Universidad de las Ciencias Informáticas

Facultad 1

Módulo Controlador para el Motor de Categorización Inteligente de Contenido basado en la arquitectura de Pizarra.

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autores: Eddy Fonseca Lahens
Mayra Ortíz Labrada

Tutor: Ing. Susel Vázquez Acuña
Co-Tutor: Ing. Ernesto Rodriguez Ortiz

La Habana. 13 de junio de 2012

"Año 54 de la Revolución"

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Eddy Fonseca Lahens	Mayra Ortíz Labrada
Ingeniero Informático	Ingeniero en Ciencias Informáticas
Ing. Susel Vázquez Acuña	Ing. Ernesto Rodriguez Ortiz

Agradecimientos

Eddy Fonseca Lahens

A mi mamá y mi papá, por apoyarme todo este tiempo y ser la guía por la cual encaminé mi vida.

A mi hermana, por existir y ser la mejor de todas.

A mi familia, por apoyarme siempre; especialmente a mi tía Margó por hacerse cargo de mí estos seis años lejos de casa.

A mi novia y su familia, que ahora son parte de la mía.

A Yannier y Manuel, que son como hermanos para mí.

A mis tutores, Susel y Ernesto, por revisar continuamente el trabajo y estar a mi disposición en todo momento.

A Kiuver, porque a él le debo gran parte de los conocimientos y la formación profesional que tengo.

A Yuniet y Ril, por ser mi familia en la universidad, por el tiempo que me dedicaron y la ayuda que me brindaron.

A mis compañeros de equipo en las clases y amigos fuera del aula: Aylena, Alberto, Jorge y especialmente Mayra, hoy mi compañera de tesis.

A mis viejas y nuevas amistades por todas las experiencias que hemos compartido.

Mayra Ortíz Labrada

A mi mamá y mi papá, por aceptar mis decisiones, apoyarme en todo este tiempo, estar presentes a lo largo de toda mi vida y por hacerme sentir orgullosa de ser su hija.

A mis abuelas, por el cariño que siempre me han brindado y por su preocupación en todos estos años. A mi abuelo, porque hoy estaría muy orgulloso de mí.

A mi hermanita, porque, por ella, siempre he querido ser un buen ejemplo a seguir.

A mi familia completa, porque siempre han estado al pendiente de mí; especialmente a mi tía Aidee y a mi prima Irina, por sus correos casi diarios, sus mensajes y su ayuda en todos los aspectos.

A Jesús, por quererme tanto y ayudarme en todo lo que ha estado a su alcance.

A mi novio, por el cambio que representó en mi vida, por su paciencia y su preocupación ante todos mis problemas. A los amigos que conocí a través de él, en especial a Alberto E.

A mis compañeros de equipo en las clases y amigos fuera del aula: Aylena, Alberto, Jorge y especialmente Eddy, hoy mi compañero de tesis.

A mis tutores, Susel y Ernesto, por revisar continuamente el trabajo y estar a mi disposición en todo momento.

A mi líder de proyecto, Kiuver, por sus opiniones y criterios siempre oportunos y certeros.

A los estudiantes y profesores de la facultad con los que me relacioné en las clases, en los juegos deportivos, en el proyecto, en el secretariado de la FEU y en el Comité Primario, por cada encuentro y cada actividad que hicimos juntos, esos estarán entre los mejores recuerdos de mi estancia en esta universidad.

Dedicatoria

Eddy Fonseca Lahens

A mi mamá Marlenis Lahens Espinosa y a mi papá Eddy Fonseca Hernández.

A mi hermana Elaine Fonseca Lahens.

A mi novia Susana Enamorado Estrada.

Mayra Ortíz Labrada

A mi mamá Mayra Labrada Álvarez y a mi papá Raúl Ortiz Cruz.

A mi novio José Antonio Vega Hermosilla.

Resumen

En la investigación se presenta el desarrollo del módulo Controlador para el proyecto Motor de Categorización Inteligente de Contenido (MOCIC). Este proyecto surge con el objetivo de categorizar grandes volúmenes de documentos digitales. Su arquitectura se basa en el modelo de Pizarra, que plantea la existencia de Fuentes de Conocimiento que son módulos especializados en tareas específicas, una estructura de datos global denominada pizarra y un mecanismo de control que guía todo el proceso de ejecución. El modelo estructura los sistemas de forma tal que las Fuentes de Conocimiento no se pueden comunicar entre ellas ni interactuar de ninguna otra manera que no sea mediante cambios en la pizarra. MOCIC no tenía forma de establecer comunicación entre sus módulos ni contaba con un mecanismo de control. Para resolver el problema de la comunicación se definió e implementó un protocolo que fue proporcionado a todos los módulos en forma de biblioteca de enlace dinámico. Durante la investigación se estudiaron diferentes sistemas que implementan la arquitectura de Pizarra, centrando el análisis en los mecanismos de control que implementan y se escogió uno basado en la ocurrencia de eventos predefinidos por adecuarse a las características del proyecto. La responsabilidad del módulo radica en definir e implementar el mecanismo de control seleccionado. Luego de implementado fue sometido a pruebas unitarias, funcionales, de integración y de rendimiento en busca de deficiencias, obteniendo, en cada caso, resultados satisfactorios.

Palabras clave: Arquitectura de Pizarra, pizarra, fuentes de conocimiento, mecanismo de control.

Índice general

Índice de figuras

Introducción

En los últimos años Internet se ha convertido en una biblioteca por excelencia, donde se puede encontrar información de cualquier índole. El número de usuarios de Internet aumenta considerablemente, en el 2006 el 82.0 % de la población mundial no lo usaba, en el 2011 ese por ciento se redujo al 65.0 % y se pronostica que para finales de este año, llegue a 2300 millones el número de personas que utilizarán este servicio; según estadísticas de la Unión Internacional de Telecomunicaciones (ITU) [1].

En Cuba, a raíz del incremento en el uso de las tecnologías, se ha hecho necesario encontrar una solución al problema que sugiere el gran cúmulo de contenido digital existente, tanto en la red como en los medios de almacenamiento digitales. A la Universidad de las Ciencias Informáticas (UCI) pertenece el Centro de Ideo-Informática (CIDI) y a este el Motor de Categorización Inteligente de Contenido (MOCIC), un proyecto en desarrollo, cuyo principal objetivo es brindar una solución integral para la categorización de grandes volúmenes de documentos digitales, los cuales pueden ser: documentos *office*, html, pdf, ps y texto plano.

MOCIC implementa el modelo de Pizarra como arquitectura de software. En su forma más básica, el modelo de Pizarra separa el sistema en una estructura de datos central y globalmente accesible, generalmente llamada pizarra y un conjunto de módulos o programas independientes, denominados Fuente de Conocimiento (KS)[1]. Las KSs realizan sus tareas tomando como base el contenido de la pizarra y tienen como objetivo realizar cambios en la misma [2]. En el caso del proyecto MOCIC, las KSs son los módulos con los que cuenta, en su mayoría dotados de Inteligencia Artificial (IA), los cuales de forma unida permitirán automatizar el proceso de clasificación de contenidos. Cada uno de estos módulos proporciona descriptores específicos para la clasificación de los documentos y el objetivo es que sus respuestas lleguen al módulo Decisor que es el encargado de asignar la categoría a cada documento. En los modelos de Pizarra se debe utilizar un mecanismo de control para guiar el ciclo básico de ejecución. MOCIC no cuenta con dicho mecanismo por lo que presenta las siguientes deficiencias:

[1] Del inglés knowledge source. Se ha decidido hacer uso de esta terminología debido a su uso universal en la literatura sobre el tema.

1. No existe comunicación entre los distintos módulos. Los módulos están especializados en una tarea específica, pero para realizar la misma, en ocasiones, es necesario que obtengan información provista por otros módulos.
2. No existe un modulador que se encargue de supervisar el estado del proceso de Categorización, de determinar los siguientes subprocesos a ser resueltos, ni de organizar el trabajo que deben realizar los módulos.
3. No se puede realizar un balance de carga entre las distintas instancias de un módulo determinado, desaprovechando la capacidad de procesamiento de las computadoras más eficientes.
4. No se puede realizar la clasificación automática de contenido teniendo en cuenta las respuestas de todos los módulos. Al no existir la comunicación entre los distintos módulos no se puede sopesar la importancia de las subsoluciones y mezclarlas de forma automática para obtener una solución concreta.

A partir de la descripción realizada sobre las deficiencias que presenta el motor, se puede resumir la **situación problemática** de la siguiente manera: MOCIC no cuenta con un módulo de control que guíe el proceso de Categorización, según plantea la arquitectura de Pizarra. De ahí que el **problema a resolver** en la investigación sea: ¿Cómo dirigir el proceso de Categorización de contenidos del proyecto MOCIC teniendo en cuenta la arquitectura de Pizarra?

Por lo que el **objeto de estudio** de la investigación será: la arquitectura de software de Pizarra. Y el **campo de acción** en el que se enmarca será: Mecanismos de control de la arquitectura de Pizarra.

Todo lo anteriormente expuesto va encauzado a obtener como **objetivo general**: desarrollar el módulo Controlador del proyecto MOCIC para dirigir el proceso de Categorización de contenidos mediante una arquitectura de Pizarra.

Para dar cumplimiento al objetivo enunciado se efectuarán las siguientes **tareas de investigación**:

1. Estudio de la arquitectura de Pizarra para el desarrollo del módulo Controlador de MOCIC.

2. Selección de las herramientas y tecnologías a utilizar en la solución.
3. Identificación de los procesos a automatizar.
4. Definición del protocolo de comunicación de MOCIC.
5. Desarrollo de una biblioteca dinámica que permita la comunicación entre los diferentes subsistemas.
6. Desarrollo del núcleo del módulo Controlador.
7. Integración del módulo Controlador a MOCIC.
8. Realización de las pruebas unitarias, funcionales, de integración y de rendimiento al módulo.

Diseño Metodológico

Los métodos tradicionales de investigación utilizados se explican a continuación:

Métodos Teóricos: De este tipo de método se utilizó el de Análisis Histórico-Lógico ya que se hizo un estudio de los diferentes sistemas que implementan la arquitectura de Pizarra y de las técnicas que utilizan para hacerlo, centrando la atención en la manera en que realizan el control del flujo de ejecución. Otro método utilizado fue el de Análisis y Síntesis, dado que se realizó un análisis de la bibliografía utilizada para el estudio del tema, haciendo una división del fenómeno a estudiar en los componentes que lo integran para comprender cómo funciona cada uno por separado; como resultado de ello, se toman las características principales para lograr modelar un sistema que logre una integración eficaz dentro de los procesos que rigen su comportamiento.

Métodos Empíricos: El método utilizado en este caso fue la Observación, con la aplicación del mismo se puede conocer la realidad mediante la percepción directa de los objetos y fenómenos. A través de este método se pudo conocer la esencia de la problemática definida, lo que ayudó al planteamiento del problema a resolver y el objeto de estudio, lo cual influye a la hora de tener un conocimiento más detallado de lo que se quiere, lo que hace falta hacer y cómo hay que hacerlo. La justificación de la utilización de este método se encuentra en el Anexo A.

Justificación de la investigación:

La presente investigación se realiza para desarrollar el módulo Controlador de MOCIC dada la necesidad de un módulo que guíe el proceso de categorización, según se plantea en la arquitectura de Pizarra. Una vez implementado el Controlador e integrado al resto de los módulos se obtendrá un motor de categorización inteligente de contenido funcional; lo que supondrá un gran aporte teórico y de valor práctico en el desarrollo de aplicaciones como los filtros de contenido, para la clasificación de noticias, etc. Teniendo una gran relevancia social de acuerdo con las características de los lugares en los que sea instalado el motor y a los fines que se persigan con ello.

Estructura del documento

La investigación está estructurada en tres capítulos, organizando la información de la siguiente manera:

Capítulo I: **Componentes de la arquitectura de Pizarra**. Contiene los elementos teóricos que respaldan el desarrollo del trabajo partiendo de su objetivo general. Se hace una descripción detallada de los componentes que conforman la arquitectura de Pizarra haciendo principal énfasis en los mecanismos de control que se utilizan en los diferentes sistemas que implementan esta arquitectura. Recoge las herramientas y tecnologías que se utilizarán para el desarrollo del módulo Controlador, así como su justificación al momento de haber sido seleccionadas.

Capítulo II: **Características del módulo Controlador**. Contiene las características del módulo a implementar, la definición de los requisitos de software, la arquitectura y el diseño. Un elemento importante a tener en cuenta en este capítulo es la definición del protocolo para la comunicación entre todos los módulos de MOCIC.

Capítulo III: **Implementación y prueba del módulo Controlador**. Contiene la descripción del proceso de implementación del módulo, los artefactos generados durante el mismo y la validación del módulo a través de la realización de pruebas unitarias, funcionales, de integración y de rendimiento.

El documento posee otros apartados para las conclusiones y las recomendaciones de la investigación, la bibliografía y las referencias utilizadas, la lista de acrónimos y los anexos.

Capítulo 1

Componentes de la arquitectura de Pizarra

Introducción

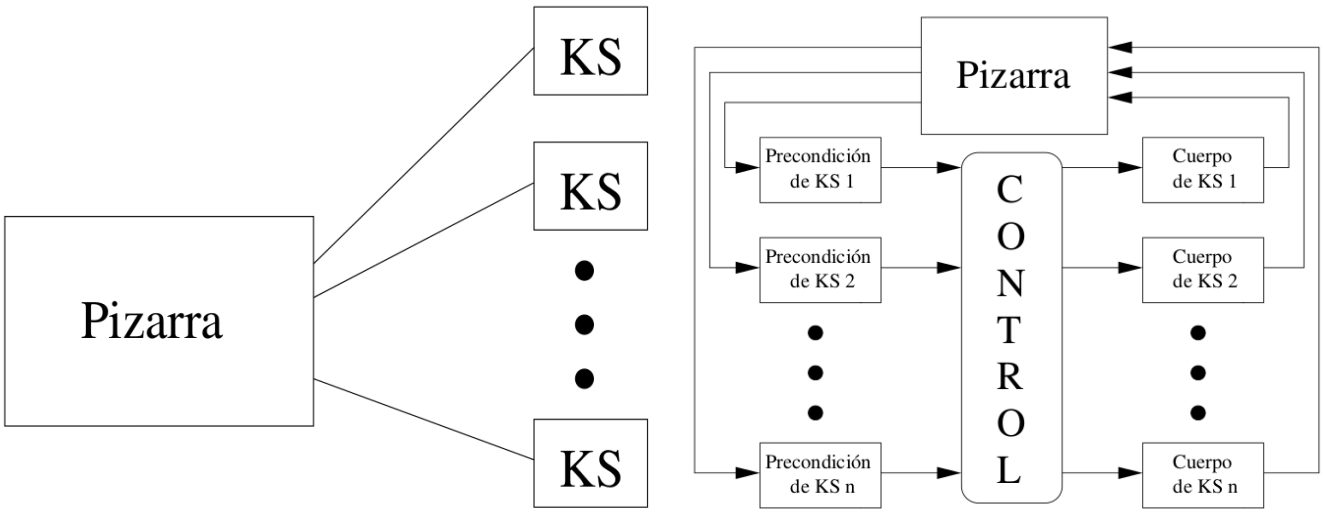
El presente capítulo contiene los elementos teóricos que respaldan el desarrollo de la investigación partiendo de su objetivo general. Se hace una conceptualización de los principales elementos que comprende de la arquitectura de Pizarra, profundizando en el componente del control y las funciones que realiza. Se describen sistemas que implementan dicha arquitectura haciendo énfasis en la manera de controlar sus ciclos de ejecución y comparándolos con el sistema Hearsay II, uno de los primeros en hacer uso del modelo de Pizarra. Una vez analizados todos los mecanismos de control implementados en estos sistemas se selecciona el más indicado para el módulo a desarrollar, basando la selección en las características y las necesidades del proyecto.

Finalmente, se identifican las herramientas y tecnologías que se utilizarán para el desarrollo del módulo Controlador, argumentando en cada caso los elementos que justifican su selección.

1.1. El modelo de Pizarra

El modelo básico de Pizarra está compuesto fundamentalmente por la pizarra y las Fuente de Conocimientos (KSs) mostrados en la Figura 1.1a. La pizarra es una estructura de datos global y usualmente está organizada de forma jerárquica, contiene las soluciones parciales, los datos de entrada y el estado del problema. Las KSs son módulos computacionales que incluyen conocimientos útiles en el contexto de la aplicación [3, 4]. Un tercer componente importante del modelo de Pizarra es el control. La Figura 1.1b añade un módulo de control al modelo básico y se muestran los dos componentes de cada KS [5]. Este modelo estructura los sistemas de forma tal que las KSs no se puedan comunicar entre sí directamente,

ni interactuar de ninguna otra forma que no sea mediante cambios en la pizarra.



a) Organización básica b) Modelo con control y partes de KS

Figura 1.1: Modelo básico de un sistema de Pizarra con control.

1.2. El entorno de Pizarra

El modelo de Pizarra solo traza algunos principios organizativos, no especifica cómo debe construirse un sistema. Para el diseño y la construcción de los sistemas se debe hacer uso de un modelo que incluya más detalles y especificaciones, denominado entorno operativo de Pizarra o entorno de Pizarra [2].

Para la implementación de las aplicaciones con arquitectura de Pizarra se utilizan diversos esquemas de razonamiento, de mecanismos de representación del conocimiento y de control. Debiéndose estas diferencias, fundamentalmente, a la naturaleza del problema que se desee resolver. Aun así, cabe destacar que las aplicaciones con este tipo de arquitectura poseen características y construcciones afines.

1.2.1. La Pizarra

En la pizarra se almacenan los resultados parciales de la solución del problema planteado. Es una estructura de datos global que se utiliza para lograr la interacción entre las diferentes KSs y las gestiones realizadas en ella son llevadas a cabo por un controlador [6, 7, 8].

En muchos sistemas, la pizarra está formada solamente por elementos del espacio de soluciones y datos de control. Esos elementos pueden ser datos de entrada, soluciones parciales, alternativas y soluciones finales. El diseño de la estructura de la pizarra refleja el plan básico sobre cómo se debe resolver el problema. Los cambios realizados en la pizarra por parte de las KSs conllevan al sistema, de forma incremental, hacia la solución del problema [5].

Si un sistema de pizarra necesita una mayor división de los datos a un nivel alto podrá contener varios paneles de pizarra, los cuales podrán tener sus propios niveles para el almacenamiento de distintos tipos de información. Tanto la fragmentación de la pizarra en paneles como la selección de los diferentes niveles de estos últimos son decisiones tomadas sobre la base del diseño propuesto.

1.2.2. Las Fuentes de Conocimiento

El objetivo de cada KS es aportar información que contribuya a encontrar la solución del problema. Dentro de cada una de ellas se encuentra una parte del conocimiento del dominio indispensable para resolver el problema planteado. Cada KS toma una parte de la información que está almacenada en la pizarra en el momento que accede y la actualiza de acuerdo con el conocimiento especializado que posee [9].

