Universidad de las Ciencias Informáticas

Facultad 6

BHIKE: HERRAMIENTA PARA EL DESARROLLO DE APLICACIONES SOBRE EL *FRAMEWORK* KSIKE

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor: Yadir Hernández Batista

Tutores:

Ing. Antonio Membrides Espinosa

Ing. Listley Castell Espinosa

La Habana, mayo de 2013
"Año 55 de la Revolución"

Declaración de autoría

Por este medio yo, Yadir Hernández Batista con carné de identidad 89122539461, declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste, firma la presente declaración jurada de autoría en La Habana a los ____ días del mes _____ del año _____.

Yadir Hernández Batista **Autor**	Listley Castell Espinosa **Tutor**	Antonio Membrides Espinosa **Tutor**

Datos de Contacto

Tutor principal: Ing. Antonio Membrides Espinosa
Correo Electrónico: amembrides@uci.cu
Especialidad: Ingeniero en Ciencias Informáticas.
Año de Graduación: 2008
Institución: XETID, MINFAR
Desempeño: Graduado de la UCI, ha incursionado en varias líneas de investigación tales como: el desarrollo de tecnologías bases para la construcción aplicaciones multimedia sobre plataformas libres, creación de una plataforma para el desarrollo de aplicaciones basadas en tecnologías Web, orientadas a Sistemas de Información Geográficos donde ejerció como arquitecto, así como desarrollo de *framework* basados en tecnologías libres para la construcción de software.

Tutor secundario: Ing. Listley Castell Espinosa
Correo Electrónico: lcastell@uci.cu
Especialidad: Ingeniero en Ciencias Informáticas
Año de Graduación: 2012
Institución: Departamento GEYSED, UCI
Desempeño: Graduada con título de oro en la UCI, actualmente trabaja en el Departamento de Geo-informática de UCI, desempeñándose en el rol de Analista en la Línea de Productos de Software Aplicativos SIG.

Resumen

El *framework* Ksike se muestra como una variante ideal para el desarrollo de aplicaciones para todo tipo de entornos y plataformas pues, debido a las abstracciones de conceptos que presenta, brinda una solución a los problemas comunes que resultan propios del desarrollo de software. Posibilita además, eliminar la brecha existente entre las formas particulares de comportamiento para los distintos ambientes de ejecución de aplicaciones, entiéndase: *Desktop* (Escritorio), *Web* y *Mobile* (Móvil). Aunque presenta disímiles beneficios, Ksike no cuenta con estudios profundos sobre sus posibilidades de empleo y más limitante aún, no cuenta con una herramienta que facilite el uso de sus capacidades.

En aras de garantizar un desarrollo ágil, ameno y con calidad, se concibe BHike como solución informática que pretende flexibilizar el uso del marco de trabajo Ksike. El mismo se presenta en calidad de Entorno Integrado de Desarrollo conteniendo un conjunto de funcionalidades que le permitan a los desarrolladores obtener, de una manera más intuitiva, la materialización de un proyecto; teniendo en consideración la posibilidad de hacer de esta una herramienta extensible al aprovechamiento de otras tecnologías. Auxiliándose del mencionado *framework* y logrando integrar un conjunto de principios que deban converger en el marco de trabajo Ksike-BHike, el desarrollador tendrá en sus manos una potente herramienta de desarrollo.

Palabras clave:

BHike, Entorno Integrado de Desarrollo, *Framework*, Ksike, Software.

Índice de contenidos

INTRODUCCIÓN

El empleo de herramientas ha sido la clave del éxito de la especie humana a través de los tiempos. Desde el uso de rocas, lanzas, flechas, hoz, hasta las más modernas utilizadas en las industrias de producción de energía con fuentes nucleares y las versátilmente empleadas en la medicina para la realización de operaciones complejas. Además, los disímiles ejemplos de aplicaciones informáticas o software, que constituyen hoy en día una compleja red de herramientas que se extiende a nivel global; todas estas fueron ideadas para la satisfacción de necesidades del hombre ante problemas particulares de su quehacer diario.

Según Roger S. Pressman "*El software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos. En los pasados 50 años, el software ha pasado de ser una resolución de problemas especializada y una herramienta de análisis de información, a ser una industria por sí misma*" (Pressman, 2002), la cual ha aumentado de manera considerable y constituye actualmente una de las más cotizadas.

La citada industria del software genera además problemas en la cotidianidad de sus trabajadores y estos no quedan exentos al desarrollo y empleo de sus propias herramientas que, sobre todo, agilizan el trabajo mejorando la calidad y la eficiencia en el desarrollo de los productos informáticos. La gama de dichas herramientas es muy amplia: procesadores de textos, editores de imágenes, manejadores de multimedia, compiladores, intérpretes, herramientas de modelado, análisis y diseño de productos de software; estas últimas muy cercanas a la concepción e implementación de nuevos software o para el mantenimiento de las soluciones existentes. Lo cierto es que, sin lugar a dudas, las herramientas informáticas son necesarias para aumentar la productividad y aprendizaje en cada una de las esferas de la vida.

