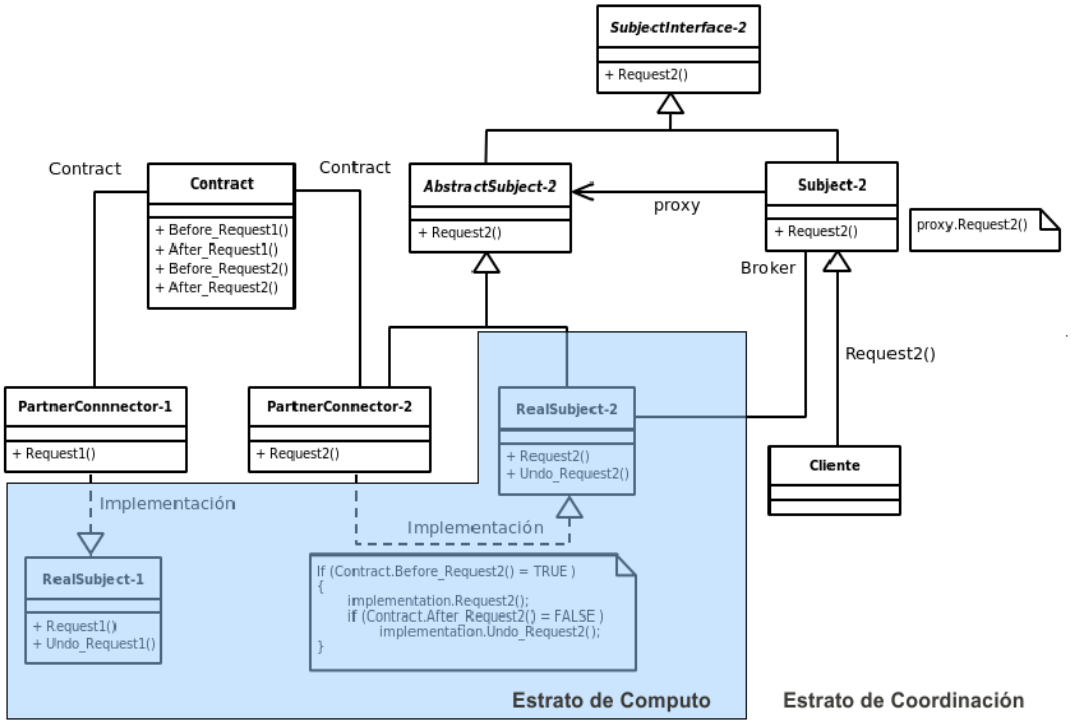


Figura 4.8: Diagrama de clases del estrato Servicios Sakai con CSC.

	MG | DP
Module	**Servicios de Contexto**
comprises	EntityListenerHandler, ContextClientHandler, TransformerRepository, ContextService, ContextServiceImpl, Entity, EntityListener, EntityEnvironment, GenericEntityEnviroment, ContextTransformer

	MG | DP
Module	**Servicios CSC**
comprises	Coordinación, Computo

	MG | DP
Module	**Coordinación**
comprises	SubjectInterfaz-2, Subject-2, AbstractSubject-2, Contract, PartnerConnector-2, PartnerConnector-1

	MG | DP
Module	**Computo**
comprises	RealSubject-1, RealSubject-2

	MG | DP
Module	**SubjectInterfaz-2**
exportsproc	Request2()

	MG | DP
Module	**Subject-2 inherits from** SubjectInterfaz-2
exportsproc	Request2()

	MG | DP
Module	**AbstractSubject-2 inherits from** SubjectInterfaz-2
exportsproc	Request2()

	MG | DP
Module	**RealSubject-2 inherits from** AbstractSubject-2
exportsproc	Request2()
	Undo_Request2()

	MG | DP
Module	**PartnerConnector-2 inherits from** AbstractSubject-2
exportsproc	Request2()

		MG	DP
Module	**Contract**		
exportsproc	After_Request1()		
	Before_Request1()		
	After_Request2()		
	Before_Request2()		

		MG	DP
Module	**PartnerConnector-1**		
exportsproc	Request1()		

		MG	DP
Module	**RealSubject-1**		
exportsproc	Request1()		
	Undo_Request1()		

		MG	DP
Module	**EntityListenerHandler**		
exportsproc	addEntityListener()		
	removeEntityListener()		

		MG	DP
Module	**ContextClientHandler**		
exportsproc	addContextClient()		
	addPublicKey()		
	authenticate()		
	removeContextClient()		

	MG | DP
Module	**TransformerRepository**
exportsproc	addContextTransformer()
	getContextTransformer()
	removeContextTransformer()

	MG | DP
Module	**ContextServiceImpl inherits from** ContextService
imports	Entity, EntityListener, ContextTransformer, ContextClient, GenericEntityEnviroment
exportsproc	setContextItem()
	addContextClient()
	addContextItem()
	addContextService()
	addContextTransformer()
	addEntity()
	addPublicKey()
	authenticate()
	getAllEntities()
	getAllEntitiesByType()
	getAllEntityIds()
	getContext()
	getContextTransformer()
	getEntity()
	getName()
	getPeerManager()
	getServerInfo()
	getURI()
	lookupEntity()
	main()
	removeContextClient()
	removeContextItem()
	removeContextService()
	removeContextTransformer()
	removeEntity()
	removeEntityListener()
	restart()
	setEntity()

		MG	DP
Module	**ContextService inherits from** TransformerRepository		
	addContextItem()		
	addEntity()		
	getAllEntities()		
	getAllEntitiesByType()		
	getAllEntityIds()		
	getContext()		
	getEntity()		
	getName()		
	getServerInfo()		
	getURI()		
	removeContextItem()		
	removeEntity()		
	setEntity()		

		MG	DP
Module	**EntityListener**		
exportsproc	contextChanged()		

		MG	DP
Module	**Entity inherits from** EntityListener		
	ContextItem		
exportsproc	addEntityListener()		
	destroy()		
	getContext()		
	getEntityConfig()		
	getEntityInfo()		
	getId()		
	init()		
	notifyListeners()		
	refreshContext()		
	removeEntityListener()		
	setContext(Context context)		

	MG | DP
Module	**GenericEntity inherits from** Entity
imports	GenericContext
exportsproc	addEntityListener()
	contextChanged()
	destroy()
	getContext()
	getContextService()
	getEntityConfig()
	getEntityEnvironment()
	getEntityInfo()
	getEntityName()
	getId()
	getInitParameter()
	getInitParameterNames()
	init()
	notifyListeners()
	refreshContext()
	removeEntityListener()
	setContext()
	setId()
	toXML()

	MG | DP
Module	**Person inherits from** GenericEntity
exportsproc	contextChanged()
	getEntityInfo()
	getLocation()
	getName()
	getNote()
	getPublicKey()
	setLocation()
	setName()
	setNote()
	setPublicKey()
	toString()
	toXML()

	MG | DP
Module	**Place inherits from** GenericEntity
exportsproc	getEntityInfo()
	getName()
	setName()
	toXML()

	MG | DP
Module	**ContextItem**
exportsproc	equals()
	getAccuracy()
	getSequenceID()
	isSecure()
	setSecure()

	MG | DP
Module	**Context**
exportsproc	contains()
	containsRelationship()
	getContextItem()
	getId()
	getItems()
	getRelationships()
	getRelationshipsTypes()
	removeContextItem()
	setContextItem()
	toXML()