Las KSs son muy flexibles, presentándose de manera simple o con muy alta estructuración. La idea original de llamarles “fuentes de conocimientos” provino de que cada una de ellas contribuyera con un elemento de conocimiento a la resolución del problema, dividiendo toda la información y los procedimientos útiles para el sistema en módulos individuales e independientes. Otra característica importante y práctica que cubría esta división del conocimiento era que varias personas podían trabajar de forma simultánea en los sistemas.

Las KSs están formadas por dos partes: precondiciones y cuerpo. Las precondiciones comprueban el estado de la pizarra e indican cuándo la KS puede ejecutarse y el cuerpo es la parte ejecutable.

1.2.3. El Control

El objetivo del control en estos modelos es que las KSs respondan de manera oportunista a los cambios realizados en la pizarra. Generalmente se encuentra conformado por un grupo de módulos que se encargan de registrar los cambios ocurridos en la pizarra y un planificador que se encarga de decidir cuáles serán las siguientes acciones a ejecutarse. La información utilizada por estos módulos es suministrada por el sistema y puede encontrarse recogida dentro de la pizarra o de forma separada.

El ciclo de control de un sistema basado en el modelo de Pizarra, es el siguiente:

1. Determinar qué KSs están activadas.
2. Seleccionar, de las KSs activadas, las que serán ejecutadas.
3. Ejecutar las KSs seleccionadas. Esta ejecución causará cambios en el estado de la pizarra, lo que activará otras KSs.
4. Volver al paso 1.

Para satisfacer el paso 2 del ciclo de control es necesario un mecanismo para elegir la KS a ejecutar de entre todas las activadas, o bien ordenar todas las KSs activadas para su ejecución en series. Este problema se conoce como el problema del control y al mecanismo utilizado para resolverlo se le denomina estrategia de control.

La selección de la estrategia más adecuada de aplicación del conocimiento depende de las características de la aplicación en concreto y de la calidad y cantidad de conocimiento del dominio disponible. El comportamiento del sistema va a estar determinado por la capacidad de elegir un panel de la pizarra y una KS específica para actuar sobre él.

La estrategia de control y los detalles relacionados con el funcionamiento del sistema varían en gran medida de sistema a sistema. A continuación se indican algunas de las consideraciones que se deben tener en cuenta para realizar el control de un entorno de Pizarra y algunas variaciones importantes entre diferentes sistemas en dichos aspectos.

Planificación de precondiciones

Generalmente las precondiciones de todas las KSs se evalúan a la vez, pero en determinados sistemas, las propias precondiciones son acciones posibles de planificar. Para cada ciclo de control, el planificador elige una precondición para evaluar o bien el cuerpo de una KS para ejecutar. Solo los cuerpos de aquellas KSs cuyas precondiciones estén evaluadas positivamente serán candidatos a ejecutarse. Todas las precondiciones de las KSs podrían ser candidatas a evaluarse, o bien se podría considerar solamente un subconjunto de ellas, definiendo dicho conjunto mediante otras condiciones, es decir, precondiciones de las precondiciones.

Duración del ciclo de control

Las KSs se ejecutan hasta su finalización por lo que la duración de cada ciclo de control difiere de acuerdo al tiempo que demore evaluar las precondiciones y ejecutarse la KS. Es muy difícil tratar de limitar el tiempo total de ejecución de una KS dado que estas pueden ser cualquier tipo de elemento. En sistemas de resolución de problemas es conveniente hacerlo, siempre que todos los módulos estén diseñados para ejecutarse lo necesariamente rápido, de manera que el tiempo total que se precise para hallar una solución sea adecuado para la generalidad de las entradas.

Consideraciones de finalización

Un sistema de resolución de problemas sin entradas continuas[1] termina de ejecutarse cuando encuentra una solución válida o cuando considera que no la puede hallar. La finalización del sistema puede estar

[1] Sistemas que reciben una entrada inicial y a continuación producen una solución

condicionada por una KS especial que incluya en sus precondiciones la evaluación de la eficacia de la solución encontrada. Otra manera de concluir sería que terminado un ciclo de control no se encuentre ninguna KS activa. Si un sistema no se encuentra diseñado correctamente es posible que se entre en un ciclo infinito, aunque es raro encontrarlo en la práctica.

Las consideraciones de finalización anteriormente explicadas no se aplican a sistemas embebidos y dinámicos porque estos no las necesitan. Los eventos externos son los encargados de realizar cambios en las entradas que se transforman en cambios en la pizarra, en este tipo de sistemas. O sea, si luego de un ciclo de control no se encuentra ninguna KS lista para ejecutarse, el sistema sencillamente espera hasta que algún evento externo active alguna.

1.3. Sistemas basados en la arquitectura de Pizarra

Entre los diferentes sistemas que implementan la arquitectura de Pizarra no se observan variaciones relevantes en la implementación de la pizarra o en la composición de sus KSs. Las desigualdades más significativas entre estos sistemas se ubican en la implementación del módulo de control. Por tal motivo, la descripción de cada uno de ellos se concentrará en este aspecto.

En la Figura 1.2 se muestran algunos de los primeros sistemas en implementar la arquitectura de Pizarra y las relaciones establecidas entre ellos [10]. El sistema Hearsay-II se tomará como referencia para la descripción de estos, debido a que la mayoría de los aspectos de control que aparecen en los sistemas posteriores fueron abordados de alguna manera en este sistema. Se analizarán las diferencias entre la aproximación al componente del control de cada uno de los sistemas presentados y Hearsay-II.

1.3.1. HEARSAY-II: Control basado en agenda

El sistema Hearsay-II poseía una base de datos con artículos relacionados con las Ciencias de la Computación. La entrada al mismo era a través de una consulta hablada a dicha base de datos y a partir de esta se debía ejecutar la consulta indicada, siendo el objetivo primordial la interpretación de la señal audible.

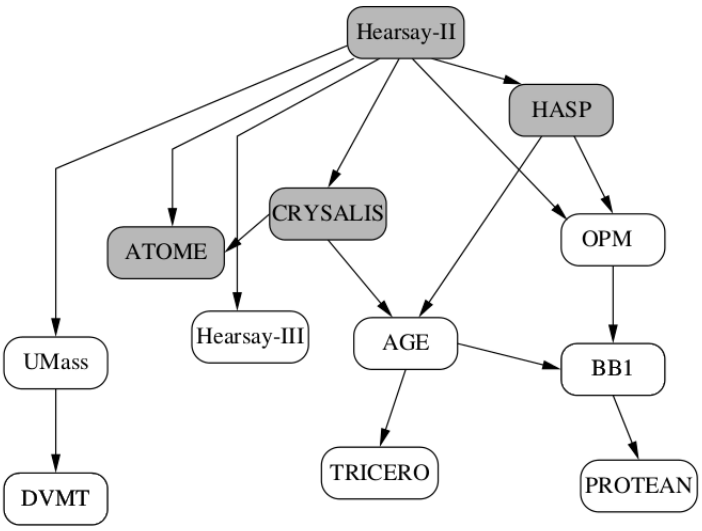


Figura 1.2: Algunos de los primeros sistemas de pizarra y sus relaciones.

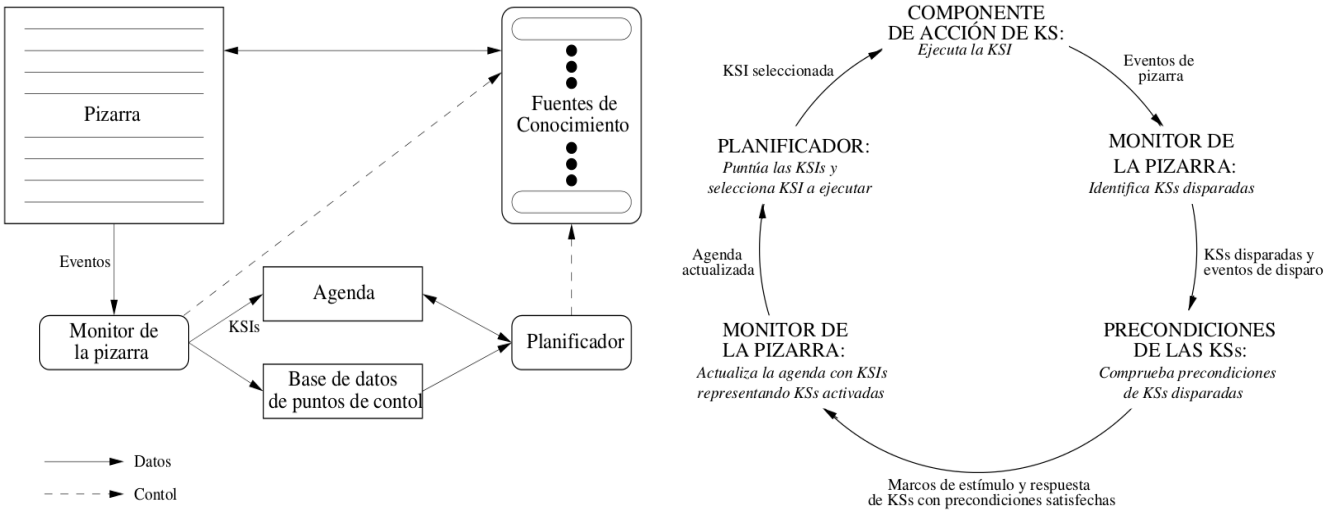
En Hearsay-II se hizo uso de un mecanismo de control basado en agenda, donde todas las acciones viables (instancias de las KSs activadas) se guardan en una agenda. En cada ciclo de control, a través de una función, se determina la importancia (o peso) de cada una de las instancias de las KSs y finalmente se selecciona la que tenga la más alta para ejecutarse.

En la Figura 1.3 se muestran los elementos principales de la arquitectura del sistema Hearsay-II y su ciclo de control. Los componentes que se relacionan con el control dentro de la arquitectura de este sistema son: el monitor de la pizarra, la agenda, el planificador y la base de datos de puntos de control [11, 12].

En una arquitectura de control basada en agenda se consideran todas las acciones posibles a realizarse en cada ciclo de control debido a la activación oportunista de las KSs lo cual es posible porque se efectúa un razonamiento guiado por datos.

La función que se utiliza para determinar la importancia de cada Fuente de Conocimiento Instanceada (KSI)[2] es lineal y cuenta con distintos factores de valores estáticos. Dichos factores son seleccionados para facilitar un razonamiento dirigido por objetivos con el que se intenta “estimar la utilidad de la acción para conseguir el objetivo global” [12]. Lo anterior se puede conseguir porque esta función de planificación

[2]Del inglés Knowledge Source Instantiation. Se ha decidido hacer uso de esta terminología debido a su uso universal en la literatura sobre el tema.



a) Arquitectura b) Ciclo de control

Figura 1.3: Hearsay II: Control basado en agenda.

no solo cuenta con los marcos de entrada y salida de cada KSI, sino que cuenta además con datos sobre el estado general de la solución del problema, conseguidos por medio de la base de datos de puntos de control.

De Hearsay-II se deben tener en cuenta las siguientes dos características.

Una es la sobrecarga producida cuando se calcula la importancia de todas las KSIs, más aún cuando se debe recalcular continuamente la de las mismas. A pesar de que lo explicado indica que todas las KSIs se calculan en cada ciclo, muchos de los sistemas basados en agenda hacen uso de algún mecanismo para restringir esta sobrecarga. En el caso de Hearsay-II las entradas de cada KSIs identifican “hipótesis claves” de tal forma que si se realizan cambios en dichas hipótesis se debe recalcular el peso de la KSI. Además, el marco de salida está vinculado con la base de datos de puntos de control, de manera que, cuando se provoquen cambios en los paneles de la pizarra sobre los que podría actuar la KSI, se debería recalcular su peso. Si ocurriera lo contrario no se debería hacer el cálculo nuevamente.

La otra característica es que en el momento de comprobar las precondiciones de alguna KS se podría producir una demora que retarde la ejecución de la KSI resultante. En el tiempo que demore ese retardo pueden producirse cambios en la pizarra y entonces el sistema debe volver a comprobar si esa KSI es

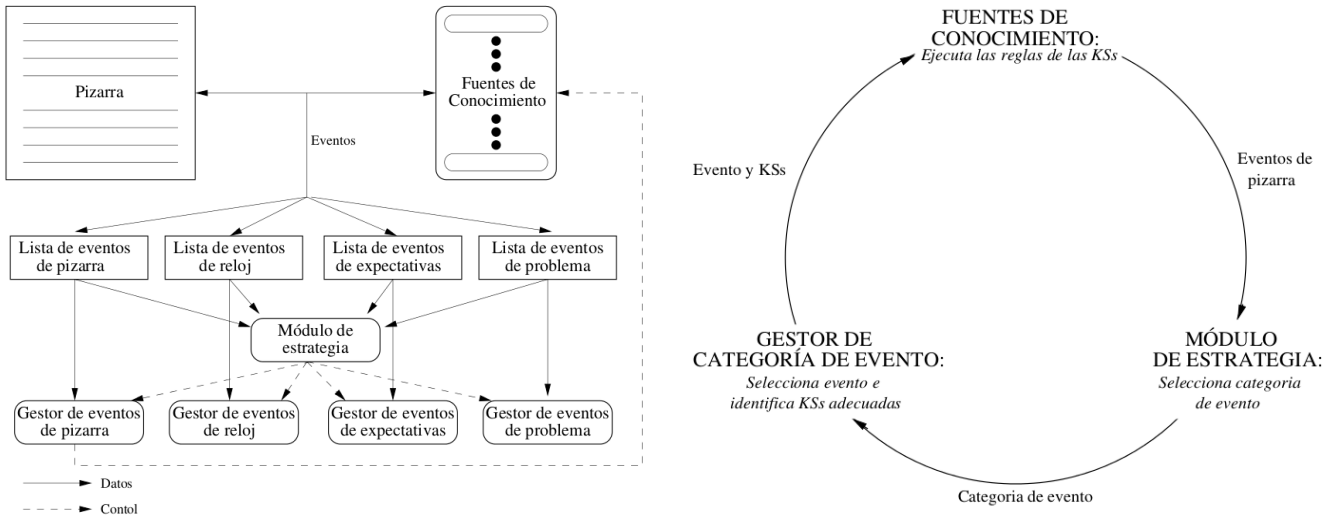
aplicable aún antes de pasar a su ejecución. Para remediar este tipo de situaciones se han presentado varios mecanismos, que van desde, sencillamente, revaluar las precondiciones después de que la KSI es seleccionada para su ejecución hasta la creación de métodos que permitan al sistema descubrir cuándo una KSI ya no es adecuada y puede ser eliminada de la agenda. El sistema Hearsay-II hace uso de etiquetas para la detección de las KSIs que pueden ser eliminadas de la agenda. Son etiquetadas determinadas estructuras de datos en la pizarra que son referenciadas desde el marco de entrada de una KSI y el sistema es el encargado de informarle de cualquier cambio que se produzca en esos datos. De acuerdo con esos cambios son revaluadas las precondiciones para determinar si la KS es aplicable todavía o si es necesario hacerle algún cambio para que pueda ser usada en un nuevo contexto.

1.3.2. HASP/SIAP: Control basado en eventos

El proyecto HASP/SIAP [13, 14] fue uno de los primeros sistemas basados en el modelo de Pizarra del Hearsay-II, estuvo diseñado para la interpretación de señales de sónar. En HASP/SIAP no se hace uso del mecanismo basado en agenda, sino de uno basado en la ocurrencia de eventos predefinidos. En Hearsay-II los cambios en la pizarra son descritos a partir de un conjunto de tipos de eventos de pizarra que se encargan de activar las KSs apropiadas, para pasar a la evaluación de sus precondiciones. En HASP/SIAP son los eventos la base principal del mecanismo de control.

Los programadores de HASP/SIAP, no hicieron uso de un conjunto primitivo de datos para indicar los cambios realizados en la pizarra, sino que definieron los cambios en la pizarra que eran realmente de interés para el sistema a través de su propio conjunto de eventos. Aparejado a esto se especificaron un conjunto de KSs que se ejecutarían de acuerdo con cada tipo de evento. La decisión más importante que debe tomar HASP/SIAP es la elección del tipo de evento como próximo punto de atención. Cuando es seleccionado el evento las KSs a ejecutarse ya están predeterminadas. O sea, con el mecanismo de control basado en eventos, estos actúan como las precondiciones de las KSs, eliminando la incertidumbre sobre cómo actuar de forma adecuada ante un evento. En la Figura 1.4 se muestra la arquitectura de HASP/SIAP y su ciclo de control básico.

En HASP/SIAP debe ser interpretado un flujo de datos de entrada continuo lo cual representa una can-



a) Arquitectura b) Ciclo de control

Figura 1.4: HASP/SIAP: Control basado en eventos.

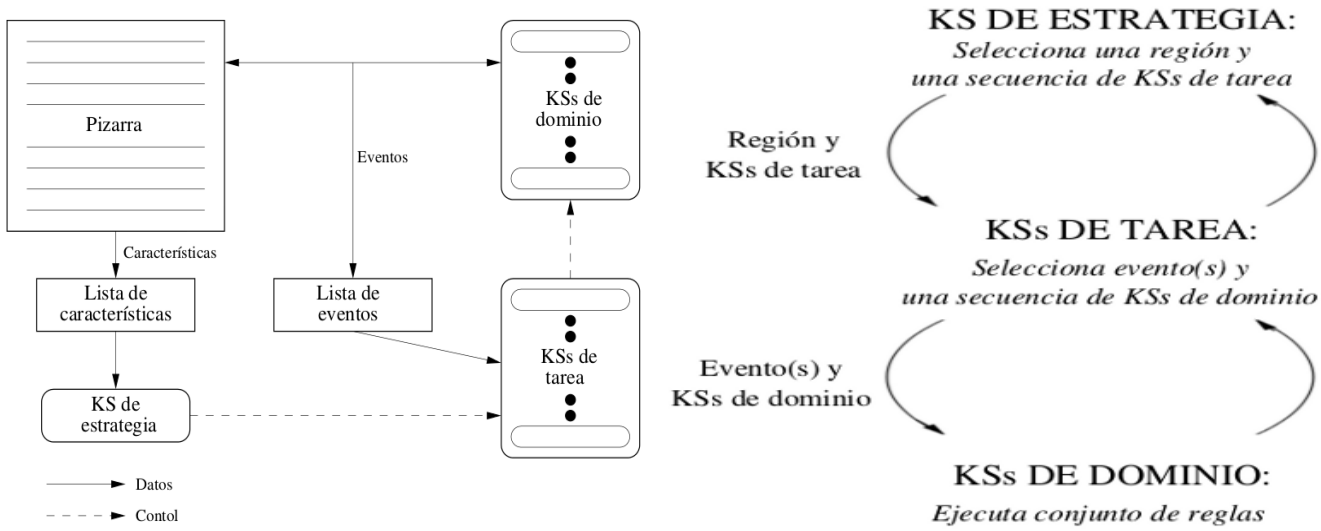
tidad considerable de datos para ser procesados. Se utilizan KSs para examinar esos datos de entrada en busca de características importantes de las hipótesis de interpretación de alto nivel.