En pos de agilizar el proceso de desarrollo de software se han creado también numerosos marcos de trabajo que brindan varias herramientas, funcionalidades, implementación de patrones y muchos otros recursos, que permiten al desarrollador enfocarse en actividades propiamente inherentes a sus modelos de negocio, pues implementarlos resulta cómodo y viable cuando se hace uso de algún marco de trabajo. Actualmente existen disímiles recursos para el desarrollo de software que en gran medida son aplicables a varias situaciones, pero los mismos divergen en lenguajes de programación, conceptos, estándares de implementación y modelos arquitectónicos. Estos se rigen además por filosofías distintas para alcanzar un propósito definido a su medida.

Algunas son menos eficientes, otras más prácticas y seguras o con ávidos propósitos, enfocadas a interfaces, comunicación, gestión de recursos, etcétera. Lo que sí es cierto es que el número de dichas tecnologías es inmenso, por lo que cuando se hace necesario implementar cualquier sistema, la incertidumbre sobre cuál tecnología elegir para dar solución a los problemas resulta ser una de las verdaderas preocupaciones de los desarrolladores. Finalmente, en la mayoría de los casos los proyectos de desarrollo emplean solo aquellas herramientas en las que se tenga habilidades o en las que empíricamente se han desarrollado soluciones previas. Esto significa tiempo a favor del desarrollo

pues dichos proyectos se ahorran la demora de tener que investigar sobre la existencia de mejores soluciones y la formación de especialistas en las tecnologías resultantes de la citada investigación.

En muchas ocasiones, por desconocimiento, no se emplean las herramientas, metodologías y marcos de trabajos adecuados para ofrecer soluciones óptimas. Por otro lado, muchas veces es necesario integrar recursos de diferentes plataformas sobre una misma solución informática y no siempre se cuenta con el ambiente ideal para la realización de dicho proceso, convergiendo en problemas de incompatibilidad y pérdida del recurso tiempo para dar respuestas prácticas y solubles.

A raíz del cada vez mayor y más creciente número de marcos de trabajo orientados al desarrollo de software, de las abismales diferencias entre los conceptos y filosofías planteados por cada uno; a finales del año 2010 un grupo de ingenieros de la UCI[1], dirigidos por el Ingeniero en Ciencias Informáticas Antonio Membrides Espinosa, inician el desarrollo del *framework* Ksike. Su objetivo principal consiste en aglomerar un gran número de tecnologías bajo una misma arquitectura, asumiendo como proyección la ruptura de las barreras entre el desarrollo de aplicaciones orientadas a entornos *web* y de escritorio. Este a su vez potencia el desarrollo de productos de software orientado a componentes reutilizables (Membrides Espinosa, 2010).

Ksike propone numerosas ventajas entre las que se puede encontrar la facilidad de integración con otras tecnologías, lo que significa que puedan ser incorporados otros *frameworks* en el desarrollo de los sistemas que emplean a Ksike como soporte. Además permite la implementación de soluciones sobre una arquitectura modular que posibilita alta flexibilidad y escalabilidad en el desarrollo de software. El mismo brinda también numerosos recursos para la construcción de aplicaciones de monitoreo, SIG[2], así como recursos para aplicaciones sobre dispositivos móviles; los cuales pueden ser de gran aprovechamiento en los proyectos productivos del centro GEYSED[3] de la UCI y otros centros productivos de la misma entidad. Entre otros elementos, propone además la generación de componentes para los entornos: *web y desktop* con alta reusabilidad entre ellos.

A pesar de los disímiles recursos que ofrece el marco de trabajo, su empleo no constituye un elemento a tener en cuenta por la comunidad de desarrollo y los proyectos productivos de la UCI para la desarrollo de soluciones informáticas. Esto representa uno de los puntos más desfavorables para el desarrollo de la tecnología, pues de esta forma no se aprovechan las ventajas que posee el ambiente de la Universidad como incubadora de empresas, sobre todo por las posibilidades que brinda esta para garantizar la soberanía tecnológica en Cuba.

Las incubadoras de empresa de base tecnológica, constituyen un elemento estratégico de la política de innovación de los países para alcanzar el desarrollo regional. Estas además, representan una de las opciones más fructíferas que pueden aplicar todas las universidades interesadas en investigación y desarrollo. En tal sentido, la literatura especializada destaca su utilización para favorecer el desarrollo

[1] **UCI** Universidad de las Ciencias Informáticas.

[2] **SIG** Sistemas de Información Geográficas.