	MG | DP
Module	**GenericContext inherits from** Context
imports	AbstractContextItem
exportsproc	addEntityListener()
	contextChanged()
	destroy()
	getContext()
	getContextService()
	getEntityConfig()
	getEntityEnvironment()
	getEntityInfo()
	getEntityName()
	getId()
	getInitParameter()
	getInitParameterNames()
	init()
	notifyListeners()
	refreshContext()
	removeEntityListener()
	setContext()
	setId()
	toXML()

	MG | DP
Module	**AbstractContextItem inherits from** ContextItem
imports	AbstractContextItem
exportsproc	equals()
	getAccuracy()
	getId()
	getSequenceID()
	isSecure()
	setSecure()

		MG	DP
Module	**Location inherits from** AbstractContextItem		
exportsproc	equals()		
	getAccuracy()		
	getLocation()		
	toString()		
	toXML()		

		MG	DP
Module	**ContextTransformer**		
exportsproc	getInType()		
	getOutType()		
	translate()		

		MG	DP
Module	**Sensores de Contexto y Actuadores**		
exportsproc	ContextMonitor, ContextActuator, ContextClient		

		MG	DP
Module	**ContextClient**		
exportsproc	getContextItem()		
	contextItemChanged()		

		MG	DP
Module	**ContextMonitor inherits from** ContextClient		
exportsproc	getContextItem()		

		MG	DP
Module	**ContextActuator inherits from** ContextClient		
exportsproc	contextItemChanged()		

Guía de Módulos:

Module Servicios y Servicios de Contexto

Este módulo contiene los módulos que deben ser modificados si cambia algún servicio Sakai, Servicios de Contexto, Servicios Bases o Servicios CSC.

Module Servicios de Contexto

Este módulo contiene los módulos que deben ser modificados si cambia algún servicio de contexto, o implementación de contexto.

Module Servicios CSC

Este módulo contiene los módulos que deben ser modificados si cambia algún servicio o la coordinación de contratos de dichos servicios.

Module Coordinación

Este módulo contiene los módulos que deben ser modificados si se reemplaza alguna clase de coordinación. Los submódulos de este módulo proveen los métodos para administrar la coordinación de contratos sensibles al contexto.

Module Computo

Este módulo contiene los módulos que deben ser modificados si se reemplaza algún servicio de Sakai. Los submódulos de este módulo proveen los servicios básicos. Los secretos ocultos en estos módulos son las diversas implementaciones de los distintos servicios provistos.

Module SubjectInterfaz-i

Como su nombre lo indica, esta es una clase abstracta que define una interfaz común de los servicios de una Herramienta provisto por *MétodoHerramientaProxy* y *Subject-2*.

Module Subject-i

Esta es una clase concreta que implementa un broker(Intermediario) manteniendo la referencia a la clase Método Proxy para encargarse de los pedidos recibidos. En tiempo de ejecución, esta entidad puede indicar a *MétodoServiceComponente* que no hay contratos involucrados o instrumentar una conexión entre *ConectorMétodosService* que conecte la clase real *MétodoServiceComponente* con el contrato ContractMétodo en el que se encuentran las reglas de coordinación.

Module AbstractSubject-i

Es una clase abstracta que define la interfaz que tienen en común *RealSubject-i* y *PartnerConnector-i*. La interfaz es heredada de *SubjectInterfaz-i* con el propósito de garantizar que ofrezca la misma interfaz de *Subject-i*(el broker) con la cual un cliente

real debe interactuar (ej., un objeto que invoque el servicio de edición de un Foro).

Module PartnerConnector-i

Efectúa la conexión entre el contrato y los objetos reales (en este caso *RealSubject-i*) involucrados como partes del contrato. De esta manera, no se requiere la creación o instanciación de nuevos objetos para coordinar el mismo objeto real agregando o sacando otros contratos. Esto es posible solamente con una nueva asociación a un nuevo contrato y con la instanciación de una conexión con una instancia existente de *PartnerConnector-i*. Esto significa que hay una sola instancia de esta clase asociada con una instancia de *RealSubject-i*.

Module Contract

Es la clase cuya instancia genera un objeto de coordinación que al ser notificado toma decisiones siempre que un pedido es invocado a través de un objeto real. La clase que implementa al contrato, llamada *Contract*, se encuentra representada dentro del estrato de coordinación.

Module EntityListenerHandler

Este modulo es una interfaz publica que define a un manejador de *Entity Listener*. Esta interfaz contiene métodos para agregar, remover, y acceso a *Entities Listeners*.

Module ContextClientHandler

Este modulo es una interfaz publica que define un manejador de Contexto de cliente(Context Client Handler). Un *Context Client Handler* es capaz de manejar *Context Monitors* remotos y *Context Actuators*, y llamarlos cuando sean necesarios. Servicios de contexto seguros son capaces de registrarse un manejador de servicios de contexto(context service handler) utilizando el método *authenticate()*. Context monitors seguros pueden agregar ítems de contexto seguros a un contexto.

Module TransformerRepository

Este modulo es una interfaz publica que contiene un rango de transformadores de contexto(Context Transformer), que pueden transformar de uno o mas tipos de información de contexto hacia otro. Se puede pedir al repositorio para obtener un transformador, proviendo el tipo de interfaz al transformador.

Module ContextService

Este modulo es una interfaz publica que define un servicio de contexto(Context Service). Esta es la interfaz remota que se vuelve accesible cuando los servicios de contexto son buscados en el registro RMI.

Module ContextServiceImpl

Este modulo contiene la implementación RMI de la interfaz *ContextService*.

Module Entity

Esta interfaz define una entidad del mundo real que desea ser modelada en orden de seguir los rastros de su contexto. Los objetos que implementan esta interfaz es por lo tanto el objeto principal en el ContextService. Esta interfaz define métodos que todas las entidades deben implementar.
Entidades reciben y responden a pedidos de clientes, usualmente a través de RMI. Además ellos son notificados si su contexto cambio, debido a que una *Entity* extiende la interfaz de *EntityListener*. Esta interfaz define métodos para inicializar una entidad, solicitudes de servicio, y remover una entidad del servicio de contexto.

Module EntityListener

Este modulo es una interfaz publica que recibe notificaciones de cambio de contexto de una entidad. Esta es la versión local de una interfaz Listener.

Module Person

Esta clase representa a una persona, con un id, un nombre, una nota, y una clave publica a ser usada.

Module Place

Esta clase implementa un lugar utilizando un id

Module Thing

Esta clase representa un elemento o algo que debe identificarse utilizando un id.

Module Context

Esta clase representa un simple Contexto para una entidad(Entity), que contiene varios tipos de Ítems de Contexto(Context Item). Toda la información de contexto es encapsulada en este objeto Contexto, el cual es serializable.

Module ContextItem

Este modulo es una interfaz publica que define un ítem de contexto. Esta interfaz debe ser implementada para ítem específicos como ubicación, actividad, estado, temperatura, etc.

Module AbstractContextItem

Esta clase implementa la mayoría de la funcionalidad de un ítem de contexto y por lo tanto adecuado para sub-clasificación cuando se crean ítem de contexto personalizados.

Module Location

Esta clase implementa la ubicación de un ítem de contexto.

Module Status

Esta clase implementa el estado de un ítem de contexto.

Module Activity

Esta clase implementa la actividad de un ítem de contexto.