A diferencia de lo que ocurre en un mecanismo de control basado en agenda, el arribo de un nuevo conjunto de datos no determina la aparición de un nuevo evento. En lugar de ello, se utilizan las expectativas para enfocar al sistema hacia la interpretación de una cantidad definida de los datos que van llegando. En HASP/SIAP no solo se exploran los datos en busca de evidencias positivas, sino que se hace para encontrar evidencias negativas también.

HASP/SIAP no hace uso de ningún modelo de estado de la solución del problema para tomar las decisiones. Lo cual se debe, en parte, a que no se recorren paralelamente diferentes caminos alternos hacia la solución. En HASP/SIAP no pueden representarse interpretaciones alternativas (no obstante sí puede representar atributos alternativos para las hipótesis, o sea, incertidumbre sobre esos atributos). Como consecuencia de ello, se hace más difícil realizar una revisión de las interpretaciones: el retroceso se realiza borrando las hipótesis afectadas e iniciando el análisis desde el punto de cambio.

1.3.3. CRYSALIS: Control jerárquico

El objetivo del sistema CRYSALIS [15, 16] era inferir la estructura atómica de proteínas de las cuales su composición era conocida, pero cuya conformación no lo era. Este fue el primer sistema con arquitectura de Pizarra que implantó la idea de un mecanismo de control jerárquico. En CRYSALIS se usa una jerarquía de KSs de control para elegir la KS de dominio que se ejecutaría, o sea, no se usa el mecanismo de planificación basado en agenda que utiliza Hearsay-II. En CRYSALIS se identifican dos niveles de conocimiento de control: uno de estrategia y otro de tarea. Solo existe una KS de estrategia y esta elige una secuencia de KSs de tarea para ser ejecutadas y estas últimas eligen un conjunto de KSs de dominio a ejecutar. En la Figura 1.5 se representa la arquitectura de CRYSALIS y su ciclo básico de control.



a) Arquitectura b) Ciclo de control

Figura 1.5: CRYSALIS: Control jerárquico.

Los módulos de estrategia y control presentan grandes diferencias con la explicación de KS que se ha dado puesto que no realizan ninguna acción sobre la pizarra ni son capaces de auto activarse.

Las KSs en CRYSALIS no tienen precondiciones y son seleccionadas directamente por las del nivel superior. Todas las KSs se implementan como conjuntos de reglas. En la KS de estrategia, la parte de condición de las reglas son condiciones de la lista de características (un resumen de las características claves de las hipótesis de la pizarra) y las partes de acción identifican una secuencia de KSs de tareas

a ejecutar. En las KSs de tarea la parte de condición de las reglas actúa sobre la lista de eventos y la parte de acción identifica una secuencia de KSs de dominio a ejecutar. En las KSs de dominio las partes de condición pueden examinar toda la pizarra y las partes de acción crean y modifican hipótesis en la pizarra.

Cuando se selecciona una KS de tarea, esta activa un conjunto predefinido de KSs de dominio para que sean ejecutadas, en lugar de elegirse la KS a ejecutar de entre todas las que se encuentran activadas en cada ciclo, lo cual puede conllevar a perder oportunismo, siendo este uno de los problemas que presenta el mecanismo de control jerárquico. Por otra parte, se obtiene una eficiencia mayor dado que no hay que realizar los cálculos necesarios para seleccionar la KS adecuada en cada ciclo de control, como se hace en Hearsay-II.

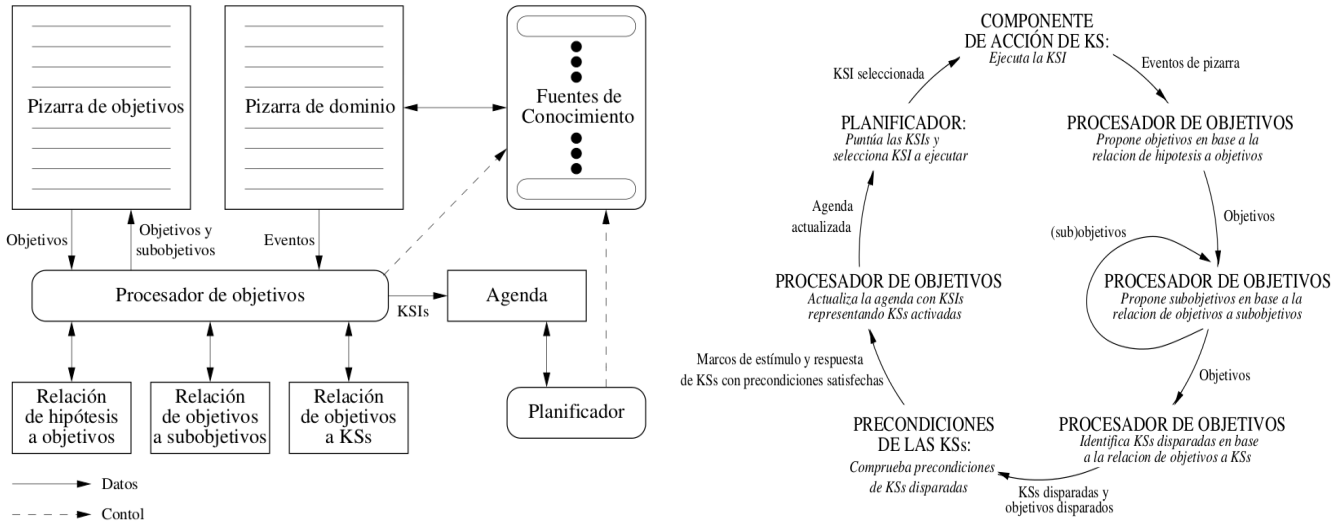
1.3.4. DVMT: Control dirigido por objetivos

Corkill [17, 18] propone la arquitectura de Pizarra dirigida por objetivos con el fin de integrar el razonamiento dirigido por objetivos con el razonamiento dirigido por datos. Las ideas de esta arquitectura se utilizaron por primera vez en el Prototipo de Monitorización de Vehículos Distribuidos (*Distributed Vehicle Monitoring Testbed – DVMT*) [19].

Este tipo de arquitectura utiliza la misma que Hearsay-II, pero le agrega una pizarra y un procesador de objetivos. Este último crea instancias de objetivos en la pizarra de objetivos y la estructura de esta es igual a la de la pizarra de dominio. El funcionamiento del procesador de objetivos está dirigido por tres funciones de relación: una relación de hipótesis a objetivos, una relación de objetivos a sub-objetivos y una relación de objetivos a KSs. En la Figura 1.6 se muestra esta arquitectura y su ciclo básico de control.

La arquitectura integra el razonamiento dirigido por datos y el razonamiento dirigido por objetivos mediante objetivos explícitos que son creados de dos maneras diferentes. El procesador de objetivos crea objetivos dirigidos por datos como respuesta a la creación o modificación de hipótesis en la pizarra de dominio, basándose en la función de relación de hipótesis a objetivos.

Estos objetivos representan la habilidad para crear hipótesis con determinadas características. Los obje-



a) Arquitectura b) Ciclo de control

Figura 1.6: DVMT: Control dirigido por objetivos.

tivos dirigidos por objetivos se crean como respuesta a la creación de otros objetivos utilizando la función de relación de objetivos a sub-objetivos. Es decir, es una forma de crear sub-objetivos. Este tipo de objetivos también se pueden crear cuando la precondición de una KS se evalúa negativamente. Si esa evaluación negativa se debe a la falta de determinadas hipótesis, la precondición puede devolver información que se utiliza para crear objetivos y para conseguir la creación de dichas hipótesis. Los objetivos dirigidos por objetivos representan el deseo de crear hipótesis con unas determinadas características con el fin de cumplir otros objetivos.

1.3.5. ATOME: Control híbrido multietapa

En el entorno de programación del sistema de Pizarra ATOME [20] la selección de KSs de dominio a ejecutar se realiza mediante un mecanismo de control derivado de Crysalis. El objetivo de los diseñadores de ATOME era aumentar la eficiencia de los sistemas con arquitectura de Pizarra sin sacrificar la flexibilidad del mecanismo de control basado en agenda. La arquitectura de control jerárquica de Crysalis puede aumentar la eficiencia en la toma de decisiones de control, pero disminuye la capacidad de reaccionar de manera oportunista cuando es necesario. ATOME extiende la arquitectura de Crysalis permitiendo re-

solver subproblemas, bien identificando directamente las KSs a ejecutar, o bien aplicando un mecanismo basado en agenda para seleccionar las KSs. La decisión sobre cuál de los dos métodos utilizar se realiza de manera dinámica, en función de qué mecanismo se considera más adecuado dado el estado actual de la solución del problema (no está claro qué factores consideran relevantes los diseñadores para realizar esta decisión). Debido a que se utiliza tanto control jerárquico como el basado en agenda y dado que la decisión sobre qué mecanismo de control utilizar se realiza para las diferentes etapas sucesivas de resolución del problema, ATOME se denomina con el término arquitectura híbrida multietapa. La arquitectura básica de ATOME es similar a la de Crysalis Figura 1.5.

Al igual que Crysalis, ATOME utiliza dos niveles de “KSs de control” y representa las KSs mediante conjuntos de reglas. La KS de estrategia selecciona las KSs de tarea a ejecutar, mientras que las KSs de tarea seleccionan las KSs de dominio a ejecutar. En la KS de estrategia, las partes de condición de las reglas son condiciones sobre resúmenes de la pizarra y las partes de acción una secuencia de KSs de tareas a ejecutar. Los resúmenes de pizarra son abstracciones del estado de la pizarra de dominio que intentan proporcionar una visión global de la calidad relativa del desarrollo de soluciones. Los resúmenes de pizarra son equivalentes a las listas de características de Crysalis y, por tanto, análogos a la base de datos de puntos de control de Hearsay-II.

Las partes de condición de las KSs de tarea son condiciones sobre los eventos del sistema y sus partes de acción identifican KSs de dominio a ejecutar. Las KSs de dominio seleccionadas, a diferencia de en Crysalis, pueden ejecutarse o bien secuencialmente o bien de manera oportunista, dependiendo del tipo de KS de tarea. Hay tres tipos de KSs de tareas: dirigidas por eventos, dirigidas por reglas, y oportunistas. En las tareas dirigidas por eventos o reglas, las KSs de dominio se identifican directamente y se ejecutan de manera secuencial. En las tareas oportunistas, el control es básicamente igual al de sistemas basados en agenda: los eventos activan KSs que esperan a que se satisfagan sus precondiciones. Una vez que esto ocurre, se introducen instancias de esas KSs en una agenda y un planificador (instanciado por la unidad de estrategia) selecciona la siguiente KSI a ejecutar.

1.3.6. Selección del mecanismo de control

En el proyecto Motor de Categorización Inteligente de Contenido (MOCIC), el proceso de Categorización de documentos constituirá un evento, este tendrá asociados un conjunto de KSs que intervendrán directamente en el mismo para conseguir el objetivo final, que es brindar una categoría al documento analizado. Se está trabajando en la inclusión de un proceso de Agrupamiento para la detección de roles, comunidades y tópicos en los correos electrónicos, este nuevo proceso constituirá otro evento y también contará con las KSs especializadas para llevar a cabo estas funciones. Tanto la Categorización de documentos como el proceso de Agrupamiento dependerán de un primer evento: el Preprocesamiento, una vez que los datos hayan sido preprocesados serán activados los eventos que sean solicitados con sus respectivos módulos.

Dadas las características expuestas y por responder adecuadamente a las necesidades del proyecto se decide utilizar un mecanismo de control basado en la ocurrencia de eventos predefinidos, similar al utilizado en el sistema HASP/SIAP (1.3.2). En el caso de MOCIC, los eventos predefinidos en los que se basará el control los constituirán los diferentes procesos con los que este cuenta (Preprocesamiento, Categorización de documentos y Agrupamiento). Con este tipo de mecanismo se elimina la incertidumbre de elegir cuáles KSs activar en cada ciclo de control, solo se tendría que identificar el tipo de evento. De acuerdo con el tipo de evento identificado se activarán los módulos específicos para su ejecución.

1.4. Metodología de desarrollo

La metodología de desarrollo de software a emplear será SCRUM *Extreme Programming* (SXP), una metodología ágil desarrollada en la Universidad de las Ciencias Informáticas (UCI) a partir de las metodologías SCRUM y *Extreme Programming* (XP). Esta plantea la generación de los artefactos indispensables para la documentación de los sistemas, centrando el interés fundamentalmente en el desarrollo de estos. Con la utilización de SCRUM para la gestión, se logra una planificación y organización inigualable; mientras que XP respalda con sus prácticas todo el proceso de desarrollo, obteniéndose de esta forma un proceso de software completo [23]. Además, esta es la metodología que utiliza MOCIC por lo que se

lograría uniformidad en la documentación asociada al proyecto.

1.5. Herramientas

Se hace un análisis de las diferentes herramientas utilizadas en la realización, tanto de la aplicación como del documento, exponiendo las características principales que llevaron a su selección.

1.5.1. Lenguaje de programación e IDE de desarrollo

Un lenguaje de programación es un conjunto de instrucciones, órdenes, comandos y reglas que permite la creación de programas. Las computadoras entienden señales eléctricas (valores 0 ó 1). Los lenguajes de programación permiten al programador indicar lo que debe hacer un programa, sin tener que escribir largas cadenas de ceros y unos, sino palabras (instrucciones) más comprensibles por las personas [25].

Para la implementación del producto final de esta investigación se utilizó el lenguaje de programación C++, a continuación se relacionan algunas de las características que llevaron a su selección:

- Es un lenguaje de programación orientado a objetos.
- Es un lenguaje muy robusto, útil para la creación de programas complejos.
- Es un lenguaje muy utilizado mundialmente, por lo que existe gran cantidad de tutoriales y ayudas en línea.
- Es el lenguaje utilizado para el desarrollo de MOCIC.

El Entorno de Desarrollo Integrado (IDE) utilizado para el desarrollo del módulo controlador es el NetBeans 7.0.1. NetBeans fue fundado como un proyecto de código abierto en el año 2000 por *Sun MicroSystems*. El IDE NetBeans constituye una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans. Es

un producto libre y gratuito sin restricciones de uso. El código fuente está disponible para su reutilización bajo las licencias: Licencia Común de Desarrollo y Distribución (CDDL) y la Licencia Pública General de GNU (GPL), ambas de código abierto y libre.[3]

1.5.2. Herramienta CASE: Visual Paradigm

La herramienta CASE que se utilizará para el modelado de los diagramas será Visual Paradigm[4], sus principales características son las siguientes:

- Soporta las últimas versiones del Lenguaje de Modelado Unificado (UML).
- Provee el modelado de procesos de negocio.
- Se integra con el IDE NetBeans.
- Proporciona el código y es compatible con el lenguaje de programación C++.
- Tiene conexión con la herramienta *Rational Rose* en sus archivos de proyecto.

1.5.3. RapidSVN

RapidSVN es un cliente gráfico de Subversion multiplataforma. Distribuido bajo la licencia GPL. Está escrito en C++ por lo que es rápido. Ha sido traducido a muchos idiomas. Proporciona una interfaz fácil de usar para las características de SubVersion.

1.5.4. Valgrind

Valgrind[5] es una herramienta libre que ayuda en la depuración de problemas de memoria y rendimiento de programas. Está compuesta por otras herramientas, entre las que se encuentra *Memcheck* que introduce

[3] Ver en http://netbeans.org/cddl-gplv2.html
[4] Ver en http://www.visual-paradigm.com
[5] Ver en http://valgrind.org

código de instrumentación en el programa a depurar, lo que le permite realizar un seguimiento del uso de la memoria y detectar los siguientes problemas:

- Uso de memoria no inicializada.
- Lectura/escritura de memoria que ha sido previamente liberada.
- Lectura/escritura fuera de los límites de bloques de memoria dinámica.
- Fugas de memoria.

1.5.5. LaTex

TeX es un programa de computadora para hacer composición tipográfica de documentos. Fue creado por D. E. Knuth. LaTex[6] es uno de los dialectos de TeX, escrito por L. B. Lamport. Entre sus características principales se encuentran:

- Útil para la escritura de libros y artículos extensos.
- Tiene facilidades para la numeración automática de capítulos, secciones, teoremas, etc.
- Es *case-sensitive*.

Como IDE para LaTex se utilizó Kile, entre sus características fundamentales están:

- Se puede compilar, convertir y ver el documento fácilmente.
- Tiene autocompletamiento de los comandos de LaTex.
- Facilita la inserción de muchas etiquetas y de símbolos estándares.
- Encontrar un capítulo o sección es muy fácil, dado que construye una lista de todos los capítulos del documento. Puede utilizar la lista para saltar a la sección correspondiente.

[6] Ver en http://www.latex-project.org/

- Ofrece una vista previa de una parte seleccionada del documento.
- Edición avanzada de comandos.

1.5.6. Doxygen

Doxygen[7] es un sistema generador de documentación para lenguajes de programación como C++, C, Java, Fortran, VHDL, PHP, C# y en cierta medida para D. Con su utilización se garantiza que la documentación generada esté organizada y sea de fácil acceso.

Conclusiones del capítulo

Luego de estudiados diferentes mecanismos de control se escogió uno basado en la ocurrencia de eventos predefinidos por adecuarse a las características y necesidades que presenta MOCIC.

El uso de la metodología SXP permitió guiar la implementación del Controlador de forma eficiente, generando los artefactos indispensables para la documentación asociada al sistema. De esta forma, se asegura la uniformidad de los documentos obtenidos con respecto a los del proyecto MOCIC.

La definición del lenguaje de programación, las tecnologías y herramientas para el desarrollo del sistema sentó las bases para iniciar su diseño y posterior implementación.

[7] Ver en http://www.doxygen.org

Capítulo 2

Características del módulo Controlador

Introducción

En el presente capítulo se recoge la propuesta del sistema a implementar y las características que este tendrá, descritas a partir de los requisitos funcionales y no funcionales acordados junto al cliente. Se representan gráficamente y a través de tablas las clases con las que contará el módulo para un mejor entendimiento de sus funcionalidades. Se define el protocolo de comunicación del Motor de Categorización Inteligente de Contenido (MOCIC) que será utilizado por todos los módulos del proyecto para el envío de información a través de la red. Se profundiza en la arquitectura y los estilos arquitectónicos seleccionados para la implementación del Controlador. Se describen la selección y utilización de patrones tanto arquitectónicos como de diseño.

2.1. Propuesta del sistema

Se propone el desarrollo de un módulo de control que sea capaz de comunicarse con el resto de los módulos del proyecto y establezca el protocolo de comunicación que se utilizará para ello. Se necesita que pueda salvar y recuperar el estado de las peticiones a procesar por cada módulo en caso de que ocurriera algún error durante la ejecución. Será configurable, de manera que se puedan realizar cambios mediante un fichero de configuración sin necesidad de hacerlos directamente en el código de la aplicación. Brindará la posibilidad de habilitar módulos de acuerdo con el tipo de evento que se vaya a ejecutar.

Es necesario que esté capacitado para que el motor funcione como un sistema distribuido, o sea, que tenga varias instancias de un mismo módulo funcionando a la vez en computadoras diferentes y en este

caso debe proporcionar un balance de carga entre las diferentes instancias de los módulos. Se le debe poder adicionar nuevas funcionalidades sin que se afecten o intervengan las que ya tuviera.