[3] **GEYSED** Geo-informática y de Señales Digitales.

de un sector o rama industrial de empresas intensivas en conocimientos (científicos, tecnológicos o de mercado) (Universia). La Universidad de las Ciencias Informáticas de acuerdo a las características propias de su comunidad, la cual es precursora por excelencia del desarrollo de software, constituye actualmente uno de los ambientes más propicios en Cuba para potenciar el desarrollo del campo de la Informática.

Luego de analizar las características que posee la Universidad como una potencial incubadora de empresas, surge la siguiente interrogante ¿por qué no se aprovecha el *framework* Ksike, producto de la propia UCI, como una alternativa para desarrollar software? Esto se evidencia a sobremanera pues el mismo actualmente no está siendo explotado, a ninguna escala, por los proyectos productivos y la comunidad de desarrollo de dicha entidad, ni siquiera con carácter académico. A raíz de la situación existente, el arquitecto principal del proyecto Ksike, Ingeniero Antonio Membrides Espinosa y el resto de los integrantes del equipo de desarrollo del *framework*, han definido una serie de estrategias dirigidas a fomentar el uso del marco de trabajo y a la inserción del mismo entre el conjunto de las soluciones factibles para el desarrollo de software en la UCI.

Estudios sobre los procesos asociados al desarrollo de software sustentado en Ksike, han evidenciado que su empleo resulta engorroso. El mismo define una arquitectura compleja de ficheros y directorios que al realizarse de forma manual ralentiza la creación, mantención y organización de los proyectos. Además que, para realizar estas operaciones, es necesario tener conocimiento previo de la información y distribución de los ficheros según define la arquitectura del marco de trabajo. También posee diversidad de parámetros, formatos y ficheros de configuración lo que introduce factores de incertidumbre en los desarrolladores.

Entre otros elementos, el *framework* proporciona además un esquema propio de POO[4] para JavaScript. Este lenguaje en sí mismo no está diseñado para soportar este tipo de paradigma de programación, por lo que el *framework* brinda un esquema que simula comportamientos muy similares al POO, permitiéndole a los desarrolladores abstraerse a su uso. Además otro de los elementos que constituye un freno sobre el empleo de este marco de trabajo resulta ser que el mismo no posee automatizaciones para: gestión de proyectos, gestión de plantillas, manejo de dependencias, manejo de componentes externos, gestión de *logs* y tratamiento de errores.

La presente investigación constituye una de las estrategias claves que se han implementado para dar solución a la situación antes mencionada. La misma define como **problema a resolver**: ¿Cómo facilitar el empleo del *framework* Ksike como una base tecnológica para la construcción de soluciones informáticas que se desarrollan en la comunidad y los proyectos productivos de la UCI?

Para darle solución al problema identificado se propone como **objetivo general**: Desarrollar una herramienta que facilite el desarrollo del software que se sustenta en el *framework* Ksike. Se tiene

[4] **POO** Siglas de Programación Orientada a Objetos.

como **objeto de estudio**: El proceso de desarrollo de IDE[5] orientados a *framework* de desarrollo y como **campo de acción**: Los procesos que intervienen en el desarrollo de software sustentado en Ksike.

A partir del análisis del objetivo general se derivaron los siguientes **objetivos específicos**:

1. Elaborar el diseño teórico-metodológico de la investigación.
2. Diseñar una arquitectura que permita escalabilidad e incorporación de nuevas funcionalidades.
3. Desarrollar un prototipo funcional que solvente las principales deficiencias en el empleo de Ksike.
4. Emplear el *framework* Ksike para la implementación del sistema en pos de demostrar, probar y ampliar sus aptitudes para el desarrollo de software.
5. Probar que la solución desarrollada cumpla con los requisitos propuestos.

Para darle cumplimiento al conjunto de objetivos específicos antes descritos se trazan las siguientes **tareas de la investigación**:

1. Realizar un estudio del estado del arte de las herramientas que permitan agilizar el desarrollo de software, garantizando configuraciones, administración, desarrollo y seguridad de las aplicaciones.
2. Caracterizar las tendencias arquitectónicas orientadas a IDE para tomar las buenas prácticas en el diseño de arquitecturas para este tipo de herramientas.
3. Describir las estructuras estándares a emplear en la herramienta en función de la generación e intercambio de información.
4. Caracterizar herramientas orientadas al análisis sintáctico y semántico del código.
5. Caracterizar herramientas orientadas a la documentación de código fuente.
6. Caracterizar herramientas orientadas al seguimiento y control de la ejecución del código fuente.
7. Realizarle pruebas a los componentes implementados.

En este punto el autor propone como **idea a defender** que**:** La utilización de una herramienta que agilice los procesos de desarrollo de software sustentado en el *framework* Ksike facilitará el empleo del mismo por parte de la comunidad de desarrollo y los proyectos productivos de la UCI.