Module ContextTransformer

Esta interfaz define un transformador de Contexto. La clases que implementan esta interfaz pueden traducir de un tipo de ContextItem a otro.

Module ContextClient

Esta clase implementa Clientes de contexto que pueden ser utilizadas para crear clientes como monitores o actuador(monitors or actuators).

Module ContextMonitor

Esta clase define un monitor de contexto(Context Monitor). En esta clase se implementan los métodos para registrarse a un *ContextMonitorHandler*, y métodos para adquirir información de contexto.

Module ContextActuator

Esta clase define un actuador de contexto(Context Actuator). En esta clase se implementan los métodos que deben ser llamados cuando un contexto que el actuador esta escuchando cambia.

CAPÍTULO 5

Variantes de Diseño

En este capítulo brindaremos opciones de diseño y desarrollo para el framework Sakai. Aquí se presentaran diferentes alternativas para graficar diagramas estratificados con el fin de evitar confusiones de interpretación y diseño. También presentaremos una alternativa de desarrollo donde un conjunto de servicios puede ser presentado como un único servicio a través de la composición dinámica de servicios.

5.1. Variantes al Documentar Sistemas Estratificados

En esta sección brindaremos las teorías presentadas en [40] para poder extender la documentación de los *Sistemas Estratificados* ya presentadas y dar un abanico de elecciones al documentarlo, de esta forma se provee al diseñador alternativas de diseño para una documentación detallada y concisa.

Notaciones Informales: Dentro de las notaciones informales utilizadas para la documentación de un sistema estratificado, podemos encontrar pilas y anillos. Comúnmente, los estratos son dibujados como pilas, donde un estrato se ubica sobre otro y sus adyacencias denotan la relación *"Allowed to Use"* (Permite utilizar), esta relación también puede ser descripta utilizando flechas.

Los anillos son una variante a las pilas, estos son representados como un conjunto de círculos concéntricos, donde el anillo mas interno se corresponde con la capa inferior de una pila y el mas externo con la superior. A su vez cada anillo puede ser subdividido en partes o sectores, así como también lo hace un capa.

No existe diferencia semántica entre un diagrama de pila y uno de anillos, salvo por un caso. En el diagrama de anillos de la figura 5.1, se asume que cada segmentos de anillo adyacentes entre si implica una relación de *"Allowed to Use"*. Lo mismo sucede en un diagrama de pila, sin embargo un diagrama de anillos posee mas adyacencias que uno de pila.

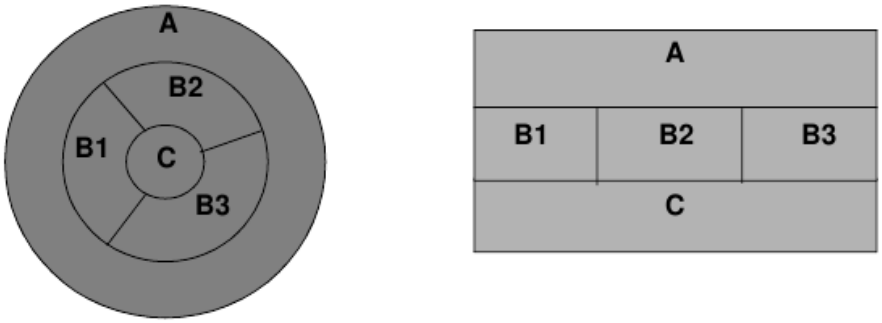


Figura 5.1: Comparación de Diagramas de Anillos y Pilas

Aquí podemos apreciar que el diagrama de anillo muestra una adyacencia entre los segmentos *B1* y *B3*, mientras que el diagrama de Pila no lo hace. Por lo tanto no existe forma de representar un diagrama de anillos como un diagrama de pila, como se mostró en la figura 5.1.

En el capitulo anterior, propusimos un sistema estratificado Sakai, donde uno de sus estratos se encuentra sub-estratificado o dividido en tres segmentos, el diagrama propuesto podría parecer equivalente a un diagrama en anillo, pero no lo es. Esto se debe a que un diagrama en anillos provee una relación no deseada(Servicios originales y servicios CSC) ya que un servicio que se encuentre entre los originales del núcleo Sakai y no haya sido modificado para implementar la coordinación de contratos, no necesitara comunicación alguna con el segmento servicios CSC y si lo hará con servicios de contexto.

La figura 5.2 mostrara que el diagrama propuesto en la sección anterior posee una diferencia semántica con uno de anillos. Esta diferencia se encuentra en las adyacencias que presentan cada segmento o subdivisión del estrato.

Mas adelante se proveerá un nuevo diagrama de pila, donde las conexiones entre segmentos se harán mas explicitas y evitaran confusiones.

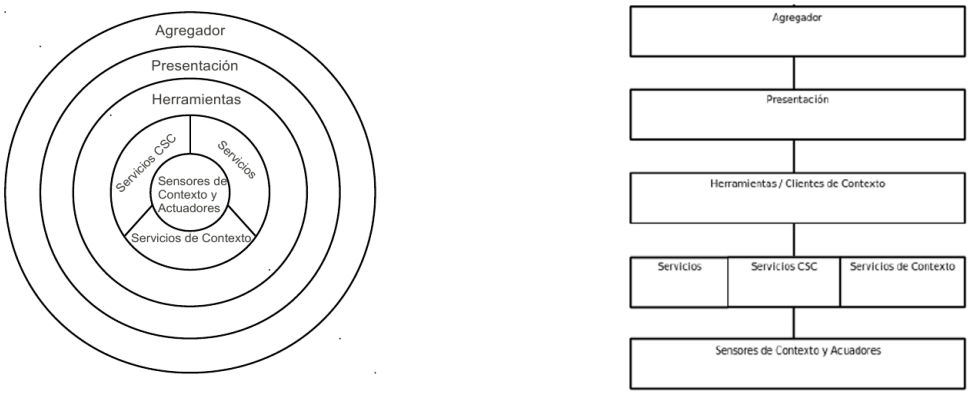


Figura 5.2: Diagrama en Anillo y Pila de la Arquitectura Sakai con CSC

Notaciones Formales: UML representa capas utilizando paquetes(Packages). Un paquete es un mecanismo general para organizar elementos en grupos. UML tiene predefinido tipos de paquetes para sistemas y subsistemas. En [32] introduce un nuevo paquete definiéndolo como un estereotipo de paquete. Los estereotipos nos permiten adaptar la definición de un elemento UML, por lo general mediante la adición de restricciones adicionales y/o cambiar la notación visual. Una capa puede ser vista como un paquete UML con la restricción que sus unidades de grupo de software y las dependencias entre paquetes sea *"Allowed to Use"*. Podemos designar una capa utilizando la notación de paquete con junto con el nombre del estereotipo «Layer» precediendo el nombre de la capa.

La relación *"Allowed to Use"* puede ser representada como un estereotipo de la dependencia de acceso UML, una de las dependencias existentes entre paquetes. Esta dependencia permite a los elementos de un paquete referenciar a elementos de otro paquete. Mas precisamente:

- Un elemento dentro de un paquete tiene permitido acceder a otros elementos del paquete
- Si un paquete accede a otro, entonces todos los elementos definidos con una visibilidad publica en el paquete accedido son visibles en el paquete que lo importa.
- Las dependencias de acceso no son transitivas. Si el paquete 1 puede acceder al paquete 2 y este puede acceder al 3, no significa que el paquete 1 pueda acceder al 3.