El Controlador será el encargado de definir los procesos de Preprocesamiento, Agrupamiento y Categorización de contenidos, en base a qué módulos enviará la información, en qué orden lo hará y cuáles serán las respuestas que deberá recibir de cada uno de ellos. Debe ser el encargado de supervisarlos mientras se estén ejecutando y para ello escribirá en un fichero el progreso de la ejecución, dicho fichero deberá ser mostrado a los interesados mediante el medio que defina el proyecto.

2.2. Definición del protocolo de comunicación

Para que los módulos de MOCIC puedan comunicarse se hizo necesario definir un protocolo para ello. Este protocolo utilizará una conexión por *Socket* a través de TCP/IP y será proporcionado en forma de biblioteca de enlace dinámico. La comunicación se llevará a cabo mediante el intercambio de mensajes, que contendrán parámetros específicos en dependencia del módulo que los envía y el proceso que se lleva a cabo en ese momento. En el Anexo D se encuentran las categorías analizadas por MOCIC y el identificador de cada una, estos son utilizados en las respuestas por lo que su orden y valor tiene relevancia. En la Tabla 2.1 se muestran los parámetros de los mensajes y el significado de los mismos.

Tabla 2.1: Parámetros y significado de los mensajes.

Parámetros	**Significado**
-e	Evento que se va a llevar a cabo, puede ser Preprocesamiento, Categorización o Agrupamiento.
-t	Contiene el valor del *timestamp* de cuando se comenzó a procesar el documento.
-i	Identificador del documento (sha1).
-l	Identificador único (uid) del lote.
-c	Especifica el nombre de la comunidad, si no se pasa deben ser detectadas las comunidades para el lote que se está procesando.
Continúa en la próxima página	

-u	Dirección desde donde se obtuvo el documento (URI).
-n	Nombre del módulo que envía el mensaje.
-r	Respuesta del módulo.
-a	Algoritmo utilizado en la Categorización.
-R	Especifica que deben detectarse los roles.
-T	Especifica que deben detectarse los tópicos.

Las respuestas de cada uno de los módulos varían en dependencia del contenido analizado, en el Anexo E se detallan sus estructuras.

2.2.1. Descripción detallada de los mensajes

Los procesos que se llevan a cabo en este momento son:

- Preprocesamiento (Copiar y separar los documentos por sus diferentes partes: imágenes, texto, etc.)
- Categorización (Clasificación de contenido en las categorías presentadas en el Anexo D.)
- Agrupamiento (Detección de roles, tópicos y comunidades de interés.)

Siempre se comienza con el proceso de Preprocesamiento en el cual se separan las diferentes partes de los documentos para luego pasar a uno de los dos otros procesos, Categorización o Agrupamiento.

Preprocesamiento:

1. Inicia el proceso cuando el Suministrador le envía un mensaje al Controlador, con los parámetros *-e -t -u -l -i [-c]*[1]. Si luego del Preprocesamiento se realizara un proceso de Agrupamiento y es

[1] Los parámetros entre [] son opcionales y pueden omitirse en el mensaje.

suministrado el parámetro *-c* este tendrá como valor el nombre de la comunidad y el lote constará solamente de esa comunidad por lo que no se detectarán más comunidades en dicho lote. En caso de no suministrar el parámetro *-c* se deberán detectar todas las comunidades existentes en el lote.

2. El Controlador es el encargado de comunicar los diferentes módulos y de guiar los procesos. Cuando recibe el mensaje del Suministrador especificando que el proceso que se está llevando a cabo es de Preprocesamiento, adiciona el mensaje al panel del Preprocesador.

3. El Preprocesador copia en el Depósito la carpeta con el documento preprocesado. Específicamente crea una carpeta en el Depósito, si ya no existe, que tiene como nombre el valor del parámetro *-l*. Dentro de esta carpeta que representa el lote de documentos, crea entonces una carpeta para cada documento a preprocesar y les pone por nombre el valor del parámetro *-i*. Dentro de estas carpetas copia la información del documento ya preprocesada.

4. Cuando el Preprocesador termina la copia, le envía un mensaje de respuesta al Controlador, con los parámetros *-e –i*.

5. El Controlador analiza el mensaje enviado por el Preprocesador y reenvía el mismo hacia el Suministrador.

6. Si el proceso que se llevará a cabo luego del Preprocesamiento es el de Categorización, a medida que se preprocesan los documentos y llegan las respuestas al Suministrador, este inmediatamente le envía un mensaje al Controlador con los parámetros *-l -i*, para que sea categorizado el documento. Si el proceso es de Agrupamiento y quedan documentos por preprocesar no hace nada y espera a que se preprocesen todos los documentos del lote. Si ya terminó de preprocesarlos todos, le manda un mensaje al Controlador con los parámetros *–l [–c] [-R, –T]*[2]. Si se están realizando los dos procesos sobre el mismo lote de documentos a medida que se van preprocesando los documentos se le van mandando los mensajes al Controlador para que los categorice y cuando se preprocese el lote completo entonces se mandan a detectar las comunidades, los roles y los tópicos en dependencia de lo que se desee. **Fin del proceso de Preprocesamiento.**

   Categorización:

[2]Los parámetros -R y -T son opcionales pero siempre debe aparecer al menos uno de ellos.

7. Si lo que se está procesando son documentos independientes y no por lotes, el Controlador envía un mensaje a cada uno de los módulos que se encargan de categorizar diferentes partes del contenido (Texto, Desnudez, Rostros, Objetos, Enlaces, ORC). Los parámetros de este mensaje son *-l -i*.

8. Cada uno de los módulos clasificadores utiliza los parámetros *-l -i* para obtener la ubicación del documento en el Depósito. La ruta hasta el directorio raíz del Depósito estará guardada en los ficheros de configuración de cada módulo. Utilizando esta ruta se construye la ruta hasta la ubicación del documento analizado de la siguiente forma: *ruta_hasta_el_deposito/-l/-i*

9. Luego de construir la ruta hasta la ubicación del documento, cada módulo lee del Depósito el contenido en específico que categoriza.

10. Luego de categorizar el contenido, los módulos envían la respuesta con los parámetros *-n -i -a -r*.

11. El Controlador redireccionará todas las respuestas con los parámetros *-n -i -a -r* hacia el Decisor.

12. El Decisor otorga la categoría final al documento y la salva en la base de datos .

13. Cuando el Decisor termina de categorizar y guardar la categoría del documento envía un mensaje de respuesta al Controlador con los parámetros *-n -i*, indicando qué documento fue el que se clasificó.

14. Llega al Controlador la respuesta del Decisor terminando la Categorización del documento. **Fin del proceso de Categorización.**

    Agrupamiento:

15. Si no le pasan como parámetro *–c* en el mensaje, el Controlador le pasa un mensaje a Comunidades de Interés (CI) con el parámetro *-l*. El detector de comunidades puede detectar varias en el mismo lote y se las envía todas al Controlador.

16. Si le pasan el *-c* entonces no se tiene que detectar la comunidad de interés dado que esta ya es conocida.

17. El Controlador le manda mensajes a Tópicos o Roles, en dependencia de la acción que se desee realizar, con los parámetros *-l -c*, para que se detecten los tópicos o los roles para las comunidades detectadas. Cuando estos detectan, los tópicos y los roles respectivamente, escriben en la base de

datos los resultados que obtuvieron y le mandan sus respuestas al Controlador para que este los libere.

18. Una vez que recibe los mensajes de respuesta el Controlador termina el proceso de Agrupamiento. **Fin del proceso de Agrupamiento.**

2.3. Descripción de los requisitos de la solución propuesta

El *Institute of Electrical and Electronics Engineers* (IEEE) [3] define un requisito como:

1. Una condición o capacidad que un usuario necesita para resolver un problema o lograr un objetivo.
2. Una condición o capacidad que debe tener un sistema o un componente de un sistema para satisfacer un contrato, una norma, una especificación u otro documento formal.
3. Una representación en forma de documento de una condición o capacidad como las expresadas en (1) o en (2).

Se pueden clasificar en funcionales y no funcionales y en ambos casos estos deben ser:

- Especificados por escrito. Como todo contrato o acuerdo entre dos partes.
- Posibles de probar o verificar.
- Descritos como una característica del sistema a entregar.

En el caso del módulo Controlador, los requisitos de software se presentan en la tabla de la Lista de Reserva del Producto (LRP)[4] que se encuentra en el Anexo B. En la Tabla 2.2 se muestra un resumen en el que se exponen solamente el número y la descripción del requisito, se encuentran agrupados según la prioridad que tienen en el negocio:

[3] El glosario estándar IEEE de la terminología de Ingeniería de Software identifica los términos actualmente en uso en el campo de la Ingeniería de Software y la definición estándar de los términos establecidos.

[4] Es una lista priorizada que define el trabajo que se va a realizar en el proyecto.

Tabla 2.2: Requisitos de Software

No	Descripción
Requisitos funcionales	
Prioridad: Muy Alta	
1	Comunicar los módulos de MOCIC.
2	Monitorear los procesos de Preprocesamiento, Categorización de documentos y Agrupamiento.
3	Balancear la carga entre las distintas instancias de los módulos.
Prioridad: Alta	
4	Habilitar módulos según eventos en funcionamiento.
5	Salvar estado de las peticiones a procesar si ocurre algún error durante la ejecución.
6	Recuperar el estado de las peticiones salvadas.
Prioridad: Media	
7	Generar logs de información relacionados al módulo Controlador.
8	Cargar fichero de configuración con todos los parámetros de la aplicación.
Requisitos no funcionales	
Categoría: Restricciones en el diseño y la implementación	
1	Utilizar TCP/IP para la comunicación.
2	Utilizar XML para el fichero de configuración.
3	Utilizar como lenguaje de programación C++.
4	Utilizar NetBeans como Entorno de Desarrollo Integrado (IDE).
5	Utilizar RapidSVN para el control de versiones.
6	Utilizar Valgrind para la revisión automática del código.
Categoría: Software	
7	Utilizar Ubuntu 10.04 como sistema operativo.
8	Instalar Biblioteca LibMagic.
9	Instalar Biblioteca Boost.
Categoría: Soporte	
10	Crear un instalador para la arquitectura Debian y sus derivados.

2.3.1. Historias de usuario

Las historias de usuario son una técnica utilizada para especificar los requisitos del software. Estas crean las bases para las pruebas funcionales y son utilizadas para estimar tiempos de desarrollo. En las Tablas 2.3, 2.4 y 2.5, se presentan las historias de usuario de los requisitos considerados de prioridad *Muy Alta* en el desarrollo del módulo Controlador. Las historias de usuario del resto de los requisitos se encuentran en el Anexo C.

Tabla 2.3: Historia de usuario: Con-1.

	Historia de Usuario
Número: Con-1	**Nombre de Historia de Usuario:** Comunicar los módulos de MOCIC.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Prog	**Iteración asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 3
Descripción:	El Controlador es el encargado de establecer la comunicación entre todos los módulos del proyecto a través de un protocolo de comunicación.

Tabla 2.4: Historia de usuario: Con-2.

	Historia de Usuario
Número: Con-2	**Nombre de Historia de Usuario:** Monitorear los procesos de Preprocesamiento, Categorización de documentos y Agrupamiento.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Prog	**Iteración asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Continúa en la próxima página	

Descripción:	Se monitorizarán los procesos de Preprocesamiento, Categorización de documentos y Agrupamiento para la detección de errores y se generará un fichero con el progreso de la ejecución que pueda ser visualizado por un agente externo.

Tabla 2.5: Historia de usuario: Con-3.

Historia de Usuario	
Número: Con-3	**Nombre de Historia de Usuario:** Balancear carga entre las distintas instancias de los módulos.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Prog	**Iteración asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 2
Descripción:	El motor funcionará de manera distribuida, por lo que no garantizará la igualdad de procesamiento en las distintas instancias de los módulos, por tal motivo se hace necesario realizar un balance de carga entre las diferentes instancias para lograr aprovechar al máximo los recursos brindados.

2.4. Diseño

Roger S. Pressman en [24] plantea como principios de un buen diseño, los siguientes:

1. La razón por la que todo existe.
2. Mantenerlo simple.
3. Mantener la visión.
4. Lo que uno produzca, otros lo consumirán.

5. Estar abierto al futuro.
6. Planear para la reutilización.
7. Pensar.

Teniendo en cuenta estos principios, a continuación se explican los detalles de la arquitectura propuesta, los patrones de diseño utilizados y las descripciones de las clases que se utilizarán en el desarrollo del módulo Controlador.

2.4.1. Arquitectura propuesta

Existen varios modelos arquitectónicos, entre ellos se encuentra el Centrado en datos y dentro de este se ubica el modelo de Pizarra, el cual, tal como quedó planteado en el capítulo anterior, es el que implementa el proyecto MOCIC. Un modelo de Pizarra generalmente se define y construye con tres componentes: Fuente de Conocimiento (KS), Pizarra y Control. En MOCIC este modelo está definido de la siguiente manera:

1. Fuente de Conocimiento o módulos: se corresponden con las instancias de los diferentes módulos por los que está formado el motor.
2. Estructura de datos tipo pizarra: división de la pizarra global en paneles privados para cada módulo.
3. Control: implementación de un módulo de control que determine qué KS aplicar, cuándo aplicarla y a cuál panel de la pizarra.

El módulo a implementar abarca tanto el mecanismo de control como la pizarra dentro de la arquitectura de MOCIC. En la Figura 2.1 se muestra la ubicación del módulo dentro de la misma y la relación que establece con las distintas KSs que integran el motor.

Para la implementación del mecanismo de control quedó propuesta la utilización de un mecanismo basado en la ocurrencia de eventos predefinidos. Los eventos son los encargados de activar las KSs apro-

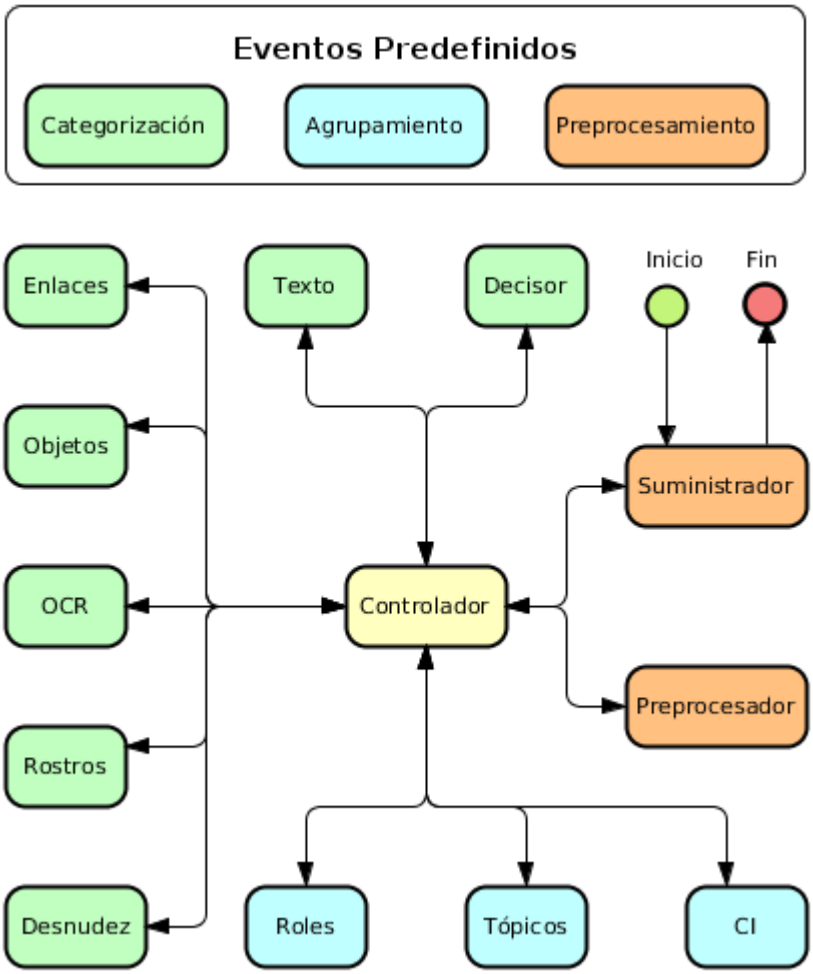


Figura 2.1: Arquitectura de software de MOCIC

piadas. Los tipos de eventos predefinidos en la solución funcionan como precondiciones de las KSs a ejecutar, eliminando así la incertidumbre sobre cómo responder de manera adecuada a un evento.

Independientemente de la arquitectura implementada por MOCIC, lo que representa y condiciona la misma en el desarrollo del módulo Controlador, la solución propuesta implementará un patrón arquitectónico basado en Plugins. Este patrón permitirá una solución extensible dinámicamente para la inclusión de nuevos eventos predefinidos. Estos eventos pueden ser implementados como componentes separados y desarrollados en paralelo por diferentes equipos.

La conexión de las KSs con el módulo Controlador se realizará a través del protocolo de comunicación de red definido en MOCIC. En la Figura 2.2 se muestra la arquitectura del módulo Controlador y la relación que se establece entre las KSs y los tipos de eventos predefinidos con los que cuenta el motor.

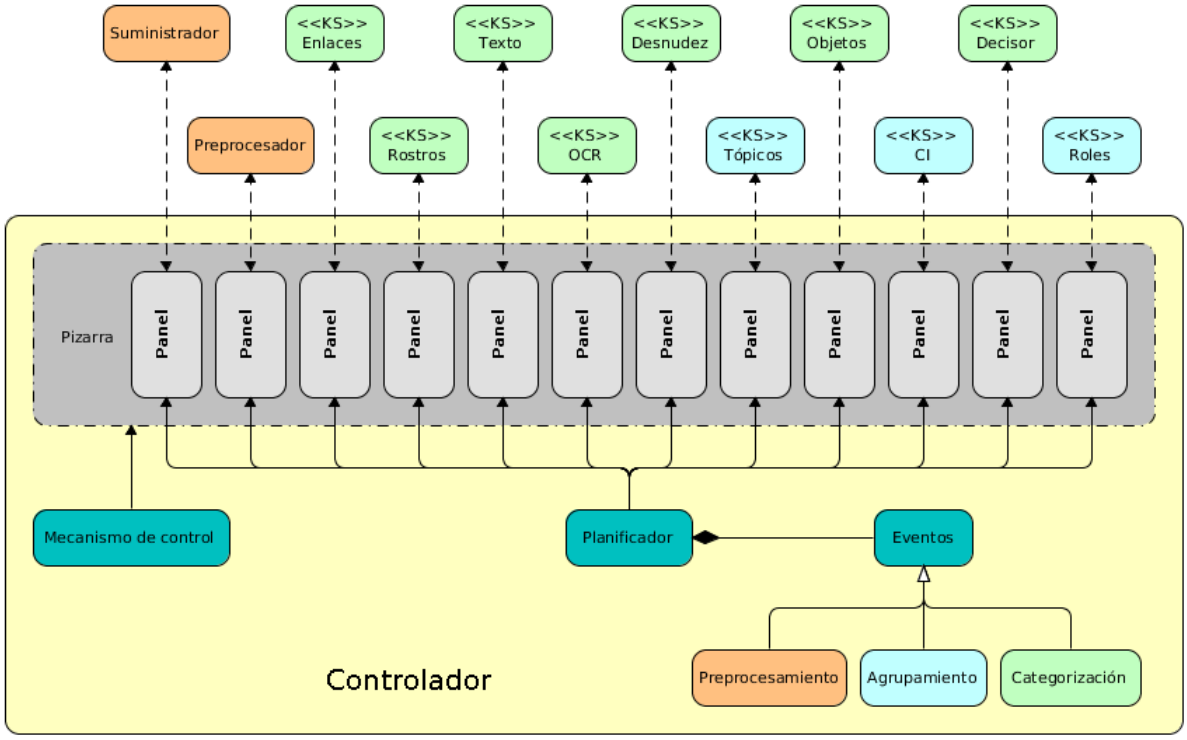


Figura 2.2: Arquitectura de software del módulo Controlador

2.4.2. Patrones de diseño

Los patrones de diseño de software constituyen un conjunto de soluciones generales a problemas similares que se presentan durante el diseño e implementación de los sistemas y se basan en la experiencia adquirida. Entre las ventajas de utilizar patrones de diseño se encuentran que: facilitan la localización de los objetos que formarán el sistema y facilitan el aprendizaje y la comunicación entre programadores y diseñadores.