Para la realización de la investigación se utiliza una **estrategia descriptiva** pues se cuenta con bibliografía abundante referente a las ventajas del desarrollo de software sustentado en los marcos de trabajo, así como de las facilidades que brindan estos para agilizar el desarrollo de soluciones con alta calidad. Dadas las características del problema los **métodos científicos** que se emplean como base para el descubrimiento del conocimiento son:

Dentro de los **métodos teóricos**:

[5] **IDE** Acrónimo de *Integrated Development Environment* (Entorno de Desarrollo Integrado).

Analítico – sintético: Este método se emplea en el estudio y análisis de las bibliografías y soluciones existentes sobre los procesos de desarrollo de IDE, sus arquitecturas y comportamientos, de forma tal que permita agrupar los resultados obtenidos en un posible aporte al desarrollo del sistema.

Hipotético – deductivo: Este método se utiliza para definir, basado en soluciones existentes, que si utilizando herramientas para agilizar los procesos de desarrollo de software sobre otros marcos de trabajo ha devenido en resultados positivos, entonces se deduce que ocurrirá de igual manera con el *framework* Ksike.

Histórico – lógico: Este método se emplea durante la investigación del estado del arte de los sistemas informáticos que automatizan los procesos de desarrollo de aplicaciones que se sustentan en *frameworks,* para de esta forma lograr mejor entendimiento del empleo de estas tecnologías. Además se realiza una selección de un conjunto de bibliografías y recursos existentes en un período de tiempo de 5 años sobre los cuales se realizan los análisis pertinentes para la obtención de información beneficiosa para la investigación.

Dentro de los **métodos empíricos**:

Observación: Se aplica para entender mejor el proceso de desarrollo de aplicaciones con la asistencia de IDE de desarrollo. También para conocer cómo automatizar el empleo de *framework* y estándares definidos por los Entornos de Desarrollo Integrados para agilizar la obtención de soluciones informáticas. Para ello se realiza la observación de soluciones existentes de ese tipo que pueden ser útiles y aporten ideas. El tipo de observación a realizar es una observación participante sistemática, ya que se hará contacto directo con la realidad que se estudia y se sistematizarán los detalles más significativos, haciendo uso de instrumentos de apoyo tales como fotografías, modelos y diagramas.

Al concluir la investigación se prevé contar con varios **resultados** o **aportes prácticos** que contribuyan al aumento de la eficiencia, calidad y rapidez en la obtención de soluciones que sirvan además para mejorar el empleo de Ksike en el desarrollo de software. En aras de obtener mayor provecho y entregar resultados tangibles se prevé la materialización de los siguientes elementos:

- ✓ Una aplicación capaz de agilizar los procesos de desarrollo de software sustentado en Ksike. La misma ayudará además a minimizar la curva de aprendizaje de dicho marco de trabajo y servirá como ejemplo de su utilización, dado que se emplea Ksike como base tecnológica para la implementación de dicha aplicación.
- ✓ Una documentación determinada por la metodología de desarrollo empleada, que facilitará la extensión de las funcionalidades por parte de miembros externos al equipo original.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se abordan los conceptos y definiciones necesarias para otorgar una mayor comprensión al lector sobre el objeto de estudio, tratando de sentar las bases conceptuales para dar solución al objetivo general anteriormente determinado. Se describen detalladamente, además, las principales características del *framework* Ksike y también se realizan varios análisis más profundos sobre las causas que afectan el desarrollo y empleo de esta tecnología.

1.1. Conceptos asociados al dominio del problema

1.1.1. Aplicación informática

Según el diccionario de la RAE[6], el término aplicación, proveniente del latín (*applicatio*, *-onis*), puede definirse en Informática como: "*programa preparado para una utilización específica, como el pago de nóminas, formación de un banco de términos léxicos, etc.*" (Real Academia Española). En otras referencias como el sitio web "*SearchSoftwareQuality"*, se define aplicación como: Diminutivo del término (software de aplicación). Son programas informáticos dineñados para realizar tareas específicas dirigidas por el usuario o en algunos casos, por otras aplicaciones (Rouse).

Luego de haber analizado algunos de los conceptos tratados por varios autores, puede definirse, a manera general, el término **aplicación** como: programa informático diseñado como herramienta para permitir a los usuarios realizar uno o diversos procesos de negocio. Generalmente suele resultar una solución informática para la automatización de ciertas tareas complicadas.

1.1.2. Entornos Integrados de Desarrollo

Los Entornos Integrados de Desarrollo (IDE, del inglés *Integrated Develompent Environment*) son conocidos también como entornos de diseño integrado, entorno integrado de depuración o entorno de desarrollo interactivo. De acuerdo con el sitio web Webopedia.com los IDE: Son aplicaciones de software que proveen a los usuarios de un constructor/diseñador de GUI[7], editor de código, un compilador y/o intérprete y un depurador (IT Business Edge). Margaret Rouse, agrega además que: Los IDE pueden ser aplicaciones independientes o pueden estar vinculados como partes compatibles con algunas aplicaciones (TechTarget, y otros, 2007).