Los paquetes también representan interfaces de capas. Los paquetes permiten 3 tipos de descripciones de interfaces:

- La suma de todas las interfaces de los elementos.
- Un conjunto de interfaces de sus elementos.
- Una interfaz de capa separada. En este caso, uno puede definir la interfaz como una colección de clases dentro de un paquete (utilizando el patrón Facade).

Los paquetes sirven para satisfacer todas las necesidades de documentación sobre estratos: Diagrama de capas, claves, excepciones, catalogo de capas, y interfaces de capas.

Variaciones: Entre las variantes que se encuentran en un diagrama de capas, podemos mencionar las siguientes:

1. Alcance. La variación más común en las capas son las convenciones que determinan el alcance de la relación *"Allowed to Use"* de cada capa. Normalmente, estos convenciones permiten al software de una capa a utilizar el software de

   a) La capa inferior mas cercana

   b) Cualquier capa inferior

   c) Cualquier capa adyacente, inferior o superior

2. Capas laterales. Muchas de las arquitecturas estratificadas lucen en la figura 5.3. Esto puede significar una de dos cosas. El software en *D* puede acceder al software

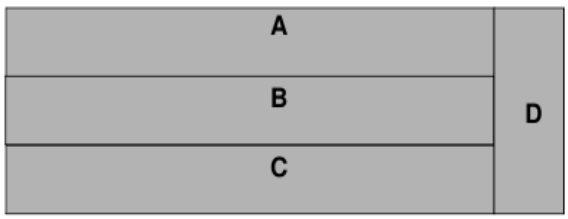


Figura 5.3: Sistema Estratificado con estrato lateral

en *A*, *B* y *C*, o el software en *A*, *B* o *C* puede utilizar el software de *D*. (Técnicamente, el diagrama dice que ambas son ciertas, sin embargo esto representa una arquitectura pobre.). Es responsabilidad del creador del diagrama especificar cual caso representa.

Una variante como esta, solo tiene sentido cuando las reglas de uso son de un solo nivel, eso significa que el estrato *A* solo utiliza el *B* y ningún otro inferior. De lo contrario, *D* podría ser el estrato superior o inferior, dependiendo del caso y la geometría lateral seria innecesaria.

3. Segmentación de estratos. A veces los estratos son divididos en segmentos denotando una descomposición de software. Cuando esto sucede, es responsabilidad del creador del diagrama decir que reglas de uso existen entre los segmentos. Debe especificar si cada segmento puede utilizar los restantes, también debe indicar que relación existe en caso que la capa inferior también se encuentre subdividida. En

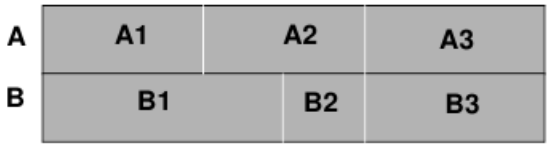


Figura 5.4: Sistema estratificado sin especificación formal de relaciones existentes

la figura 5.4 no podemos apreciar que tipo de relaciones existen entre cada subdivisión. Una forma de enunciar esto es mediante la utilización de flechas, estas indican una relación *"Allowed to Use"* entre los estratos que la comprenden. La figura 5.5 representa vía flechas las relaciones entre estratos.

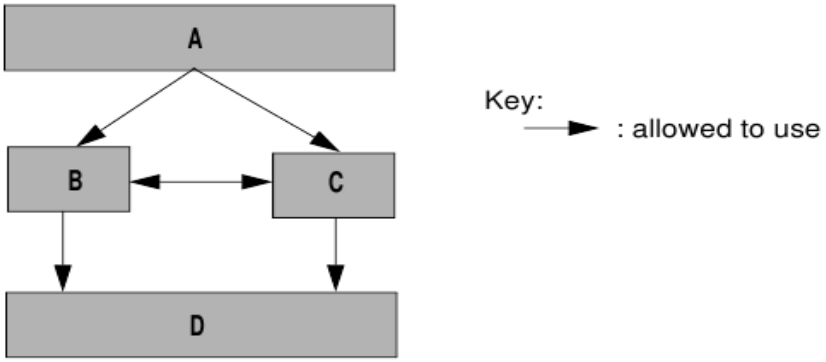
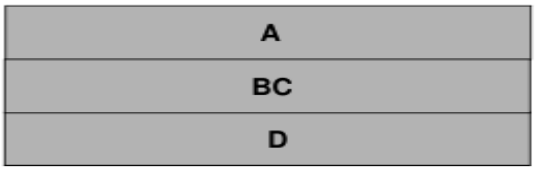


Figura 5.5: Sistema estratificado con flechas de relación

Aquí podemos apreciar que el estrato *A* tiene permitido utilizar tanto *B* como *C*, los cuales tienen permitido utilizar *D* y entre ellos. Este diagrama es equivalente al siguiente:



Donde el estrato *BC* es la unión de contenidos del estrato *B* y *C*. Esto se debe a que la relación representada por ambos es la misma. La descomposición del estrato *BC*, en *B* y *C*, brinda información adicional que nada tiene que ver con estratificación.

4. Estratificación a través de herencia. Esta es una variación que es consistente con el modelo básico de capas presentando en [32], pero interesado en el mundo orientado a objetos. Estratificación a través de herencia ocurre cuando una clase base se encuentra en un estrato inferior y la subclase que hereda la interfaz especificada por la clase base se encuentra en un estrato superior. La clase base forma parte de una maquina virtual que puede ser utilizada por mas aplicaciones para la creación de instancias de la clase base.

En el capitulo 4 describimos la nueva arquitectura Sakai y es presentada en la figura 4.1. En este diagrama no se puede inferir que tipo de relaciones existen entre los estratos *"Herramientas / Clientes de Contexto"*, *"Servicios y Servicios de contexto"*, y *"Sensores de contexto y Actuadores"*. En la figura 5.6. mostraremos una variante del diagrama que brinda mayor información de las relaciones existentes entre capas.

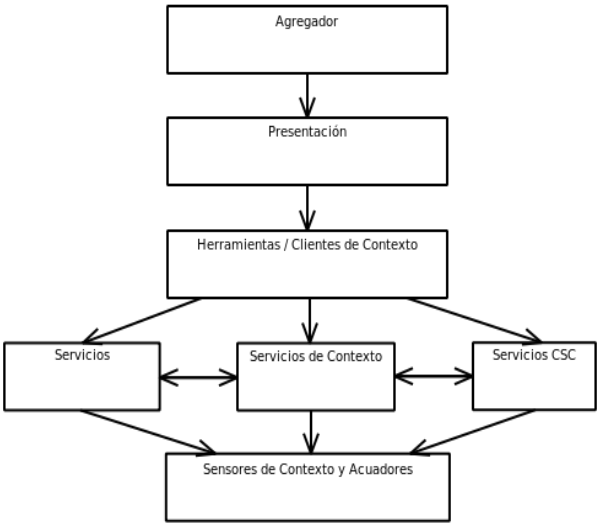


Figura 5.6: Arquitectura Sakai con flecha de relación

Aquí podemos apreciar que las herramientas tienen acceso a servicios de contexto, servicios CSC y servicios sin coordinación de contratos. Estos se comunican entre si para la obtención de información de contexto, así como también para la coordinación de contratos, de esta forma podemos hacer posible la extensión del framework Sakai.