Dentro de los más reconocidos se haya el conjunto *General Responsibility Asignment Software Patterns* (GRASP), que representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones; de estos se utilizaron, para el diseño del módulo Controlador, los

siguientes:

- **Experto** para asignar una responsabilidad al experto en información, dígase la clase que cuenta con la información necesaria para cumplir la responsabilidad.
- **Bajo Acoplamiento** para asignar las responsabilidades de forma tal que las clases se comuniquen con el menor número de clases que sea posible.
- **Alta Cohesión** para asignar a las clases responsabilidades que trabajen sobre una misma área de la aplicación y que no tengan mucha complejidad.

Otro conjunto de patrones muy utilizado son los Gang of Four (Gof) en español: Pandilla de los Cuatro, formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, los cuales respecto a su propósito pueden ser:

- Creacionales: Resuelven problemas relativos a la creación de objetos.
- Estructurales: Resuelven problemas relativos a la composición de objetos.
- De Comportamiento: Resuelven problemas relativos a la interacción entre objetos.

Del conjunto Gof, en el diseño del módulo Controlador se utilizó:

Template Method: Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.

2.4.3. Descripción de las clases

La descripción de las clases se realiza a través de diagramas y tablas, mostrándose a continuación.

Diagrama

Los diagramas de clases describen los tipos de objetos que existen en el sistema y las diferentes relaciones que se establecen entre ellos, además se muestran los atributos y operaciones de cada una de las clases que lo conforman [21].

En la Figura 2.3, se muestra el diagrama correspondiente a la biblioteca de enlace dinámico implementada para establecer la comunicación dentro del motor.

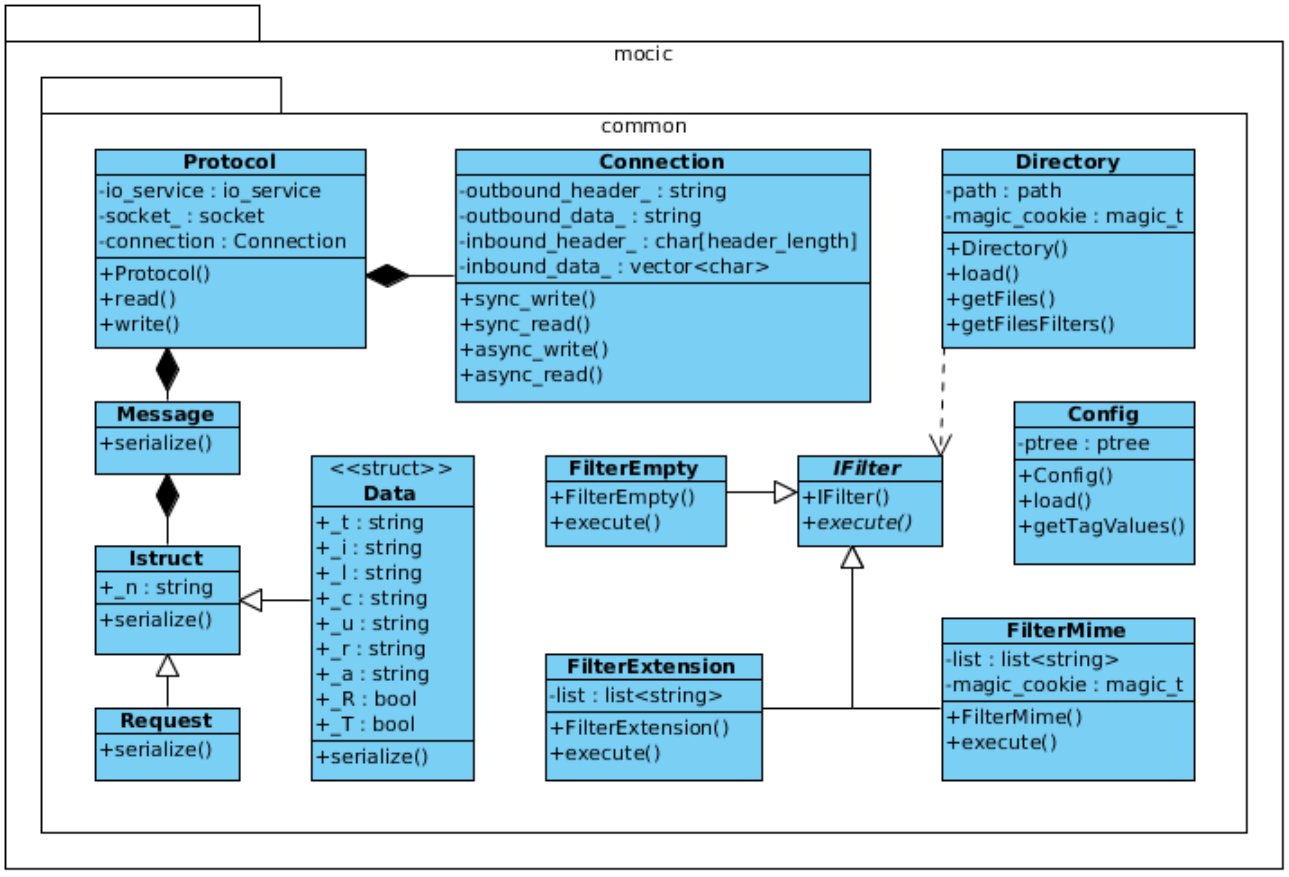


Figura 2.3: Diagrama de Clases del Paquete common

Descripción

Mediante el empleo de las tablas para la descripción de las clases se pueden conocer detalles sobre los atributos y las funcionalidades que estas contienen. A partir de la Tabla 2.6 hasta la 2.14 se presentan las descripciones de las clases más relevantes dentro del paquete common. El resto de las descripciones se pueden consultar en el Anexo F.

Tabla 2.6: Descripción de la clase Protocol.

Nombre: Protocol	
Tipo de clase: Control	
atributo:	**Tipo:**
io_service	io_service
socket_	socket
conection	Conecction
Para cada responsabilidad:	
Nombre:	Protocol()
Descripción:	Constructor de la clase.
Nombre:	read()
Descripción:	Leer una estructura de datos.
Nombre:	write()
Descripción:	Escribir una estructura de datos.

Tabla 2.7: Descripción de la clase Connection.

Nombre: Connection	
Tipo de clase: Control	
atributo:	**Tipo:**
outbound_header_	string
outbound_data_	string
inbound_header_	char[header_length]

inbound_data_	char[header_length]
Para cada responsabilidad:	
Nombre:	sync_write()
Descripción:	Escribir un mensaje de forma sincrónica.
Nombre:	sync_read()
Descripción:	Leer un mensaje de forma sincrónica.
Nombre:	async_write()
Descripción:	Escribir un mensaje de forma asincrónica.
Nombre:	async_read()
Descripción:	Leer un mensaje de forma asincrónica.

Tabla 2.8: Descripción de la clase Directory.

Nombre: Directory	
Tipo de clase: Control	
atributo:	**Tipo:**
path	path
magic_cookie	magic_t
Para cada responsabilidad:	
Nombre:	Directory()
Descripción:	Constructor de la clase.
Nombre:	load(path:path)
Descripción:	Método encargado de cargar un archivo.
Nombre:	getFiles(recursive: bool=false): list <string >
Descripción:	Método para buscar archivos en un directorio de manera recursiva.
Nombre:	getFilesFilters(filter:Filter*, recursive:bool=false): list <string >
Descripción:	Método para buscar archivos en un directorio de manera recursiva.

Tabla 2.9: Descripción de la clase Config.

Nombre: Config	
Tipo de clase: Control	
atributo:	**Tipo:**
ptree	ptree
Para cada responsabilidad:	
Nombre:	Config()
Descripción:	Constructor de la clase.
Nombre:	load(filename:string): void
Descripción:	Método para cargar un fichero.
Nombre:	getTagValues(path:path):list <string >
Descripción:	Método para obtener información de un xml.

Tabla 2.10: Descripción de la clase Filter.

Nombre: Filter	
Tipo de clase: Control	
Para cada responsabilidad:	
Nombre:	Filter()
Descripción:	Constructor de la clase.
Nombre:	execute(path:path): bool
Descripción:	Ejecuta la acción del filtro.

Tabla 2.11: Descripción de la clase Data.

Nombre: Data	
Tipo de clase: Entidad	
atributo:	**Tipo:**
Continúa en la próxima página	

date	string
filter	string
id	string
deposit	string
url	string
response	string

Tabla 2.12: Descripción de la clase Message.

Nombre: Message	
Tipo de clase: Control	
Para cada responsabilidad:	
Nombre:	serialize()
Descripción:	Construye el mensaje que se va a enviar.

Tabla 2.13: Descripción de la clase Istruct.

Nombre: istruct	
Tipo de clase: Control	
atributo:	**Tipo:**
_n	string
Para cada responsabilidad:	
Nombre:	serialize()
Descripción:	Construye el mensaje que se va a enviar.

Tabla 2.14: Descripción de la clase Request.

Nombre: Request
Tipo de clase: Control
Para cada responsabilidad:

Nombre:	serialize()
Descripción:	Construye el mensaje que se va a enviar.

En la Figura 2.4, se muestra el diagrama correspondiente al paquete controlador.

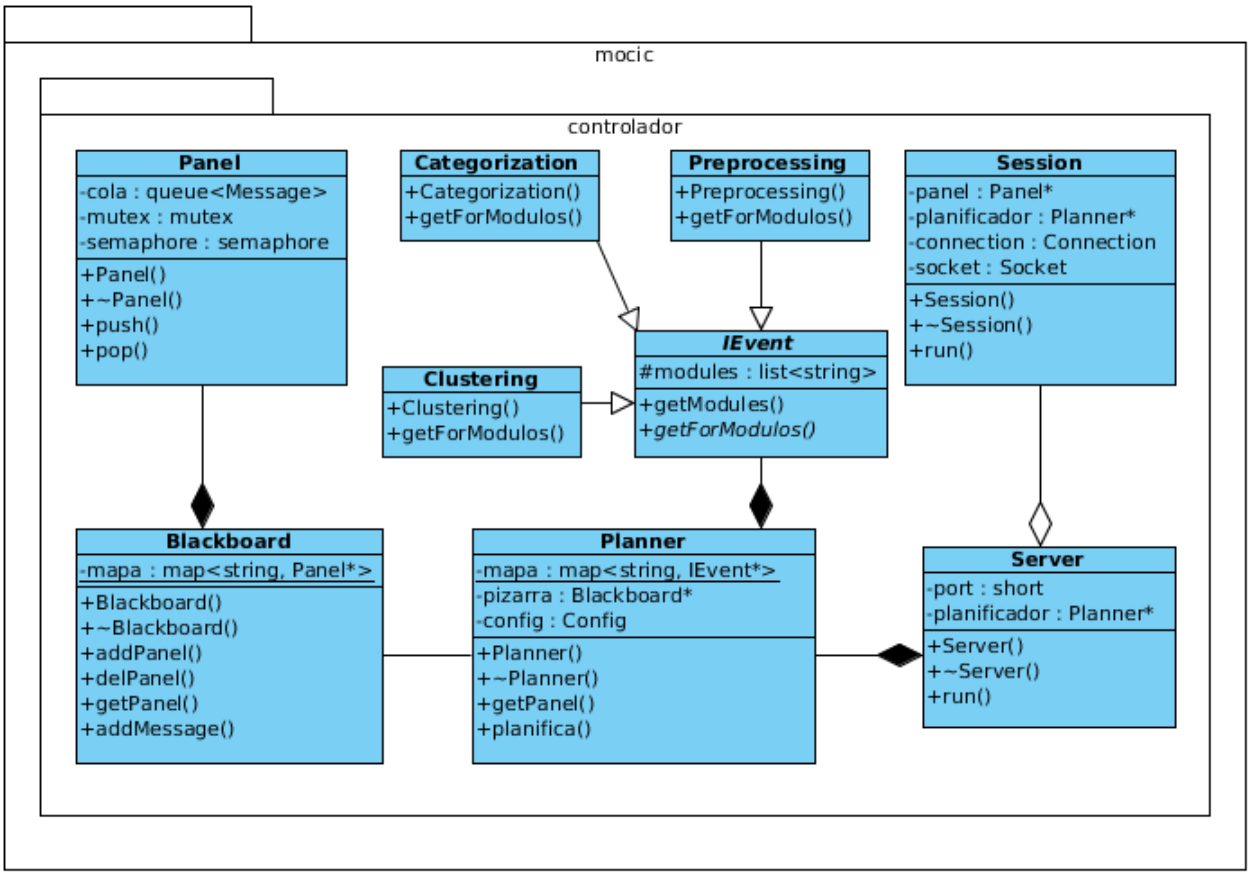


Figura 2.4: Diagrama de Clases del Paquete controlador

Descripción

A partir de la Tabla 2.15 hasta la 2.20 se muestran las descripciones de las clases más relevantes dentro del paquete controlador. El resto de las descripciones se pueden consultar en el Anexo G.

Tabla 2.15: Descripción de la clase Blackboard.

Nombre: Blackboard	
Tipo de clase: Control	
atributo:	**Tipo:**
mapa	map<string, Panel* >
Para cada responsabilidad:	
Nombre:	Blackboard()
Descripción:	Constructor de la clase.
Nombre:	Blackboard()
Descripción:	Destructor de la clase.
Nombre:	addPanel(std::string)
Descripción:	Método para adicionar un panel a la pizarra.
Nombre:	delPanel(std::string)
Descripción:	Método para eliminar un panel de la pizarra.
Nombre:	getPanel(std::string)
Descripción:	Método para acceder a un panel de la pizarra.
Nombre:	addMessage(struct Data)
Descripción:	Método para adicionar un nuevo mensaje.

Tabla 2.16: Descripción de la clase Panel.

Nombre: Panel	
Tipo de clase: Control	
atributo:	**Tipo:**
cola	queue<Message >
mutex	mutex
semaphore	interprocess_semaphore
Para cada responsabilidad:	
Continúa en la próxima página	

Nombre:	Panel()
Descripción:	Constructor de la clase.
Nombre:	Panel()
Descripción:	Destructor de la clase.
Nombre:	push()
Descripción:	Método para adicionar elementos a la cola.
Nombre:	pop()
Descripción:	Método para extraer elementos de la cola.

Tabla 2.17: Descripción de la clase Planner.

Nombre: Planner	
Tipo de clase: Control	
atributo:	**Tipo:**
mapa	map<string, IEvento* >
pizarra	Pizarra*
config	Config
Para cada responsabilidad:	
Nombre:	Planner()
Descripción:	Constructor de la clase.
Nombre:	Planner()
Descripción:	Destructor de la clase.
Nombre:	getPanel(std::string)
Descripción:	Método para obtener un panel de la pizarra.
Nombre:	planifica(std::string)
Descripción:	Método para distribuir los mensajes en los paneles de la pizarra.

Tabla 2.18: Descripción de la clase Server.

Continúa en la próxima página

Nombre: Server	
Tipo de clase: Control	
atributo:	**Tipo:**
port	short
planificador	Planificador*
Para cada responsabilidad:	
Nombre:	Server()
Descripción:	Constructor de la clase.
Nombre:	Server()
Descripción:	Destructor de la clase.
Nombre:	run()
Descripción:	Método para iniciar el servidor.

Tabla 2.19: Descripción de la clase Session.

Nombre: Session	
Tipo de clase: Control	
atributo:	**Tipo:**
panel	Panel*
planificador	Planificador*
connection	Conecction
socket	Socket
Para cada responsabilidad:	
Nombre:	Session()
Descripción:	Constructor de la clase.
Nombre:	Session()
Descripción:	Destructor de la clase.
Nombre:	run()
Descripción:	Método para iniciar la sesión del cliente.

Tabla 2.20: Descripción de la clase IEvent.

Nombre: IEvent	
Tipo de clase: Control	
atributo:	**Tipo:**
modules	list<string >
Para cada responsabilidad:	
Nombre:	getModules()
Descripción:	Método para acceder a los módulos.
Nombre:	getForModulos(std::string): list<string>
Descripción:	Método para devolver los paneles a los que va dirigido un mensaje.

Conclusiones del capítulo

La descripción de los requisitos de software y su traducción a historias de usuarios permitió documentar con claridad las funcionalidades del sistema.

Con la definición del protocolo de comunicación de MOCIC se garantiza que el envío de información a través de mensajes entre todos los módulos del proyecto sea de forma homogénea.

El uso de un patrón arquitectónico basado en Plugins permite una solución extensible para la inclusión de nuevos eventos predefinidos. El empleo de los patrones de asignación de responsabilidades: Experto, Bajo Acoplamiento y Alta Cohesión aseguran un diseño estructurado en clases con un bajo nivel de complejidad y establece la menor dependencia posible entre ellas, dando la responsabilidad de una tarea a la clase que conoce toda la información necesaria para ello.

Los diagramas presentados visualizan con claridad las clases con sus atributos, métodos y relaciones, constituyendo el punto de partida y guía para la implementación del sistema.

Capítulo 3

Implementación y prueba del módulo Controlador

Introducción

A continuación se presenta la documentación asociada a la implementación del módulo Controlador. Se incluyen diagramas que facilitan el entendimiento de los componentes y el despliegue del sistema. Son presentadas las planillas con las descripciones de los casos de prueba que se realizaron al módulo para su validación y se exponen los resultados obtenidos en cada una de ellas.

3.1. Implementación

La implementación de un software puede verse como la creación de un sistema a través de la realización de los requisitos definidos en la etapa de planificación.

3.1.1. Plan de Entrega

En el Plan de entrega se incluyen las características de las iteraciones a realizar durante el desarrollo del sistema. Tiene como entrada las historias de usuario previamente definidas. La asignación de cada historia de usuario a una iteración determinada se basó en la prioridad definida por el cliente para cada una de las mismas.