Citando un artículo publicado en "ProgramacionDesarrollo.es", se define que un IDE "*(…) es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, puede utilizarse para varios (…) puede denominarse como un entorno de programación que ha sido tratado como un programa de aplicación (…) puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje*

[6] **RAE** Acrónimo de Real Academia Española.

[7] **GUI** Acrónimo de *Graphic User Interface,* en español Interfaz Gráfica de Usuario.

de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto (…)" (ProgramacionDesarrollo.es, y otros, 2011).

De manera general puede resumirse que los IDE son aplicaciones informáticas que proporcionan servicios integrales a los programadores de computadoras para el desarrollo de software, a través de un vasto conjunto de herramientas y recursos que facilitan el diseño e implementación de las aplicaciones. Se componen normalmente por:

1. Un editor de código fuente.
2. Un compilador y / o intérprete.
3. Automatización para la generación de herramientas.
4. Un depurador.

Luego de haber enunciado los principales componentes de los IDEs, debe comprenderse además que estos son sistemas para la automatización de los procesos de desarrollo de aplicaciones de software y que pueden ser para un único lenguaje de programación o para varios.

1.1.3. Módulo de aplicación

El término módulo, en Informática y especialmente en la rama de la programación de aplicaciones, está referido a una de las características conceptuales de la OOP[8], dígase la modularidad; según define Eugenia Bahit en su libro "OOP y MVC[9] en PHP[10]". Esta hace referencia a la modularidad como: la capacidad "*(…) que permite dividir una aplicación en varias partes más pequeñas (denominadas módulos), independientes unas de las otras (…)*" (Bahit).

El colectivo de autores del libro "Guía de Arquitectura N-Capas Orientada al Dominio" con .Net 4.0 (Beta), producido por Microsoft Ibérica S.R.L, tratan el término módulo desde el punto de vista del modelo del negocio. "*(…) Los módulos se utilizan como un método de organización de conceptos y tareas relacionadas (normalmente bloques de negocio diferenciados) y poder así reducir la complejidad desde un punto de vista externo (…)*" (de la Torre LLorente, y otros, 2010).

A modo de condensar lo referido con anterioridad por varios autores, se hace notar que en Ingeniería Informática, específicamente en una de sus disciplinas, la Arquitectura de Software, el término módulo hace referencia a componente, pieza o parte de software de aplicación. Constituye un elemento autónomo, escrito dentro de un contexto que posibilita que sus funcionalidades sean útiles en la creación de distintas piezas de software. Esto permite que sea desarrollado de manera independiente y pueda ser modificado internamente, ya sea por concepto de diseño o adición de nuevas funcionalidades lógicas, sin afectar significativamente el resto del sistema.

Para obtener una mejor comprensión sobre el término módulo, es necesario analizar otros dos conceptos: componente y *plugin*, que en las disciplinas de Ingeniería Informática comparten estrecha

[8] **OOP** Siglas de *Object Oriented Programming,* en español Programación Orientada a Objetos, POO.

[9] **MVC** Siglas de Modelo-Vista-Controlador.

[10] **PHP** Acrónimo de *Hypertext Pre-processor* (Pre-procesador de Hipertexto).

relación con el concepto de módulo anteriormente definido por lo que es imprescindible acotar sus similitudes y diferencias. El concepto de componente referido a arquitectura de software se entenderá como: organización fundamental de un sistema en partes que se relacionen unas con las otras, con el entorno, así como los principios que orientan su diseño y evolución. Debe quedar claro que en esta definición el concepto de "componente" es genérico e informal (Reynoso, y otros, 2004).

Hay varias docenas más de definiciones de componente, pero Clemens Alden Szyperski proporciona una muy adecuada: un componente de software es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas (Szyperski, y otros, 2002). Que sea una unidad de composición en vez de construcción quiere decir que no es preciso confeccionarla; se puede comprar hecha, se puede producir en casa para que otras aplicaciones de la empresa la utilicen en sus propias composiciones.

Por otra parte el término *plugin* según Rytis Sileika quiere decir: componente de software que extiende las funcionalidades de una aplicación principal. Esta técnica es muy conocida y usada por muchas aplicaciones como por ejemplo los navegadores web. Muchos de los navegadores más populares tienen soporte para extensiones o *plugin*s. A modo de ejemplo, una página web puede incluir un fichero de multimedia Adobe Flash embebido, pero el navegador no conoce (y no tiene que conocer) como manejar este tipo de fichero. Si existe un *plugin* que tiene la capacidad de procesar y mostrar el contenido de los archivos Adobe Flash, este es procesado por el *plugin;* en caso de que no existiera, el objeto simplemente no se mostrará al usuario final. La ausencia del *plugin* apropiado no constituye un impedimento para mostrar el contenido de la página web (Sileika, 2010).