5.2. Composición Dinámica de Componentes de Servicios

En esta sección comentaremos dos enfoques de composición dinámica de servicios introducida en [33] que puede ser utilizado en nuestra Arquitectura Sakai, y así proveer al estrato *Herramientas* una única interfaz de servicio resultante de la composición dinámica de los servicios de contexto, servicios originales del núcleo Sakai, los servicios CSC y

Servicios *"externos"*. Cuando hablamos de un Servicio *"Externo"* nos estamos refiriendo a un servicio no provisto por Sakai que puede utilizarse para brindar un servicio adicional. Por ejemplo, si un foro posee restricciones de acceso y publicación de información para aquellos alumnos que posean un numero menor a X materias aprobadas, podría utilizarse un servicio compuesto conformado por los servicios ya disponibles por Sakai y un servicio externo provisto por "Alumnado" para obtener el numero de materias rendidas por el alumno y así validar si satisface las restricciones del foro.

La utilización del framework propuesto por David Mennie, Bernard Pagurek[33], no solo nos brinda la posibilidad de utilizar servicios externos, si no que también nos permite ajustar, cambiar y modificar servicios en tiempo de ejecución. Estas características fueron también el motivo para introducir la coordinación de contratos sensibles al contexto, por lo que no resulta raro o útil intentar agregar las propiedades de este framework a la plataforma Sakai.

Existen dos enfoques para la composición dinámica de servicios:

- Interfaz Compuesta de Servicios(Composite Service Interface): Múltiples componentes de servicios se comunican entre si como entidades separadas para proveer un mejor servicio que es accesible a través de una interfaz común
- Servicio Compuesto Independiente(Stand-alone Composite Service): Los componentes de servicios son combinados en tiempo de ejecución para mejorar el servicio como una entidad única, autónoma.

El primer paso en crear un servicio compuesto, es ubicar los componentes de servicios que proveen la funcionalidad que esta por ser agregada al nuevo servicio. Todos los componentes de servicio deben ser almacenados en un directorio de componentes que pueden ser accesados en tiempo de ejecución. Cada componente debe tener una clara descripción de que operaciones puede llevar a cabo, que métodos pueden ser extraídos o utilizados en la creación de un compuesto, y los requerimientos de entrada y salida del componente. Una vez que los componentes fueron ubicados, se debe determinar que tipo de composición dinámica se realizara. Hay varias ventajas y desventajas asociadas a la selección de un método particular de composición. En algunos casos, mas de uno son posibles. Sin embargo, la elección del mejor método debe ser basada en como el servicio compuesto va a ser utilizado y la eficiencia del servicio resultante.

Crear una Interfaz Compuesta de Servicios La idea de este enfoque es crear una nueva interfaz que provea un único servicio compuesto a partir de un conjunto de servicios que colaboran entre si. En la arquitectura propuesta por David Mennie, Bernard Pagurek

hacen uso de una extensión del patrón de diseño Facade[8] para crear esta interfaz. Facade intenta proveer una interfaz unificada a un conjunto de componentes y manejar la delegación de peticiones entrantes para el componente apropiado. Sin embargo, esto es realizado en tiempo de diseño. El Facade dinámico facilita la actualización de la interfaz del servicio compuesto como un componente es agregado en tiempo de ejecución. Si los servicios pertenecientes de la composición no están situados en el mismo nodo de una red y están distribuidos en toda la red, se necesitan enviar mensajes entre componentes a través de la interfaz.

La ventaja de esta técnica es la velocidad a la cual el servicio compuesto es creado. Esto se debe a que ningún segmento de código de los componentes necesita ser movido o integrado en orden de asegurar la funcionalidad del servicio compuesto.

Esta técnica también es referida como un fusión de interfaces ya que la interfaz de cada uno de los componentes envueltos son fusionados en una única interfaz. Sin embargo, las interfaces del servicio 1 al n no pueden ser simplemente "pegadas". Se necesitan hacer modificaciones para dirigir correctamente los mensajes entrantes hacia el componente apropiado en el orden correcto.

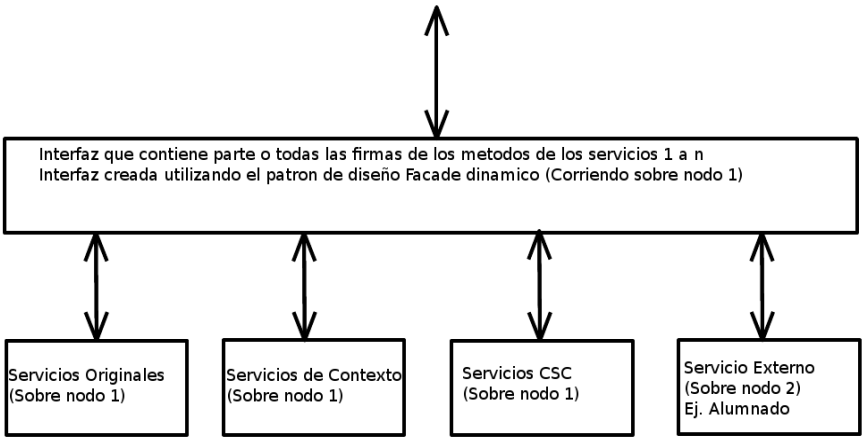


Figura 5.7: Interfaz de Servicio Compuesta sobre Servicios Sakai

La figura 5.2, es un ejemplo concreto de una interfaz de servicio compuesta sobre Sakai, donde el componente que corre sobre el nodo 2, provee servicios de Alumnado para consultas e historial sobre estudiantes. Aquí se puede ver que los servicios de núcleo Sakai, servicios de contexto y servicios CSC corren sobre el mismo nodo, mientras que el servicio de alumnado se encuentra en otro nodo de la red. La comunicación entre dichos nodos se realizara vía mensajes y suele ser menos eficiente en la performance debido a la demora en la comunicación de componentes, crear un Servicio compuesto independiente es una alternativa mas eficiente en performance.

Crear un Servicio Compuesto Independiente Si la performance de la composición de servicios es mas critica, crear un servicio independiente resulta mejor que una fusión. Existen dos medios para crear un servicio compuesto independiente.

El primer enfoque no lleva a ensamblar dinámicamente servicios de una forma similar a el ensamblamiento que se realiza con tubos y filtros [14]. La ventaja de este enfoque reside en que cada componente permanecen independientes. Esto significa que no deben compartir información de estados ni su dependencia con otros componentes. Existe una variante mas compleja que permite la realización de bucles ante la necesidad de ejecutar un componente mas de una vez. En este caso el diseño exige un fundamento sólido para explicar y entender el cálculo completo; un análisis teórico riguroso y el uso de especificaciones formales son apropiados para demostrar que el sistema termina y que produce un resultado deseado.

Otro enfoque posible es la creación de un nuevo servicio independiente. Aquí, la lógica del servicio, o el código de cada componente es ensamblado en un nuevo servicio compuesto. En general, no todo el código de cada componente puede ser reutilizado, ya que ciertos métodos son específicos de un servicio individual o no son útiles en el contexto del servicio compuesto. Por esta razón, los métodos compuestos son identificados en la especificación de cada servicio. Aunque la construcción de un servicio compuesto independiente en tiempo de ejecución es una tarea muy compleja, posee la ventaja principal que un un servicio compuesto independiente puede ser reutilizado y compuesto fácilmente con otros servicios.