A continuación se muestra el Plan de entrega del módulo Controlador.

Tabla 3.1: Plan de Entrega.

Release	Descripción de la iteración	Orden de la HU a implementar	Duración
1	Una vez concluida esta iteración se habrá logrado la comunicación entre el Controlador y el resto de los módulos de MOCIC.	Con-1, Con-2, Con-3	7
2	Una vez concluida esta iteración el módulo Controlador contará con opciones de configuración que agilizarán el proceso de categorización.	Con-4, Con-5, Con-6,	6
3	Una vez concluida esta iteración se podrá acceder a las configuraciones del Controlador a través de un fichero y se tendrán registros de las trazas del módulo.	Con-7, Con-8	3

3.1.2. Diagramas de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Un componente es una unidad modular con interfaces bien definidas que es reemplazable en su entorno. Representan todos los elementos de software que intervengan en la realización de la aplicación dígase simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc [22].

En las Figuras 3.1 y 3.2 se muestra el diagrama de componentes del módulo Controlador, dividido por paquetes para un entendimiento más claro.

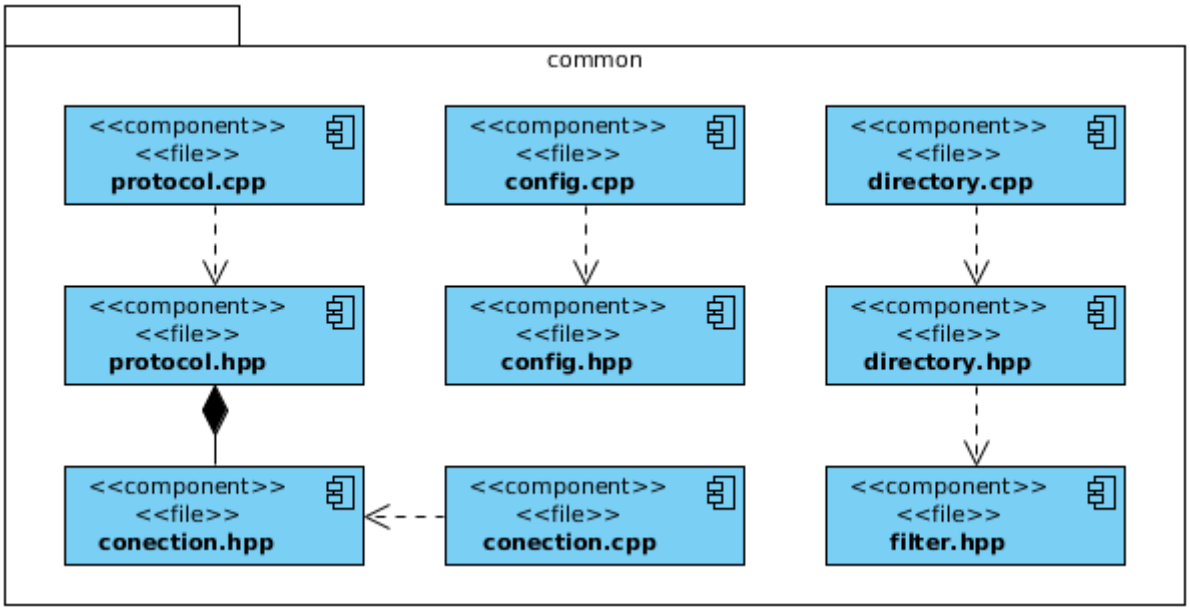


Figura 3.1: Diagrama de Componentes - Paquete common

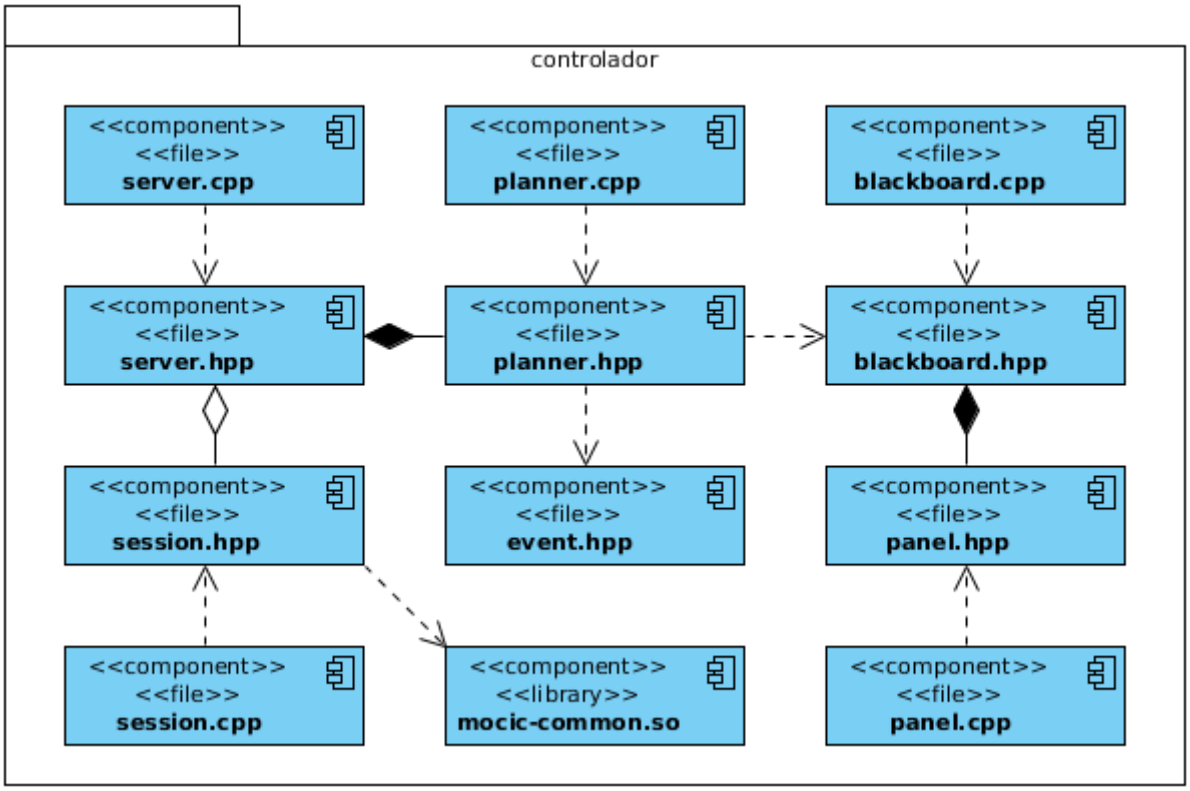


Figura 3.2: Diagrama de Componentes - Paquete controlador

3.1.3. Diagrama de despliegue

Los diagramas de despliegue muestran el flujo de ejecución en la arquitectura de los sistemas. Los nodos están conectados por vías de comunicación para crear sistemas de red [22].

En la Figura 3.3 se muestra el diagrama de despliegue del módulo Controlador.

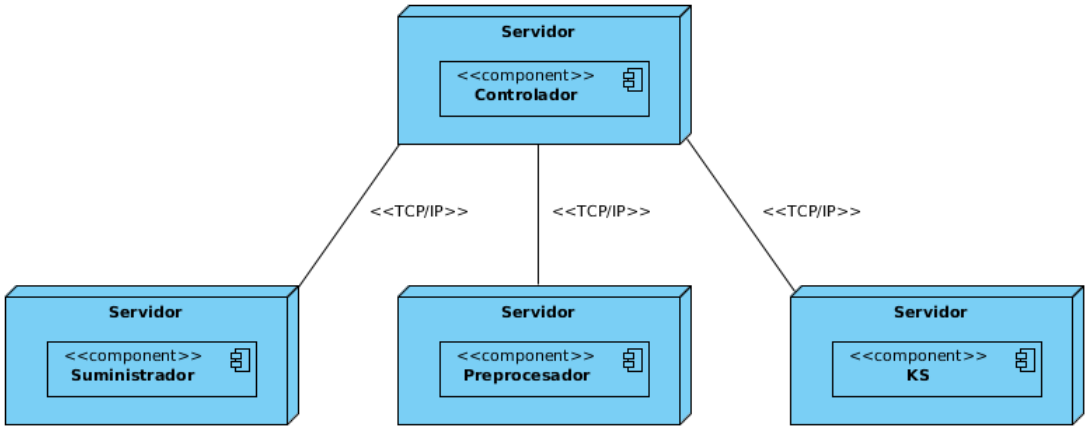


Figura 3.3: Diagrama de Despliegue

3.2. Revisión del código

Las revisiones automáticas del código ofrecen garantías sobre el nivel de calidad del software asegurando que este cumpla con los estándares y prácticas definidas. Una vez que se realiza la revisión se recoge un conjunto de resultados, entre los más comunes se encuentran:

- Errores potenciales.
- Incumplimiento de los estándares de programación.
- Malas prácticas de programación difundidas en la organización.
- Posibles fraudes.

Tener ese tipo de información a disposición de los desarrolladores permite tomar acciones al respecto para asegurar el nivel de calidad requerido.

Revisión con la herramienta Valgrind.

En la primera revisión realizada con la herramienta Valgrind se detectó una fuga de memoria de 424983 bytes en 24000 bloques causada por la ausencia del destructor en la estructura de datos que contiene los mensajes. Luego de implementado este destructor la herramienta arrojó 0 bytes de fuga de memoria.

3.3. Diseño y ejecución de las Pruebas de Software

La realización de pruebas representa una actividad de gran relevancia dentro del desarrollo de software, siendo una de las vías más comunes para controlar la calidad del producto que se esté implementando.

3.3.1. Pruebas unitarias

Las pruebas unitarias se utilizan para comprobar la validez del código escrito y el correcto funcionamiento de cada módulo por separado antes de pasar a la integración del sistema completo. Se debe tratar de probar la mayor cantidad de código posible ya que estas también pueden ser utilizadas como documentación. Las pruebas deben ser diseñadas de manera que puedan ser reutilizadas y la ejecución de una no debe interferir en la ejecución de otras.

A partir de la Tabla 3.2 hasta la 3.11 se muestran las pruebas unitarias realizadas a los métodos de mayor importancia dentro del módulo.

Tabla 3.2: Matriz de caso de prueba. Config_load.

Función	Escenario	Ruta del fichero	Respuesta del sistema
Continúa en la próxima página			

load	Escenario 1: Abrir un fichero XML.	V	Se carga el fichero XML.
load	Escenario 2: La ruta del fichero es incorrecta.	I	Error, no se encuentra la ruta indicada.

Tabla 3.3: Matriz de caso de prueba con datos. Config_load.

Función	**Escenario**	**Ruta del fichero**	**Respuesta del sistema**	**Resultado Obtenido**
load	Escenario 1: Abrir un fichero XML.	“tests/config/test.conf”	Se carga el fichero XML.	Resultado esperado: Verdadero
load	Escenario 2: La ruta del fichero es incorrecta.	“tests/conf/tests.conf”	Error, no se encuentra la ruta indicada.	Resultado esperado: Falso.

Tabla 3.4: Matriz de caso de prueba. Config_getTagValues.

Función	**Escenario**	**Ruta del fichero**	**Respuesta del sistema**
getTagValues	Escenario 1: Encontrar datos en el XML.	V	filename.
getTagValues	Escenario 2: La ruta del fichero es incorrecta.	I	Error, no se encuentra la ruta indicada.

Tabla 3.5: Matriz de caso de prueba con datos. Config_getTagValues.

Función	**Escenario**	**Ruta del fichero**	**Respuesta del sistema**	**Resultado Obtenido**
Continúa en la próxima página				

getTagValues	Escenario 1: Encontrar datos en el XML.	“debug.filename”	filename.	Resultado esperado: Verdadero
getTagValues	Escenario 2: La ruta del fichero es incorrecta.	“debug.filenames”	Error, no se encuentra la ruta indicada.	Resultado esperado: Falso.

Tabla 3.6: Matriz de caso de prueba. Directory_load.

Función	**Escenario**	**Ruta del directorio**	**Respuesta del sistema**
load	Escenario 1: Indexar archivos del directorio.	V	Se indexan los archivos.
load	Escenario 2: La ruta del directorio es incorrecta.	I	Error, no se encuentra la ruta indicada.

Tabla 3.7: Matriz de caso de prueba con datos. Directory_load.

Función	**Escenario**	**Ruta del directorio**	**Respuesta del sistema**	**Resultado obtenido**
load	Escenario 1: Indexar archivos del directorio.	“tests/directory”	Se indexan los archivos.	Resultado esperado: Verdadero.
load	Escenario 2: La ruta del directorio es incorrecta.	“tests/directori”	Error, no se encuentra la ruta indicada.	Resultado esperado: Falso.

Tabla 3.8: Matriz de caso de prueba. Directory_getFiles.

Función	**Escenario**	**Si es recursiva**	**Respuesta del sistema**

Continúa en la próxima página

getFiles	Escenario 1: Buscar archivos en un directorio, recursivamente.	V	Lista con los datos encontrados.
getFiles	Escenario 2: Buscar archivos en un directorio, no recursivamente.	V	Lista con los datos encontrados.
getFiles	Escenario 3: Se introducen valores incorrectos	I	Error de compilación (falta de argumentos).

Tabla 3.9: Matriz de caso de prueba con datos. Directory_getFiles.

Función	**Escenario**	**Si es recursiva**	**Respuesta del sistema**	**Resultado obtenido**
getFiles	Escenario 1: Buscar archivos en un directorio, recursivamente.	True	Lista con los datos encontrados.	Resultado esperado: Verdadero.
getFiles	Escenario 2: Buscar archivos en un directorio, no recursivamente.	False	Lista con los datos encontrados.	Resultado esperado: Verdadero.
getFiles	Escenario 3: Se introducen valores incorrectos	“ ”	Error de compilación (falta de argumentos).	Error de compilación.

Tabla 3.10: Matriz de caso de prueba. Directory_getFilesFilters.

Función	**Escenario**	**Filtro**	**Si es recursiva**	**Respuesta del sistema**
Continúa en la próxima página				

getFilesFilters	Escenario 1: Buscar archivos en un directorio recursivamente	V	V	Lista con los archivos encontrados en el directorio, según las restricciones de la búsqueda.
getFilesFilters	Escenario 2: Buscar archivos en un directorio, no recursivamente	V	V	Lista con los archivos encontrados en el directorio, según las restricciones de la búsqueda.
getFilesFilters	Escenario 3: Se introducen valores incorrectos	I	I	Error de compilación (falta de argumentos).

Tabla 3.11: Matriz de caso de prueba con valores. Directory_getFilesFilters.

Función	**Escenario**	**Filtro**	**Si es recursiva**	**Respuesta del sistema**	**Resultado obtenido**
getFilesFilters	Escenario 1: Buscar archivos en un directorio recursivamente	“.odt”	True	Lista con los archivos encontrados en el directorio, según las restricciones de la búsqueda.	Resultado esperado: Verdadero.
getFilesFilters	Escenario 2: Buscar archivos en un directorio, no recursivamente	“.odt”	False	Lista con los archivos encontrados en el directorio, según las restricciones de la búsqueda.	Resultado esperado: Verdadero.
getFilesFilters	Escenario 3: Se introducen valores incorrectos	“ ”	“ ”	Error de compilación (falta de argumentos).	Error de compilación.

3.3.2. Pruebas funcionales

A partir de la Tabla 3.12 hasta la 3.17 se muestran algunas de las pruebas funcionales realizadas al módulo Controlador junto a los resultados obtenidos.

Tabla 3.12: Caso de prueba Func_3.

Casos de Prueba	
Número de caso de prueba: Func_3	**Número de Historia de Usuario:** Con-3
Descripción de la Prueba: Realizar balance de carga entre distintas instancias de un módulo.	
Condiciones de ejecución: Mensajes en el Controlador disponibles para el módulo a probar.	
Entradas: Ejecutar varias instancias del módulo.	
Resultado esperado: No existencia de instancias inactivas mientras el Controlador tenga mensajes por procesarse.	
Evaluación: Prueba satisfactoria	

Tabla 3.13: Caso de prueba Func_4.

Casos de Prueba	
Número de caso de prueba: Func_4	**Número de Historia de Usuario:** Con-4
Descripción de la Prueba: Habilitar módulos según eventos en funcionamiento.	
Entradas/Pasos de ejecución: 1. Adicionar un evento en el fichero de configuración. 2. Reiniciar el Controlador.	
Resultado esperado: Se adicionan al registro de logs del sistema, los módulos que están funcionando.	
Evaluación: Prueba satisfactoria	

Tabla 3.14: Caso de prueba Func_5.

Casos de Prueba	
Número de caso de prueba: Func_5	**Número de Historia de Usuario:** Con-5
Descripción de la Prueba: Salvar estado de las peticiones a procesar si ocurre algún error durante la ejecución.	
Entradas: Recibir un mensaje.	
Resultado esperado: Guardar en la carpeta del panel asignado al módulo el mensaje serializado.	
Evaluación: Prueba satisfactoria	

Tabla 3.15: Caso de prueba Func_6.

Casos de Prueba	
Número de caso de prueba: Func_6	**Número de Historia de Usuario:** Con-6
Descripción de la Prueba: Recuperar el estado de las peticiones salvadas.	
Condiciones de ejecución: Existencia de mensajes en el Controlador.	
Entradas: Reiniciar la ejecución del Controlador.	
Resultado esperado: El Controlador reestablece los paneles con los mensajes que tenía antes de reiniciarse.	
Evaluación: Prueba satisfactoria	

Tabla 3.16: Caso de prueba Func_7.

Casos de Prueba	
Número de caso de prueba: Func_7	**Número de Historia de Usuario:** Con-7
Descripción de la Prueba: Generar logs de información relacionados al módulo Controlador.	
Condiciones de ejecución: Ejecutar el demonio rsyslogd.	
Entradas: Recibir mensajes.	
Resultado esperado: Los logs del módulo se guardan en la ruta: /var/log/mocic/controller.log	
Continúa en la próxima página	

Evaluación: Prueba satisfactoria

Tabla 3.17: Caso de prueba Func_8.

Casos de Prueba	
Número de caso de prueba: Func_8	**Número de Historia de Usuario:** Con-8
Descripción de la Prueba: Cargar fichero de configuración con todos los parámetros de la aplicación.	
Condiciones de ejecución: Existencia del fichero de configuración.	
Entradas: Ruta del fichero de configuración.	
Resultado esperado: Cargar correctamente los datos de la aplicación.	
Evaluación: Prueba satisfactoria	

El departamento de Calidad del Centro de Ideo-Informática (CIDI) realizó una revisión de las funcionalidades del sistema encontrando una no conformidad no significativa que fue resuelta inmediatamente.

3.3.3. Pruebas de integración

Las pruebas de integración son ejecutadas para comprobar que todos los componentes del sistema operen correctamente cuando son combinados. El objetivo es tomar los componentes probados independientemente y construir una estructura de programa que esté acorde con lo que dicta el diseño.

A partir de la Tabla 3.18 hasta la 3.20 se presentan algunas de las pruebas de integración realizadas entre el módulo Controlador y el resto de los módulos de Motor de Categorización Inteligente de Contenido (MOCIC). Dada la similitud entre las pruebas realizadas a los módulos de Texto, Imagen, Roles, Comunidades, Decisor y Tópicos, solo se muestra la realizada en conjunto al de Imagen, el resto puede consultarse en el Anexo H de este documento.