A modo de resumen se pueden identificar los módulos, así como los *plugin* o extensiones, como tipos particulares de componentes de software. El primero como parte intrínseca de un programa informático y a los segundos como extensiones de las funcionalidades de una aplicación que pueden o no existir sin afectar el funcionamiento de la misma.

1.1.4. Plantilla de software

Aunque existen varias definiciones de plantillas o *template* como se le conoce en inglés, en este documento se hará referencia al término como: contenido pre-formateado que sirve como punto de inicio para la creación de un nuevo elemento. Sirven para evitar la creación constante de contenidos similares, permitiendo la replicación de estos (TechTerms.com).

De manera general las plantillas proporcionan elementos o patrones de similitud que permiten generar contenidos de forma ágil. Es análogo al término molde, muy empleado en la mecánica y otras disciplinas para la fabricación de partes. Las mismas proporcionan una separación entre la forma o estructura y su contenido. Es un medio que permite guiar, portar o construir un diseño o esquema predefinido. Agiliza el trabajo de reproducción de muchas copias idénticas o casi idénticas (que no tiene que ser tan elaborado, sofisticado o personal). Si se quiere un trabajo más refinado, más creativo,

la plantilla no es sino un punto de partida, un ejemplo, una idea aproximada de lo que se quiere hacer, o partes comunes de una diversidad de copias.

A partir de una plantilla pueden, así mismo, diseñarse y fabricarse otras nuevas. En la actualidad los sistemas de plantillas son muy utilizados para separar la lógica del programa del formato visualizado, permitiendo generar, a partir de datos variados, vistas muy similares, lo que permite acoplar a un mismo estilo las interfaces visuales de dichos sistemas. Típicamente, estas incluirán variables y posiblemente unos pocos operadores lógicos para permitir una mejor adaptabilidad de la misma.

1.1.5. Marco de trabajo

Algunos autores describen un *framework* en el desarrollo de software como una estructura de soporte definida en la cual otro proyecto puede ser organizado y desarrollado (Codebox). Marc Clifton, en su artículo "*What Is A Framework*" publicado en el sitio web del Proyecto Código (*Code Project*), menciona varios elementos categóricos de los marcos de trabajo. En aras de que los mismos facilitan el trabajo con complejas tecnologías, unifican componentes/objetos haciéndolos más útiles, promueven la implementación consistente y flexible de aplicaciones. Además los mismos pueden ser fácilmente depurados y probados (Clifton, 2003).

A modo de resumen un *framework* o marco de trabajo es un conjunto de bloques de software prefabricado que los desarrolladores pueden emplear, extender o personalizar a favor de desarrollar soluciones informáticas específicas. Con ellos los desarrolladores no tienen que preocuparse por implementar aplicaciones desde el inicio una y otra vez. Son colecciones de componentes/objetos que garantizan que ambos: diseño y código fuente, puedan ser reutilizados.

1.1.6. Proyecto de software

Según Juan Palacio, autor del libro *Scrum Manager* "Gestión de Proyectos", proyecto clásicamente se conoce como: "*Conjunto único de actividades necesarias para producir un resultado definido en un rango de fechas determinado y con una asignación específica de recursos*". Los proyectos tienen objetivos y características únicas, como son: la cantidad de trabajadores, así como la duración del mismo (Palacio, 2008).

Continuando con las definiciones descritas dentro de las disciplinas de ISW[11], en este caso en particular de las empleadas por el PMI[12], el PMBOK[13] en su cuarta edición, dedicada específicamente a la Dirección de Proyectos, sirve como guía a los desarrolladores y describe que: *"Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único. La naturaleza temporal de los proyectos indica un principio y un final definidos. El final se alcanza cuando se logran*

[11] **ISW** Acrónimo de Ingeniería de Software.

[12] **PMI** Siglas de *Project Management Institute,* en español Instituto de Administración de Proyectos.

[13] **PMBOK** Acrónimo de *Project Management Body of Knowledge,* en español Cuerpo de Conocimientos de la Gestión de Proyectos.

los objetivos del proyecto o cuando se termina el proyecto porque sus objetivos no se cumplirán o no pueden ser cumplidos, o cuando ya no existe la necesidad que dio origen al proyecto" (Jimenez, 2008).
En el contexto de la concepción de las aplicaciones informáticas empleando determinada tecnología, el proyecto está encaminado a describir la composición física basado en la alineación de las distintas partes que lo constituyen. En este intervienen ciertas particularidades tales como: la estructura jerárquica de sus entes, tipos de archivos, ficheros de configuración, etcétera (Membrides Espinosa, 2010).
La MSDN[14] brinda un concepto aterrizado al IDE *Visual Studio* que es de gran interés para este trabajo de investigación, pues categoriza a los proyectos como un paquete contenedor dentro de las soluciones, a fin de administrar, compilar y depurar lógicamente los elementos que componen la aplicación. Esto tiene como objetivo ayudar en la organización y realización de las tareas comunes con los elementos que se están desarrollando (Microsoft, 2011). "*El resultado de un proyecto puede ser un programa ejecutable (exe), un archivo de biblioteca dinámica (dll) o un módulo, entre otros*" (Microsoft, 2011).