Extensión de la propuesta a Sakai En este apartado daremos una breve descripción de como agregar las características de la arquitectura propuesta en [33] para extender nuestra plataforma Sakai a una plataforma distribuida. Para poder ofrecer esta variante de diseño, Sakai debería ser capaz de agregar las capacidades de composición de componentes y conexión entre componentes distribuidos. Para esto es necesario hacer uso de la tecnología de conexión Jini[34] como repositorio de servicios y sistema de recuperación. También es necesario implementar o utilizar el servicio creado por David Mennie, Bernard Pagurek, llamado *"Composition Manager"* para supervisar todos los aspectos de composición dinámica de servicios.

Asumimos que el lector esta familiarizado con el concepto de JavaBeans. Sin embargo, describiremos las características claves de ERCSP (Extensible Runtime Containment and Services Protocol). ERCSP provee una API que habilita la interconexión entre Beans en tiempo de ejecución. Permite a un Bean consultar a su entorno por ciertas capacidades y servicios disponibles. Esto permite al Bean ajustar dinámicamente su comportamiento al contenedor o el contexto en el que se encuentre. La API consiste en dos partes: un contenedor lógico jerárquico para los componentes beans y un método para descubrir los

servicios provistos por el bean dentro de la jerarquía. El contenedor permite la agrupación de beans de forma lógica que facilita la navegación.

La figura 5.3a muestra como un JavaBean puede ser anidado a los servicios Jini para crear un componente servicio. De esta forma podemos utilizar Jini como componente de almacenamiento y recuperación, mientras tomamos ventaja de las características de composición de JavaBeans.

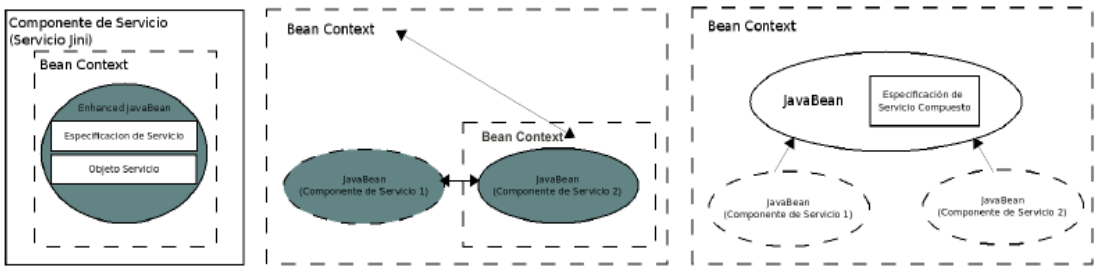


Figura 5.8: Composición Dinámica de Servicios y Jini

La figura 5.3b muestra como un BeanContext puede ser introducido de un componente de servicio en otro en tiempo de ejecución para crear un componente de servicio independiente(Stand-Alone Composite Service). La figura 5.3c Muestra que un servicio compuesto independiente mas reutilizable puede ser creado donde todo el código es contenido dentro de un único JavaBean.

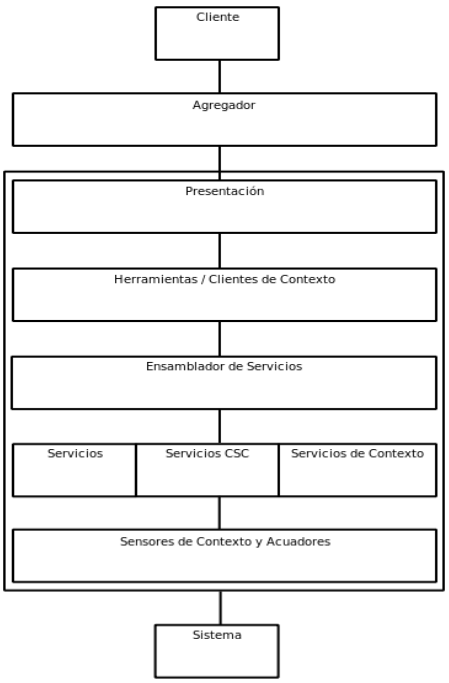


Figura 5.9: Arquitectura de Sakai Distribuido

En conclusión, si deseáramos incluir estas características a Sakai, su Arquitectura se vería afectada por la introducción de una nueva capa denominada *Ensamblador de Servicios* en su sistema estratificado, como se puede apreciar en la figura 5.4. Esta capa implementaría todas las características propuestas en [33] con el fin de brindar a Sakai Métodos de composición de Servicios y conexión entre componentes distribuidos.

CAPÍTULO 6

Conclusiones y Trabajos Futuros

6.1. Conclusiones

A lo largo de más de medio año de trabajo se desarrollo parte de la documentación de un sistema colaborativo (con adaptaciones para ambientes e-learning) que facilita en gran medida la comprensión de la estructura del sistema. Este trabajo es un aporte significativo a la documentación ya existente. A través de este aporte, se ven reflejados aspectos sobre el comportamiento, estructura y desarrollo del sistema colaborativo Sakai. A través del análisis de los diseños primitivos y las propuestas de modificación, se establecieron conceptos e información necesaria para tener en cuenta en las interpretaciones de diseño y desarrollo de herramientas, servicios y la tecnología que los envuelve. En este sentido, las propuestas arquitectónicas y vistas documentadas resueltas son producciones originales para la comunidad Sakai.

Se propuso una arquitectura acorde para la inclusión de contratos sensibles al contexto en framework web colaborativos sensibles al contexto. De esta forma pueden desarrollarse diferentes implementaciones de integración de CSC en un entorno colaborativos tipo e-learning; orientada hacia adaptación dinámica del sistema extendiendo, personalizando y mejorando los servicios sin la necesidad de recompilar y/o reiniciar el sistema. Cabe destacar que los procesos de compilación y puesta en servicio de Sakai son efectuados una sola vez, cualquier tipos de cambios y configuración serán impuestas a través los contratos sensibles al contexto.

Partiendo de la problemática que se aborda en este trabajo y teniendo en cuenta el tipo de solución planteado, queda reflejada la necesidad de completar la documentación relacionada con los aspectos arquitectónicos.

Es posible concluir que desde la visión de la documentación arquitectónica de un sistema heterogéneo como el FWCsc compuestos por diferentes subsistemas, se logran representaciones útiles sobre las conexiones iter-subsistemas. Promoviendo un tipo de representación que permitan resaltar aspectos orientados a la integración y/o agregación de componentes.

6.2. Trabajos Futuros

La siguiente lista presenta algunas de las posibles contribuciones que pueden agregarse a esta tesina, como posibles trabajos futuros:

- Seria interesante poder modelar el framework Sakai utilizando los conceptos de desarrollo de software dirigidos por modelos, mencionados en [20]. Así como también realizar la transformación de modelo a modelo del entorno e-learning Sakai.
- Podría extenderse la documentación desarrollada, incluyendo nuevas vistas, patrones utilizados, creación de objetos, deformaciones y especializaciones comunes. También podría incluirse algún punto de documentación que mejore la inclusión de CSC a la arquitectura.

CAPÍTULO 7

Apéndices

A.1. Codigo: Herramienta Sakai MVC utilizando JSF.