Tabla 3.18: Caso de prueba Int_1.

Casos de Prueba	
Número de caso de prueba: Int_1	**Módulo a integrar:** Suministrador
Descripción de la Prueba: Comprobar que el Suministrador reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo Suministrador y el Preprocesador.	
Entradas/Pasos de ejecución: El módulo Suministrador le envía un mensaje al Controlador, este adiciona ese mensaje al panel del Preprocesador, cuando se haya preprocesado el mensaje se le envía la respuesta al Suministrador.	
Resultado esperado: El Suministrador recibe un mensaje del Controlador con la respuesta del mensaje enviado.	
Evaluación: Prueba satisfactoria	

Tabla 3.19: Caso de prueba Int_2.

Casos de Prueba	
Número de caso de prueba: Int_2	**Módulo a integrar:** Preprocesador
Descripción de la Prueba: Comprobar que el Preprocesador reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo de Preprocesador.	
Entradas/Pasos de ejecución: El módulo Preprocesador realiza una petición al Controlador, este le envía un mensaje con el documento a preprocesar.	
Resultado esperado: Se recibe un mensaje del módulo Preprocesador con la respuesta del documento preprocesado.	
Evaluación: Prueba satisfactoria	

Tabla 3.20: Caso de prueba Int_3.

Casos de Prueba
Continúa en la próxima página

Número de caso de prueba: Int_3	**Módulo a integrar:** Imagen
Descripción de la Prueba: Comprobar que el subsistema de Imagen reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo de Imagen.	
Entradas/Pasos de ejecución: El módulo de Imagen le hace una petición al Controlador, este le envía un mensaje con el documento a procesar.	
Resultado esperado: Se recibe un mensaje del módulo de Imagen con la respuesta del documento procesado.	
Evaluación: Prueba satisfactoria	

Durante la realización de las pruebas de integración se identificaron tres no conformidades que se solucionaron en un plazo de veinticuatro horas.

3.3.4. Pruebas de rendimiento

Las pruebas de rendimiento son las que se le realizan a un software para comprobar su correcto funcionamiento bajo determinadas condiciones de trabajo. La computadora utilizada para ejecutar las pruebas posee las siguientes características:

- Microprocesador: Intel Core 2 CPU T5600 @ 1.83 GHz x2.
- Memoria: Kingston DDR2, 2,5 GB.
- Disco duro: Fujitsu MHV2160B 5600 Rpm.
- Tarjeta de red: Broadcom Corporation NetXtreme BCM5751M Gigabit Ethernet PCI Express.
- Sistema operativo: Ubuntu Desktop, versión 12.04 de 32-bit, núcleo Linux 3.2.0-25-generic-pae.

En los gráficos de las Figuras 3.4 a la 3.6 se muestra la relación entre el tiempo (en segundos) y la cantidad de mensajes, en la Figura 3.7 se relaciona el uso de la memoria y la cantidad de mensajes.

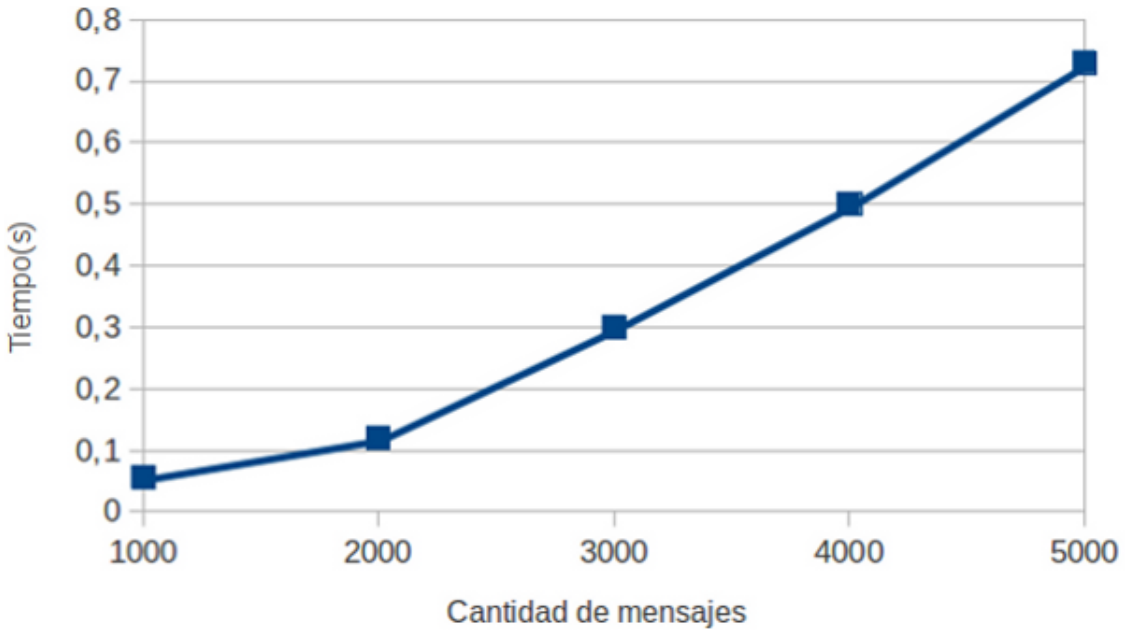


Figura 3.4: Gráfico de envío de mensajes.

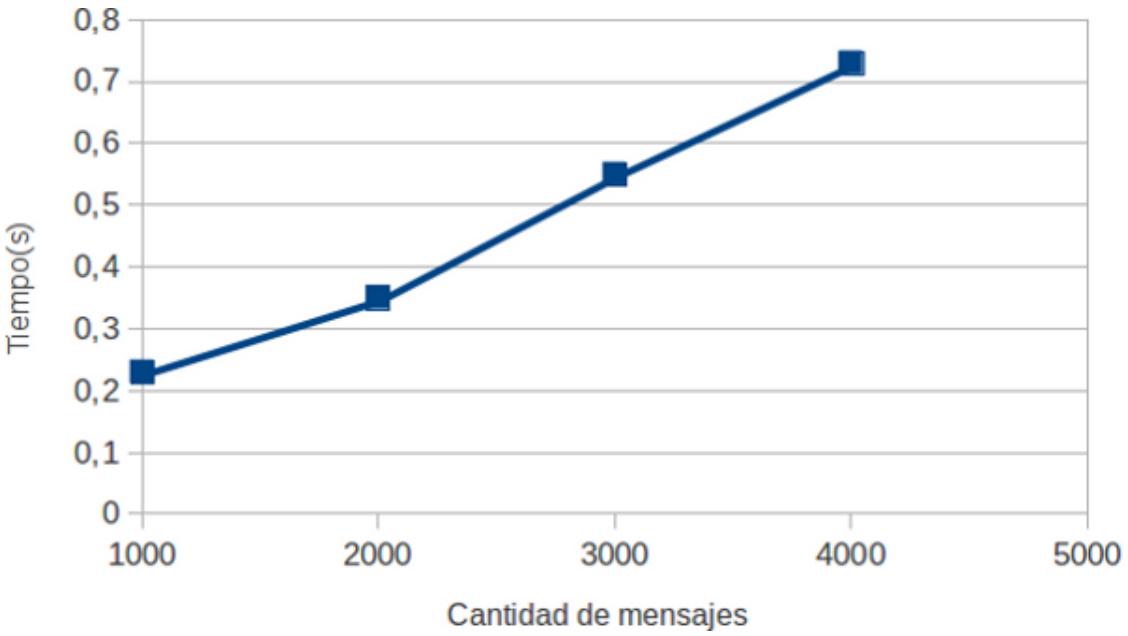


Figura 3.5: Gráfico de recibo de mensajes.

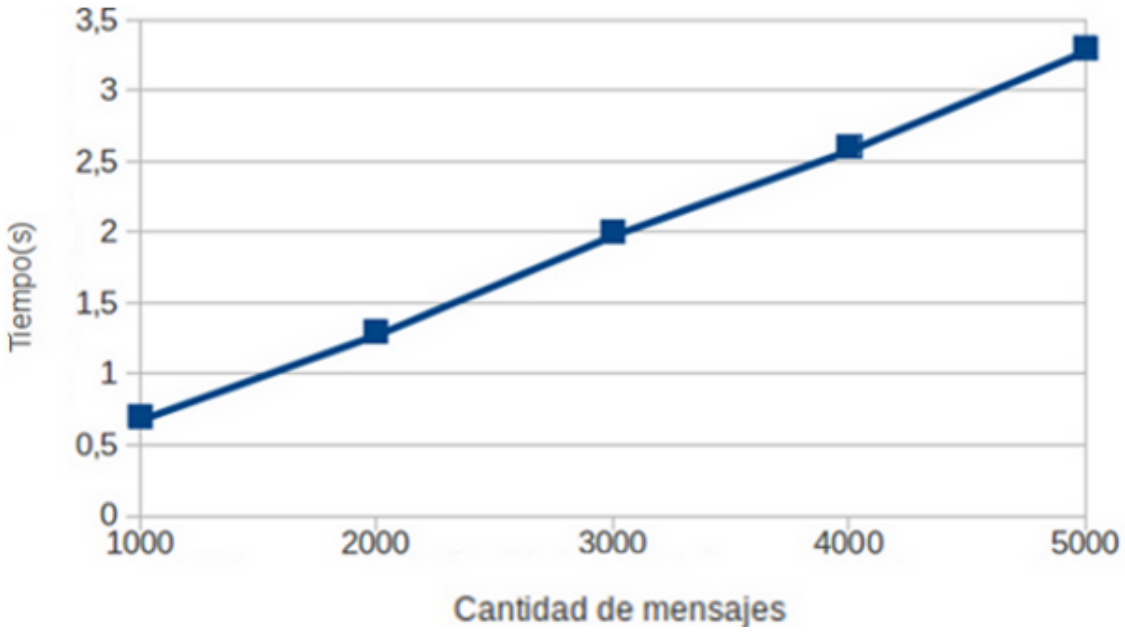


Figura 3.6: Gráfico de envío y recibo de mensajes.

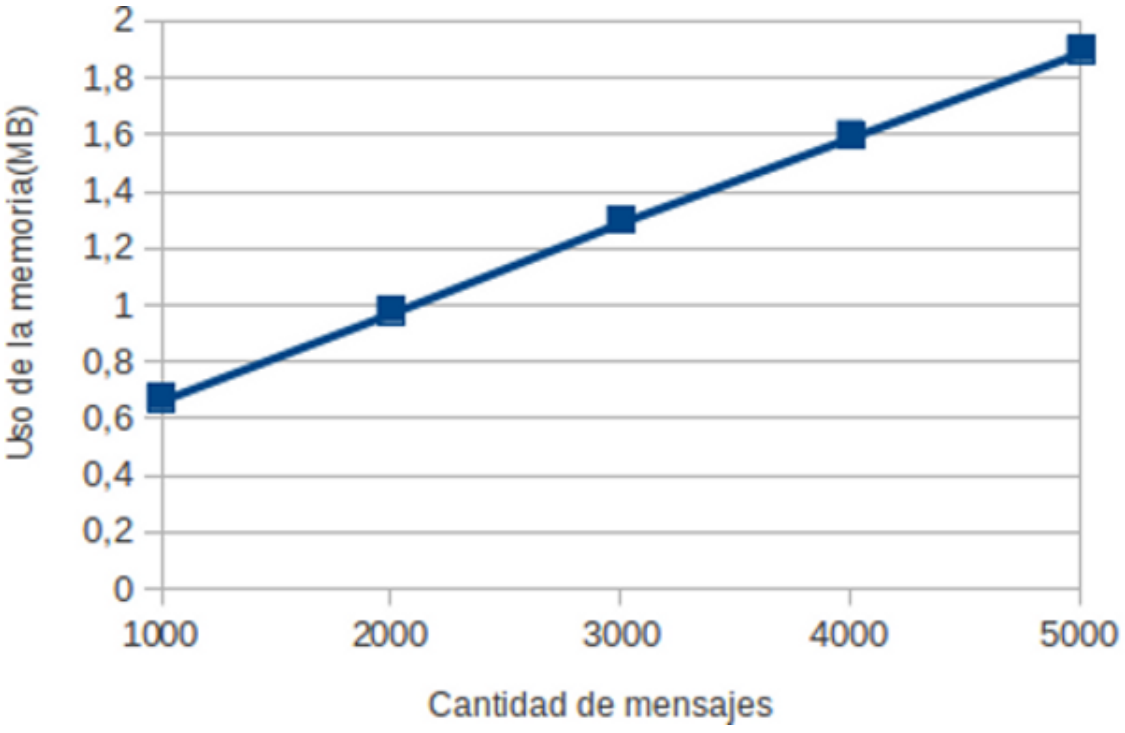


Figura 3.7: Gráfico del uso de la memoria.

Conclusiones del capítulo

A partir del diagrama de componentes se describieron las diferentes partes físicas del sistema y sus relaciones. Como un elemento de gran importancia se destaca la creación del paquete Common, el cual se utilizó para la comunicación entre todos los módulos de MOCIC.

Teniendo en cuenta la evaluación por parte del cliente de cada una de las funcionalidades implementadas, a través de los casos de prueba, fue validado el correcto funcionamiento del sistema implementado. Se comprobó que el Controlador se integra correctamente a MOCIC por lo cual el motor queda listo para ser utilizado.

Conclusiones

- Con la utilización de SXP, como metodología de desarrollo de software, se garantizó que la documentación y los artefactos generados fueran solamente los imprescindibles pudiendo dedicar más tiempo a la implementación del módulo.
- La selección del mecanismo de control basado en eventos predefinidos contribuyó a definir la arquitectura propia del Controlador mediante el uso de un patrón arquitectónico basado en Plugins.
- El uso de un patrón arquitectónico basado en Plugins dotó al módulo de la flexibilidad necesaria para la inclusión de nuevos eventos de forma dinámica.
- La escritura y almacenamiento de logs de información relacionados al módulo contribuyó a la depuración y detección de errores en la aplicación.
- Una vez definido el protocolo de comunicación a utilizar con todos sus parámetros se implementó una biblioteca de enlace dinámico que fue proporcionada al resto de los módulos del proyecto permitiendo establecer la comunicación entre todos estos.
- El diseño y la aplicación de diferentes tipos de pruebas validaron el correcto funcionamiento del módulo mediante la corrección de errores y deficiencias encontradas durante la ejecución de las mismas.
- Con el desarrollo completo del módulo Controlador se le proporcionó a MOCIC un componente imprescindible de acuerdo con la arquitectura de software que implementa, motivo por el cual el motor puede funcionar correctamente y llevar a cabo todos sus procesos de forma satisfactoria.

Recomendaciones

Una vez concluida la investigación se proponen las siguientes recomendaciones vinculadas al entendimiento y mejora del sistema desarrollado:

- Crear un certificado para el proyecto MOCIC que permita la autenticación y conexión cifrada por SSL entre los módulos.
- Implementar la escucha de la señal **reload** del comando *start-stop-daemon* para recargar la configuración.
- Analizar la posibilidad de implementar el Controlador de forma distribuida.

Acrónimos

CDDL Licencia Común de Desarrollo y Distribución

CI Comunidades de Interés

CIDI Centro de Ideo-Informática

Gof Gang of Four

GPL Licencia Pública General de GNU

GRASP *General Responsibility Asignment Software Patterns*

IA Inteligencia Artificial

IDE Entorno de Desarrollo Integrado

IEEE *Institute of Electrical and Electronics Engineers*

ITU Unión Internacional de Telecomunicaciones

KS Fuente de Conocimiento: por sus siglas en inglés: Knowledge Source

KSI Fuente de Conocimiento Instanceada: Knowledge Source Instantiation

LRP Lista de Reserva del Producto

MOCIC Motor de Categorización Inteligente de Contenido

SXP SCRUM *Extreme Programming*

UCI Universidad de las Ciencias Informáticas

XP *Extreme Programming*

Referencias Bibliográficas

[1] Unión Internacional de Telecomunicaciones. The word in 2011: Facts and figures. 2011.

[2] M.R. Cabrer. *Una arquitectura para sistemas inteligentes adaptativos basada en el modelo de pizarra*. PhD thesis, Ph. D. thesis, Universidad de Vigo, 2000.

[3] D.D. Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.

[4] N. Carver and V. Lesser. Evolution of blackboard control architectures. *Expert Systems with Applications*, 7(1):1–30, 1994.

[5] K. W. Chau and F. Albermani. An expert system on design of liquid-retaining structures with blackboard architecture. *Expert Systems*, 21(4):183 – 191, 2004.

[6] B. Hayes-Roth. A blackboard architecture for control. *Artificial intelligence*, 26(3):251–321, 1985.

[7] H.P. Nii. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. In *Computation & intelligence*, pages 447–474. American Association for Artificial Intelligence, 1995.

[8] V. Jagannathan, R. Dodhiawala, and L.S. Baum. *Blackboard architectures and applications*, volume 3. Academic Pr, 1989.

[9] R. Rizo, F. Llorens, M. Pujol, and I. Artificial. Arquitecturas y comunicación entre agentes. Technical report, SBN 84-8454-182-7, Páginas, inicial: 181, final: 214, Fecha: 2002, Editorial: ECU. Alicante.

[10] T. Morgan and R. Engelmore. Blackboard systems, 1988.

[11] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy. The hearsay-ii speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys (CSUR)*, 12(2):213–253, 1980.

[12] V.R. Lesser and L.D. Erman. A retrospective view of the hearsay-ii architecture. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 790–800. Citeseer, 1977.

[13] H.P. Nii, E.A. Feigenbaum, and J.J. Anton. Signal-to-symbol transformation: Hasp/siap case study. *AI magazine*, 3(2):23, 1982.

[14] H.P. Nii. Blackboard application systems, blackboard systems and a knowledge engineering perspective. *AI magazine*, 7(3):82, 1986.

[15] R. Engelmore and A. Terry. Structure and function of the crysalis system. In *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 1*, pages 250–256. Morgan Kaufmann Publishers Inc., 1979.

[16] A. Terry. Using explicit strategic knowledge to control expert systems. *Blackboard Systems, Addison-Wesley*, pages 159–188, 1988.

[17] D.D. Corkill, V.R. Lesser, and E. Hudlická. Unifying data-directed and goal-directed control: An example and experiments. In *Proceedings of the National Conference on Artificial Intelligence*, pages 143–147, 1982.

[18] D.D. Corkill. Framework for organizational self-design in distributed problem solving networks. *Dissertation Abstracts International Part B: Science and Engineering[DISS. ABST. INT. PT. B- SCI. & ENG.],*, 43(12):1983, 1983.

[19] L. Conway, V.R. Lesser, and D.G. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI magazine*, 4(3):15, 1983.

[20] H. Laasri and B. Maitre. Flexibility and efficiency in blackboard systems: Studies and achievements in atome. *Blackboard Architectures and Applications*, pages 309–322, 1989.

[21] M. Fowler, R. SCOTT, and K. Scott. *UML gota a gota*. Pearson Educación, 1999.