1.2. Causas determinantes del Objeto de Estudio

1.2.1. Descripción general

En el ambiente de escritorio, por ser primogénito entre los entornos orientados a la construcción de aplicaciones, se han formulado la mayoría de las filosofías y técnicas generales aplicables al desarrollo de software. Con el surgimiento y evolución de la web, la utilización del protocolo HTTP[15], así como tecnologías AJAX[16] para establecer la comunicación entre cliente y servidor, se han creado un conjunto nuevo de filosofías devenidas en marcos de trabajo. Los mismos están dirigidos por ende al desarrollo de aplicaciones específicas para la web. La forma de desarrollo ha variado o se ha perfeccionado consecuentemente, haciendo diferentes actualmente los principios e implementación de soluciones para ambos entornos.
Ksike surge como necesidad de obtener una plataforma para la construcción de aplicaciones orientadas a entornos web, aprovechando las potencialidades de las mejores tendencias provenientes de ambientes de escritorio; está enfocado además en maximizar el rendimiento y agilizar el proceso de desarrollo. Sin embargo esto no debería de ir en decremento de la solidez del mismo, permitiendo soportar cualquier tipo de sistema indiferentemente de la lógica de negocio que este necesite gestionar. Otros de los elementos a tener en cuenta es que es sumamente extensible y fomenta la consistencia de código entre los desarrolladores. Además el mismo potencia el desarrollo de productos de software basados en el paradigma de POO así como todas las ventajas que este provee (Membrides Espinosa, 2010).

[14] **MSDN** Acrónimo de *MicroSoft Developer Network* Red de Desarrolladores de Microsoft.
[15] **HTTP** Siglas de *Hyper-Text Transfer Protocol* (Protocolo de Transferencia de Hiper-Texto).
[16] **AJAX** Acrónimo de *Asyncronic JavaScript and XML* (JavaScript y XML Asincrónicos).

Este proyecto se inició a mediados del mes de julio del 2010 y constituye el reflejo del interés profesional de sus integrantes en aras de lograr el objetivo propuesto, potenciar la construcción ágil de software. Inicialmente fue concebido como un método autodidacta para la comprensión a fondo del funcionamiento de otros marcos de trabajo, pero en la medida que se avanzó en la investigación se fueron incorporando nuevos conceptos, reorientándose de esta forma la meta a seguir (Membrides Espinosa, 2010). Ksike le permite al desarrollador concentrarse en la lógica de la aplicación y abstraerse de la comunicación cliente-servidor, asumiendo como proyección la ruptura de las barreras entre el desarrollo de aplicaciones orientadas a entornos web y de escritorio.

Entre las principales características de la versión más actualizada, la Ksike 1.1 Elephant, se encuentran:

- ✓ API[17] en el lenguaje JavaScript para el desarrollo de la parte cliente.
- ✓ API en el lenguaje PHP para el desarrollo del lado servidor.
- ✓ Enlazador de Componentes.
- ✓ Módulo para la gestión de bases de datos de PostgreSQL.
- ✓ Patrones de Diseño: *Singleton*, *Observer*, *Factory*.
- ✓ Mecanismo de publicación.
- ✓ Mecanismos de Inversión de Control e Inyección de Dependencias vinculados a un contenedor de servicios.

Como valor añadido al paquete de esta versión se encuentran además algunos *drivers* que brindan un conjunto de funcionalidades que extienden las capacidades del *framework*, tales como:

- ✓ *Driver* WallDOM para la generación de interfaces para dispositivos móviles de gama baja.
- ✓ *Driver* para el manejo de ficheros.
- ✓ *Driver* para el mapeo de bases de datos con el ORM[18] Doctrine.
- ✓ *Driver* que integra el motor de plantillas Twig[19].

Todos estos elementos son facilidades que brinda el marco de trabajo para la construcción de aplicaciones o programas informáticos. Además el *framework* propone una tecnología propia que se basa en el aprovechamiento de las mejores tendencias arquitectónicas. Permite tener un control total de su comportamiento y posibilita alta escalabilidad sobre las soluciones.

1.2.2. Situación problemática

Sin embargo, a pesar de los disímiles beneficios abordados anteriormente, el empleo de Ksike no constituye un hecho realista orientado al desarrollo de soluciones en las comunidades de desarrollo y los proyectos productivos de la UCI, quienes continúan seleccionando otras tecnologías en vez del suscitado marco de trabajo. Debido a que actualmente este no cuenta con los mejores niveles de

[17] **API** Siglás de *Application Programming Interface,* en español Interfaz de Programación de Aplicaciones.