TasklistTool:

```
package org.sakaiproject.tool.tasklist;

import javax.servlet.*;
import java.io.*;;

public class TasklistTool extends HttpServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
            res.setContentType("text/html; charset=UTF-8");
            PrintWriter out = res.getWriter();

            out.println("<h1>Sakai Tasktool</h1>");
            out.println("Hello, "+ req.getRemoteAddr());
        }
}
```

TasklistController.Java:

```
package org.sakaiproject.tool.tasklist;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.*;
import org.sakaiproject.tool.tasklist.api.TaskListService;
import org.sakaiproject.tool.api.Tool;
import org.sakaiproject.component.cover.ComponentManager;

public class TaskListController extends HttpServlet {

        private static final long serialVersionUID = 1L;
        private TaskListService taskListService;
        private static Log log = LogFactory.getLog("TaskListController");

        public void init() {
            log.info("TaskListController init()");
            ComponentManager componentMgr = ComponentManager.getInstance();
            taskListService = (TaskListService)
              componentMgr.get("org.sakaiproject.tool.tasklist.api.TaskListService"); }
        public void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException
          {
             finish(req,res);
          }
        public void doPost(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException
          {
             String taskText = (String)req.getAttribute("taskText");
             taskListService.addTask(taskListService.createTask(taskText));
             finish(req, res);
          }
        private void finish(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException
        {
            req.setAttribute("tasks", taskListService.getAllTasks());
            req.setAttribute("username", taskListService.getCurrentUserDisplayName());
            req.setAttribute(Tool.NATIVE_URL, Tool.NATIVE_URL);
            // dispatch to the target
            String target = "/tasklist/TaskList.jsp";
            RequestDispatcher dispatcher =
                getServletContext().getRequestDispatcher(target);
            dispatcher.forward(req, res); }
}
```

Tasklist.jsp:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Task List Tool</title>
        <link href="/library/skin/tool_base.css" type="text/css"
              rel="stylesheet" media="all" />
        <link href="/library/skin/default/tool.css" type="text/css"
              rel="stylesheet" media="all" />
        <link href='/sakai-tasklist-tool/css/TaskList.css'
              rel='stylesheet' type='text/css' />
        <script type="text/javascript" language="JavaScript"
              src="/library/js/headscripts.js"></script>
</head>
<body>
<div class="portletBody">
<form id="tasks" name="tasks" method="post"
  action="http://localhost:8080/mercury/sakai.tasklist/mercury/TaskList"
  enctype="application/x-www-form-urlencoded">

        <h3 class="insColor insBak insBorder">Task List</h3>
        Hello, <%= request.getAttribute("username") %>
        <table border="0" cellspacing="0" class="chefEditItem">
                <tbody>
                  <tr>
                    <td class="chefLabel">
                          <input id="task_text" type="text" value="" size="60"/>
                    </td>
                        <td class="chefValue">
                          <input id="add_task_button" type="submit"
                            value="Add Task" onclick="clear_tasks();" />
                    </td></tr></tbody></table>

        <table id="tasks:tasklist" class="listHier">
                <thead>
                  <tr>
                    <th class="firstHeader"></th>
                    <th class="secondtHeader">ID</th>
                    <th class="thirdHeader">Owner</th>
                    <th class="fourthHeader">Task</th></tr></thead>

                <tbody id="tasks:tasklist:tbody_element">
                  <tr>
                    <td class="firstColumn">
                      <input type="checkbox"
```

```
                        name="tasks:tasklist_0:taskSelect"
                        id="tasks:tasklist_0:taskSelect"
                        value="true" /></td>
                    <td class="secondColumn">1</td>
                    <td class="thirdColumn">admin</td>
                    <td class="fourthColumn">Strip out Hibernate and JSF.</td></tr>

                  <tr>
                    <td class="firstColumn">
                      <input type="checkbox"
                        name="tasks:tasklist_1:taskSelect"
                        id="tasks:tasklist_1:taskSelect"
                        value="true" /></td>
                    <td class="secondColumn">2</td>
                    <td class="thirdColumn">admin</td>
                    <td class="fourthColumn">Draw some nice diagrams.</td>
                  </tr>
                </tbody>
            </table>

            <p class="act">
              <input id="tasks:deleteTask" name="tasks:deleteTask" type="submit"
                value="Delete" onclick="clear_tasks();" /></p>


            <input type="hidden" name="tasks_SUBMIT" value="1" />
            <input type="hidden" name="tasks:_link_hidden_" />

            <script type="text/javascript">
            <!--
            function clear_tasks() {
              var f = document.forms['tasks'];
              f.elements['tasks:_link_hidden_'].value='';
              f.target='';
            }
            clear_tasks();
            //-->
            </script>
    </form></div></body></html>
```

Task:

```
package org.sakaiproject.tool.tasklist.api;
import java.util.Date;

public interface Task extends java.io.Serializable {

        public Long getId();
        public void setId(Long id);
        public String getOwner();
        public void setOwner(String owner);
        public String getWorkSiteId();
        public void setWorkSiteId(String workSiteId);
        public Date getCreationDate();
        public void setCreationDate(Date creationDate);
        public String getTask();
        public void setTask(String task);
}
```

TaskListService:

```
package org.sakaiproject.tool.tasklist.api;
import java.util.Collection;
import org.sakaiproject.tool.tasklist.api.Task;

public interface TaskListService {

        public Collection getAllTasks(String workSiteId);
        public Collection getAllTasks();
        public void addTask(Task task);
        public void removeTask(Long id);
        public String getWorkSiteId();
        public String getCurrentUserId();
        public String getCurrentUserDisplayName();
        public Task createTask(String taskText);
}
```

TaskListServiceImpl.java:

```
package org.sakaiproject.tool.tasklist.impl;

import java.util.*;
import org.apache.commons.logging.*;
import org.sakaiproject.user.api.UserDirectoryService;
import org.sakaiproject.tool.*;

public class TaskListServiceImpl implements TaskListService {

        private Log log = LogFactory.getLog(this.getClass());
        private TaskListManager taskListManager;
        private ToolManager toolManager;
        private UserDirectoryService userDirectoryService;

        public TaskListServiceImpl()
        {
                log.info("Constructor");
        }

        public void setTaskListManager(TaskListManager taskListManager)
        {
                log.info("setTaskListManager");
                this.taskListManager = taskListManager;
        }

        public void setToolManager(ToolManager toolManager)
        {
                log.info("setToolManager");
                this.toolManager = toolManager;
        }

        public void setUserDirectoryService(UserDirectoryService userDirectoryService)
        {
                log.info("setUserDirectoryService");
                this.userDirectoryService = userDirectoryService;
        }

        public Collection getAllTasks(String workSiteId)
        {
                log.info("getAllTasks");
                return taskListManager.findAllTasks(workSiteId);
        }

        public Collection getAllTasks()
        {
```

```
                log.info("getAllTasks");
                return this.getAllTasks(this.getWorkSiteId());
        }

        public void addTask(Task task)
        {
                log.info("addTask");
                taskListManager.saveTask(task);
        }

        public void removeTask(Long id)
        {
                log.info("removeTask");
                taskListManager.deleteTask(id);
        }

        public String getWorkSiteId()
        {
               log.info("getWorkSiteId");
               String workSiteId = toolManager.getCurrentPlacement().getContext();
               log.info("getWorkSiteId = " + workSiteId);
               return workSiteId;
        }

        public String getCurrentUserId()
        {
                log.info("getCurrentUserId");
                return userDirectoryService.getCurrentUser().getEid();
        }