[22] S.W. Ambler. *The elements of UML 2.0 style*. Cambridge Univ Pr, 2005.

[23] Gladys Marsi Peñalver Romero. *MA-GMPR-UR2 Metodología ágil para proyectos de software libre.* PhD thesis, Universidad de las Ciencias Informáticas, 2008.

[24] R.S. Pressman. *Ingeniería del software*. McGraw Hill(Sexta Edición), 2005.

[25] Root Secure in General. Root secure :: The 2 faces of security. 2008.

Bibliografía

[ÁS] C. Álvarez and A.F. Soto. Diseño de un sistema multiagente para un software de simulación de celdas de producción robóticas.

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[BRMZ95] R.J. Boxwell, I.V. Rubiera, B. McShane, and J.R. Zaratiegui. *Benchmarking para competir con ventaja*. McGraw-Hill, 1995.

[CLM10] I. Castillo, F. Losavio, and A. Matteo. La orientación a aspectos y el nuevo estándar square para requisitos de software. *Rev. Fac. Ing. UCV*, 25(4):17–25, 2010.

[DL86] E.H. Durfee and V.R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, volume 58, page 64, 1986.

[DL91] E.H. Durfee and V.R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *Systems, Man and Cybernetics, IEEE Transactions on*, 21(5):1167–1183, 1991.

[Dur87] E.H. Durfee. A unified approach to dynamic coordinatio: Planning actions and. 1987.

[GSO98] A. Garcia-Serrano and S. Ossowski. Inteligencia artificial distribuida y sistemas multiagentes. *Inteligencia Artificial*, 2(6), 1998.

[HR95] B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(1-2):329–365, 1995.

[HRPL+95] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic. A domain-specific software architecture for adaptive intelligent systems. *Software Engineering, IEEE Transactions on*, 21(4):288–301, 1995.

[MÁ] J.I.B. Marañón and E.P. Álvarez. Aplicaciones de la tecnología agente a los sistemas de información de las organizaciones.

[MR] L. Machuca and P. Rodríguez. Arquitectura multiagente para un sistema e-learning centrado en la enseñanza de idiomas. *Nuevas Ideas en Informática Educativa*, 5:83–91.

[NA79] H.P. Nii and N. Aiello. Age (attempt to generalize): A knowledge-based program for building knowledge-based programs. In *Proceedings of the 6th international joint conference on Artificial intelligence-Volume 2*, pages 645–655. Morgan Kaufmann Publishers Inc., 1979.

[NCBS] V. Nicolau, J.O. Coronel, J.F. Blanes, and J. Simó. Implementación de un middleware de control para sistemas con recursos limitados.

[NS07] N. Nethercote and J. Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM Sigplan Notices*, 42(6):89–100, 2007.

[Pli08] Darya Plinere. Blackboard architecture programming for product life cycle stage definition. *Scientific Proceedings of RTU: Computer Science*, 36:1 – 7, 2008.

[PP11] K. Paternina Palacio. Ingeniería del software, un enfoque práctico. *INGENIATOR*, 1(1), 2011.

[PSB97] J.L. Posadas, J. Simó, and F. Blanes. Un modelo para el desarrollo de aplicaciones distribuidas. el servidor de comunicaciones. *Proc. of Jornadas españolas de Automática (JA'97). Gerona*, 1997.

[PSP+03] JL Posadas, JE Simó, JL Poza, P. Pérez, G. Benet, and F. Blanes. Una arquitectura para el control de robots móviles mediante delegación de código y sistemas multiagente. In *IV Workshop de Agentes Físicos-WAF*, volume 3, 2003.

[RK91] E. Rich and K. Knight. Introduction to artificial intelligence, 1991.

[Som05] I. Sommerville. *Ingeniería del software*. Pearson Educación, 2005.

Anexo A

Justificación del uso de la Observación

El proceso de observación del objeto, se puede describir en los siguientes pasos:

1. Tomar conciencia del objeto.
2. Reconocer el objeto a grandes rasgos.
3. Describir el objeto.

1. **Tomar conciencia del objeto.**

   El proyecto MOCIC cuenta con un conjunto de módulos, en su mayoría dotados de inteligencia artificial, que de forma unida permitirán automatizar el proceso de categorización de grandes volúmenes de documentos digitales. Este proyecto implementa una arquitectura de Pizarra, la cual está compuesta básicamente por 3 elementos: las Fuente de Conocimientos (KSs), la pizarra y el control. A grandes rasgos, esta arquitectura funciona de la siguiente forma: las KSs están especializadas en una tarea en concreto y acceden a la pizarra, donde se encuentran los datos, soluciones parciales y estado del problema, para hacer cambios de acuerdo a su conocimiento; el objetivo del control es organizar, planificar y definir la manera en la que las KSs acceden a la pizarra.

2. **Reconocer el objeto a grandes rasgos.**

   Actualmente, en MOCIC se cuenta con las KSs, que son todos sus módulos, pero no existe forma de que estos se comuniquen con la pizarra para realizar cambios en la solución del problema planteado, en su caso, el problema es la clasificación de contenidos. Por tal motivo se necesita contar con un módulo que se encargue de implementar un mecanismo de control que garantice la comunicación entre todas las KSs para que el motor funcione correctamente.

3. **Describir el objeto.**

   Según las características del proyecto y la arquitectura que implementa se hace necesario realizar un estudio del arte sobre este tema para obtener documentación suficiente para poder seleccionar el mecanismo de control adecuado para MOCIC. Esta es la razón por la que se plantea como problema a resolver: ¿Cómo controlar el proceso de Categorización de contenidos del proyecto MOCIC teniendo en cuenta la arquitectura de Pizarra? Y como objeto de estudio: la arquitectura de software de Pizarra. Enmarcando el estudio en los mecanismos de control de la arquitectura de Pizarra.

Anexo B

Lista de Reserva del Producto

Tabla B.1: Lista de Reserva del Producto

Asignado a	Item*	Descripción	Estimación	Estimado por
Requisitos funcionales				
Prioridad: Muy Alta				
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	1	Comunicar los módulos del Motor de Categorización Inteligente de Contenido.	3	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	2	Monitorear los procesos de Preprocesamiento, Categorización de documentos y Agrupamiento.	2	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	3	Balancear la carga entre las distintas instancias de los módulos.	2	Prog
Prioridad: Alta				
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	4	Habilitar módulos según eventos en funcionamiento.	2	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	5	Salvar estado de las peticiones a procesar si ocurre algún error durante la ejecución.	2	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	6	Recuperar el estado de las peticiones salvadas.	2	Prog
Prioridad: Media				
Continúa en la próxima página				

Eddy Fonseca (Prog) Mayra Ortíz (Prog)	7	Generar logs de información relacionados al módulo Controlador.	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	8	Cargar fichero de configuración con todos los parámetros de la aplicación.	2	Prog
Requisitos no funcionales				
Categoría: Restricciones en el diseño y la implementación				
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	1	Usar TCP/IP para la comunicación.	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	2	Utilizar XML para el fichero de configuración.	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	3	Utilizar como lenguaje de programación C++.	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	4	Utilizar NetBeans como Entorno de Desarrollo Integrado (IDE).	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	5	Utilizar RapidSVN para el control de versiones.	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	6	Utilizar Valgrind para la revisión automática del código.	1	Prog
Categoría: Software				
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	7	Usar Ubuntu 10.04 como sistema operativo.	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	8	Instalar Biblioteca LibMagic.	1	Prog
Eddy Fonseca (Prog) Mayra Ortíz (Prog)	9	Instalar Biblioteca Boost.	1	Prog
Categoría: Soporte				

Continúa en la próxima página

Eddy Fonseca (Prog) Mayra Ortíz (Prog)	10	Crear un instalador para la arquitectura Debian y sus derivados.	3	Prog

Anexo C

Historias de usuario

Tabla C.1: Historia de usuario: Con-4.

	Historia de Usuario
Número: Con-4	**Nombre de Historia de Usuario:** Habilitar módulos según eventos en funcionamiento.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Prog	**Iteración asignada:** 2
Prioridad en Negocio: Media	**Puntos Estimados:** 2
Descripción:	Una vez identificado el evento a ejecutar se habilitan los módulos requeridos.

Tabla C.2: Historia de usuario: Con-6.

	Historia de Usuario
Número: Con-6	**Nombre de Historia de Usuario:** Salvar estado de las peticiones a procesar si ocurre algún error durante la ejecución
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Prog	**Iteración asignada:** 2
Prioridad en Negocio: Media	**Puntos Estimados:** 2
Descripción:	En caso de que ocurriera algún error durante la ejecución de un proceso se deben guardar los estados de las peticiones que se estuvieran procesando para continuar haciéndolo en otro instante.

Tabla C.3: Historia de usuario: Con-7.

Historia de Usuario	
Número: Con-7	**Nombre de Historia de Usuario:** Recuperar el estado de las peticiones salvadas.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Prog	**Iteración asignada:** 2
Prioridad en Negocio: Media	**Puntos Estimados:** 2
Descripción:	Una vez que el sistema pueda funcionar nuevamente es necesario que las peticiones que fueron interrumpidas puedan volver a procesarse desde el punto en el que se quedaron antes del error.

Tabla C.4: Historia de usuario: Con-8.

Historia de Usuario	
Número: Con-8	**Nombre de Historia de Usuario:** Generar logs de información relacionados al módulo Controlador.
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Prog	**Iteración asignada:** 2
Prioridad en Negocio: Baja	**Puntos Estimados:** 1
Descripción:	Se deben registrar en un fichero de texto las trazas del módulo para su posterior comprobación si fuera necesario.

Tabla C.5: Historia de usuario: Con-9.

Historia de Usuario	
Número: Con-9	**Nombre de Historia de Usuario:** Cargar fichero de configuración con todos los parámetros de la aplicación.
Modificación de Historia de Usuario Número: Ninguna	
Continúa en la próxima página	

Usuario: Prog	**Iteración asignada:** 2
Prioridad en Negocio: Baja	**Puntos Estimados:** 2
Descripción:	El módulo debe ser capaz de cargar el fichero en el que se encontrarán todas sus opciones de configuración de forma que estas puedan ser modificadas con más facilidad.

Anexo D

Categorías analizadas por el Motor de Categorización Inteligente de Contenido

Tabla D.1: Categorías analizadas por MOCIC.

Categorías	Descripción	ID
Ciencias	Contenidos de carácter científico (Física, Geografía, Matemáticas, Medio ambiente, Química, Tecnología, Agricultura, Astronomía, Biología, Ciencia alternativa, Ciencia tecnología y sociedad, Ciencias de la Tierra, Ciencias Sociales).	1
Computadoras	Contenidos relacionados con computadoras; hardware (fabricantes, componentes, precios), software (S.O, programas, temas relacionado al software libre y privativo, computación paralela, Inteligencia Artificial, Juegos, algoritmos), temas de telecomunicaciones, robótica, realidad virtual, seguridad informática.	2
Deportes	Contenidos relacionados con el deporte; clubes, equipos, federaciones deportivas, resultados deportivos, eventos como: Juegos Olímpicos, campeonatos mundiales.	3
Juegos	Contenidos sobre juegos de computadoras, que promueven los juegos de azar, casino y agencias de apuestas.	4
Sustancias dañinas	Contenidos donde se promueve el uso de drogas.(LSD, heroína, cocaína, XTC, pot, *amphetamines*, hemp, éxtasis), que promueven el consumo de alcohol y el tabaquismo.	5
Continúa en la próxima página		

Salud	Contenidos que promueven la salud; medicinas, hospitales, hábitos dietéticos, enfermedades, primeros auxilios, sexología, salud mental.	6
Pornografía	Contenidos que contienen actividad sexual explícita, pornografía infantil, la Pedofilia, la Pederastia.	7
Violencia	Contenidos que hagan apología o inciten a la violencia, a la guerra, al racismo, a la desigualdad entre el hombre y la mujer, a la violencia familiar, a la fabricación de explosivos, venenos, manuales para fabricación de bombas, instrucciones para asesinar personas.	8

Anexo E

Estructura de las respuestas de los módulos

Tabla E.1: Estructura de las respuestas.

Respuesta	Valores válidos para las respuestas en dependencia del módulo.
-n text -r X	La X equivale al identificador de la categoría al que pertenece el texto. El rango admitido por X es [0,8] donde X es un número entero. Cuando X = 0, significa que no se pudo categorizar el documento en ninguna de las categorías, por un error ocurrido mientras se analizaba.
-n face -r X:X:X:..:X	Cada X representa una imagen y el valor de la X representa la cantidad de rostros detectados en la imagen. El rango admitido por X es [0,50] donde X es un número entero. Si no se pudo analizar la imagen no será pasado ningún valor. Ver ejemplo 1.
-n nude -r X:X:X:..:X	Cada X representa una imagen y el valor de la X representa si se identificó o no desnudez en en la imagen. El rango admitido por X es [0,1] donde X es un número entero. Cuando X = 0 no se detectó desnudez en esta imagen. Cuando X = 1 sí se detectó desnudez en esta imagen. Si se pasa un elemento vacío significa que no se pudo analizar la imagen. Ver ejemplo 1.
Continúa en la próxima página	

-n object -r X:X:X:..:X	Cada X representa un objeto y toma el valor del identificador de la categoría a la que pertenece. El rango admitido por X es [0,8] donde X es un número entero. Cuando X = 0 no se pudo clasificar el objeto en ninguna de las categorías. Si se pasa un elemento vacío significa que no se pudo analizar la imagen. Ver ejemplo 1.
-n link -r X	La X equivale al identificador de la categoría a la que pertenece el contenido teniendo en cuenta los enlaces. El rango admitido por X es [0,8] donde X es un número entero. Cuando X = 0 No se pudo clasificar el contenido teniendo en cuenta los enlaces en ninguna de las categorías definidas.
-n ocr -r X	La X equivale al identificador de la categoría a la que pertenece el texto clasificado. El rango admitido por X es [0,8] donde X es un número entero . Cuando X = 0 significa que no se pudo clasificar el texto en ninguna de las categorías definidas.

Ejemplo 1: “-r 3:4::5:6::8:4:1”. No se pudieron analizar ni la imagen 3 ni la 6.

Anexo F

Descripción de las clases del paquete common

Tabla F.1: Descripción de la clase FilterEmpty.

Nombre: FilterEmpty	
Tipo de clase: Control	
Para cada responsabilidad:	
Nombre:	FilterEmpty()
Descripción:	Constructor de la clase.
Nombre:	execute(path:path): bool
Descripción:	Ejecuta la acción del filtro.

Tabla F.2: Descripción de la clase FilterExtension.

Nombre: FilterExtension	
Tipo de clase: Control	
atributo:	**Tipo:**
list	list <string >
Para cada responsabilidad:	
Nombre:	FilterExtension(list <string >)
Descripción:	Constructor de la clase.
Nombre:	execute(path:path): bool
Continúa en la próxima página	

Descripción:	Ejecuta la acción del filtro.

Tabla F.3: Descripción de la clase FilterMime.

Nombre: FilterMime	
Tipo de clase: Control	
atributo:	**Tipo:**
list	list <string >
magic_cookie	magic_t
Para cada responsabilidad:	
Nombre:	FilterMime(list <string >)
Descripción:	Constructor de la clase.
Nombre:	execute(path:path): bool
Descripción:	Ejecuta la acción del filtro.

Anexo G

Descripción de las clases del paquete controlador

Tabla G.1: Descripción de la clase Categorization.

Nombre: Categorization	
Tipo de clase: Control	
Para cada responsabilidad:	
Nombre:	Categorization()
Descripción:	Constructor de la clase.
Nombre:	getForModulos()
Descripción:	Método para devolver los paneles a los que va dirigido un mensaje.

Tabla G.2: Descripción de la clase Clustering.

Nombre: Clustering	
Tipo de clase: Control	
Para cada responsabilidad:	
Nombre:	Clustering()
Descripción:	Constructor de la clase.
Nombre:	getForModulos()
Descripción:	Método para devolver los paneles a los que va dirigido un mensaje.

Tabla G.3: Descripción de la clase Preprocessing.

Nombre: Preprocessing	
Tipo de clase: Control	
Para cada responsabilidad:	
Nombre:	Preprocessing()
Descripción:	Constructor de la clase.
Nombre:	getForModulos()
Descripción:	Método para devolver los paneles a los que va dirigido un mensaje.

Anexo H

Casos de prueba de Integración

Tabla H.1: Caso de prueba Int_4.

Casos de Prueba	
Número de caso de prueba: Int_4	**Módulo a integrar:** Texto
Descripción de la Prueba: Comprobar que el módulo de Texto reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo de Texto.	
Entradas/Pasos de ejecución: El módulo de Texto le hace una petición al Controlador, este le envía un mensaje con el documento a procesar.	
Resultado esperado: Se recibe un mensaje del módulo de Texto con la respuesta del documento procesado.	
Evaluación: Prueba satisfactoria	

Tabla H.2: Caso de prueba Int_6.

Casos de Prueba	
Número de caso de prueba: Int_6	**Módulo a integrar:** Roles
Descripción de la Prueba: Comprobar que el módulo de Roles reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo de Roles .	
Entradas/Pasos de ejecución: El módulo de Roles le hace una petición al Controlador, este le envía un mensaje con el documento a procesar.	
Continúa en la próxima página	

Resultado esperado: Se recibe un mensaje del módulo de Roles con la respuesta del documento procesado.
Evaluación: Prueba satisfactoria

Tabla H.3: Caso de prueba Int_7.

Casos de Prueba	
Número de caso de prueba: Int_7	**Módulo a integrar:** Comunidades
Descripción de la Prueba: Comprobar que el módulo de Comunidades reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo de Comunidades.	
Entradas/Pasos de ejecución: El módulo de Comunidades le hace una petición al Controlador, este le envía un mensaje con el documento a procesar.	
Resultado esperado: Se recibe un mensaje del módulo de Comunidades con la respuesta del documento procesado.	
Evaluación: Prueba satisfactoria	

Tabla H.4: Caso de prueba Int_8.

Casos de Prueba	
Número de caso de prueba: Int_8	**Módulo a integrar:** Tópicos
Descripción de la Prueba: Comprobar que el módulo de Tópicos reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo de Tópicos.	
Entradas/Pasos de ejecución: El módulo de Tópicos le hace una petición al Controlador, este le envía un mensaje con el documento a procesar.	
Resultado esperado: Se recibe un mensaje del módulo de Tópicos con la respuesta del documento procesado.	
Evaluación: Prueba satisfactoria	

Tabla H.5: Caso de prueba Int_9.

Casos de Prueba	
Número de caso de prueba: Int_9	**Módulo a integrar:** Decisor
Descripción de la Prueba: Comprobar que el módulo de Decisor reciba la información del Controlador correctamente y viceversa.	
Condiciones de ejecución: Debe estar en ejecución el módulo de Decisor.	
Entradas/Pasos de ejecución: El módulo de Decisor le hace una petición al Controlador, este le envía un mensaje con el documento a procesar.	
Resultado esperado: Se recibe un mensaje del módulo de Decisor con la respuesta del documento procesado.	
Evaluación: Prueba satisfactoria	