        public String getCurrentUserDisplayName()
        {
                log.info("getCurrentUserDisplayName");
                return userDirectoryService.getCurrentUser().getDisplayName();
        }

        public Task createTask(String taskText)
        {
                Task task = new TaskImpl();
                task.setOwner(this.getCurrentUserId());
                task.setWorkSiteId(this.getWorkSiteId());
                task.setCreationDate(new Date());
                return task;
        }
}
```

Bibliografía

[1] A. Sartorio, M. Cristiá: First Approximation to DHD Design and Implementation. Clei electronic journal, Vol.12 N. 1. (2009).

[2] UWA Consortium.: Ubiquitous web applications. Proceedings of The eBusiness and eWork Conference (e2002), 16–18 October, Prague, Czech Republic.(2002)

[3] G. D. Abowd, Classroom 2000: an experiment with the instrumentation of a living educational environment. IBM System Journal 38, 4 (1999), 508-530.

[4] N. Oliver, Towards Perceptual Intelligence: Statistical Modeling of Human Individual and Interactive Behaviors. PhD thesis, MIT Media Lab, 2000.

[5] A. Sartorio Los contratos context-aware en aplicaciones para educación e investigación. En P. San Martín, A. Sartorio, G. Guarnieri, G. Rodriguez: Hacia un dispositivo hipermedial dinámico. Educación e Investigación para el campo audiovisual interactivo. Universidad Nacional de Quilmes (UNQ). ISBN: 978-987-558-134-0. (2008)

[6] B. Meyer:Applying Design by Contract, IEEE Computer, 40-51. (1992)

[7] A. Sartorio, M. Cristiá: Primera aproximación al diseño e implementación de los DHD. XXXIV Congreso Latinoamericano de Informática, CLEI, (2008).

[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley (1995)

[9] A. Davy, B. Jennings: Coordinating Adaptation of Composite Services. Proceedings of the Second International Workshop on Coordination and Adaptation Techniques for Software Entities. WCAT'05 Glasgow , Scotland (2005)

[10] http://caeti.uai.edu.ar/04/03/14/886.asp

[11] P. Dourish: What we talk about when we talk about context. Personal and Ubiquitous Computing, vol. 8, N° 1, Roma, 2004, pp. 19-30, disponible en http://www.springerlink.com/content/y8h8l9me8yabycl3/

[12] A. Sartorio. (2011)Contratos sensibles al contexto para el Dispositivo Hipermedial Dinámico.

[13] J.Gouveia, G.Koutsoukos, L.Andrade J.L.Fiadeiro. Tool Support for Coordination-Based Software Evolution

[14] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford. Documenting Software Architecture, *Views and Beyond* Addison-Wesley

[15] M. Cristía. Diseño de Software (2008). http://www.fceia.unr.edu.ar/asist/

[16] M. Cristía. Catálogo Incompleto de Estilos Arquitectónicos (2006). http://www.fceia.unr.edu.ar/asist/

[17] G. Rodriguez, La teoría de los sistemas complejos aplicada al modelado del Dispositivo Hipermedial Dinámico. Tesis doctoral. UNR (2010).

[18] L.F. Andrade, J.L. Fiadeiro, J. Gouveia, A. Lopes y M. Wermelinger. Patterns for Coordination.

[19] L.F. Andrade y J.L.Fiadeiro. Architecture Based Evolution of Software Systems.

[20] C. Pons, R. Giandini, G. Pérez, Desarrollo de software dirigidos por modelos (Conceptos teóricos y su aplicación práctica).

[21] D.L. Parnas, P.C. Clements, and D.M. Weiss, "The modular structure of complex systemsïn ICSE '84: Proceeding of the 7th international conference on software engineering. Piscataway NJ, USA: IEEE Press, 1984, pp.408-417.

[22] L. Bass, P. Clements, and R. Kazman, "Software architecture in practice"(Second Edition). Boston: Pearson Education, 2003.

[23] J.E. Bardram, Design, implementation, and evaluation of the Java Context Awareness Framework(JCAF). Department of computer science, University of Aarhus, 2005.

[24] J.E. Bardram, The Java Context Awareness Framework(JCAF)- A Service Infrastructure and Programming Framework for Context-Aware Applications. Department of computer science, University of Aarhus.

[25] G.D. Abowd, Software engineering issues for ubiquitous computing. *In proceedings of de 21st international conference on software engineering*, paginas 75-84. IEEE Computer Society Press, 1990.

[26] J.E. Bardram, Application of Context-Aware Computing in Hospital Work - Examples and design Principles. *In proceedings of the 2004 ACM symposium on Applied Computing, paginas 1574-1579. ACM Press, 2004.*

[27] L. Capra, W. Emmerich, C. Mascolo. CARISMA: Context-Aware Reflective Middleware System for Mobile Application. IEEE *Transactions on software Engineering*, October 2003.

[28] A. Dey, G.D. Abowd, D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 2001.

[29] K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. *In Proc. PerCom'04*.IEEE, 2004.

[30] K. Henricksen, J.Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In Mahmoud Naghshimeh and Friedemann Mattern, editors, *Proceedings of Pervasive 2002: Pervasive Computing : First International Conference, volume 2414 of Lecture Notes in Computer Science*, paginas 167-180, Zrich, Switzerland, August 2002.

[31] J. Hightower, B. Brumitt, and G. Borriello. The location stack: A layered model for location in ubiquitous computing. *In Proceedings of the Fourth IEEE Workshop on mobile Computing Systems and Applications (WMCSA'02)*. IEEE Computer Society Press, 2002.

[32] F. Bachmann, L. Bass, J. Carriere, P.C. Clements, D. Garlan, "Software Architecture Documentation in Practice: Documenting Architectural Layers"(2000). Computer Science Department. Paper 692.

[33] D. Mennie, B. Pagurek, An Architecture to Support Dynamic Composition of Service Components

[34] L. Cable, Sun Microsystems, "JiniTM Technology Starter Kit Documentation Package", Version 1.0.1, sNovember, 1999.

[35] M. Cristiá, "Catalogo incompleto de Estilos Arquitectónicos", 2006.

[36] D. L. Parnas, "On the criteria to be used in descomposing systems into modules," Communications of the ACM, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.

[37] E. Bascón Pantoja, El patrón de diseño Modelo-Vista-Controlador y su implementación en java Swing.

[38] S. Burbeck, Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC).

[39] J. Deacon. Model-View-Controller (MVC) Architecture.

[40] F. Bachmann, L. Bass, J. Carriere, P. Clements, D. Garlan, J. Ivers, R. Nord, R. Little, "Software Architecture Documentation in Practice: Documenting Architectural Layers", 2000.

[41] G. Rodriguez, La teoría de los sistemas complejos aplicada al modelado del Dispositivo Hipermedial Dinámico. Tesis doctoral. UNR (2010).

[42] G. Rodriguez, A. Sartorio, P. San Martín, SEPI: una herramienta para el Seguimiento y Evaluación de Procesos Interactivos del DHD. XV Congreso Argentino de Ciencias de la Computación. En prensa (2010).

[43] D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications.In Proc. of the Conference on Human Factors in Computing Systems (CHI'99), Pittsburgh, PA, USA, May 1999.

[44] A. Sartorio, G. Rodriguez, M. Vaquero, Condicionales DEVS en la coordinación de contratos sensibles al contexto para los DHD. XVI Congreso Argentino de Ciencias de la Computación. En prensa (2010).