[18] **ORM** Siglas de *Object-Relational Mapping*, en español Mapeo Objeto-Relacional.

[19] **Twig** Motor de Plantillas implementado en PHP.

divulgación, distribución y documentación, emplear el *framework* resulta engorroso. Además, el mismo posee varios procesos que complejizan el trabajo de los desarrolladores y que, constituyen en conjunto, las principales causas por las que actualmente no se aplica esta tecnología. De estos elementos se harán algunas descripciones a continuación.

1.2.2.1. Taxonomía de Ksike

Ksike propone dentro de su marco, esquemas organizativos o taxonomías para: proyecto, aplicación, *plugin*/módulo, *driver* y *packages*. Dichos esquemas están dirigidos a describir la composición física de las soluciones informáticas que se desarrollan haciendo uso del propio marco de trabajo. El *framework* permite manejar bajo estos conceptos, elementos tales como: la estructura jerárquica de sus componentes, varios tipos de archivo, dependencias, así como los ficheros de configuración. Enunciados estos, es de suponer que los mismos presenten una compleja estructura de directorios, ficheros de configuración, clases y otros recursos sobre la cual se hace una descripción a continuación.

Figura 1.2.1: Estructura de un proyecto Ksike.

El esquema de organización propuesto por Ksike para proyectos necesita de los directorios ***core***, ***lib*** y ***plugins***, como se puede observar en la Figura 1.2.1. Además requiere un fichero **index.html** o **index.php** que es el PAC[20] para la ejecución de los componentes y funciones del *framework*. De los directorios mencionados se debe acotar que el primero se emplea para manejar configuraciones o extensiones propias del núcleo del marco de trabajo para el proyecto. El segundo se usa para desplegar las librerías y componentes externos que serán utilizados para la construcción del proyecto. Por último el directorio ***plugins*** se emplea para organizar los *plugins* o módulos del sistema en desarrollo.

Por otro lado, las aplicaciones y *plugins*/módulos de Ksike, comparten casi 100% de similitud en cuanto a composición física, pues conceptualmente las aplicaciones de Ksike son los módulos principales de los proyectos. Estas están especializadas en gestionar los sucesos que ocurren en el sistema, así como el manejo de información asociada al resto de los módulos: principal elemento que diferencia la aplicación sobre los demás módulos. Las aplicaciones y módulos están compuestas por los directorios ***client*** y ***server***, que son empleados para separar la lógica de negocio aplicada a los entornos cliente y servidor respectivamente, como se puede apreciar en la Figura 1.2.2. Además poseen el directorio **cfg**, estrictamente empleado para almacenar datos de configuración. En el caso particular de las aplicaciones, se encuentra la presencia del directorio ***log***, donde se almacenan los registros de sucesos del sistema.

[20] **PAC** Siglas de Punto de Acceso Común.

Figura 1.2.2: Estructura de una aplicación y un *plugin* de Ksike.

Adentrándose en la estructura de directorios del lado cliente, como se puede apreciar en la Figura 1.2.3, se pueden identificar algunos otros directorios como son el **css**, empleado para los archivos de estilo de las aplicaciones *web*. También el directorio **img**, utilizado para los recursos de imágenes, los que son muy comunes y están muy aparejadas a los estilos CSS que se les aplican a las páginas web. Los elementos más importantes, de todos los que se encuentran en el cliente, resultan ser el directorio **js** y su contenido.

Cuando una aplicación o un módulo de Ksike definen hacer uso de alguna funcionalidad que se ejecute en el lado cliente, deben automáticamente incluir el directorio **js** y un fichero de JavaScript con el nombre del módulo o **Main.js** en caso de la aplicación. Dentro de este fichero se define una clase, con el mismo nombre del fichero, en la cual se programan las funciones principales que se ejecutan en el navegador *web*. En el dicho directorio se almacenan además todas las clases JavaScript que se empleen para la programación de la lógica del lado cliente.

Figura 1.2.3: Estructura interna del directorio *client*.

Figura 1.2.4: Estructura interna del directorio *server*.

Dentro de la estructura del servidor (directorio ***server***) se pueden encontrar los directorios ***common*** e ***include*** y un fichero con el nombre del módulo, al igual que en el lado cliente. En el fichero **Main.php**, en el caso del módulo de aplicación, se define una clase de igual nombre. La innegable necesidad en cuanto a la nomenclatura de los ficheros, tanto del lado cliente como servidor, reside en que Ksike maneja la comunicación dentro de un módulo direccionando las peticiones de manera automática desde el cliente (JavaScript) hacia el servidor (PHP), garantizando la independencia, tanto física como lógico-funcional de los módulos.

Luego de haber analizado al detalle el esquema de organización de Ksike, se induce que esta puede ser muy sencilla, pero también puede complejizarse a sobremanera de acuerdo a la cantidad de elementos que se incluyan en los proyectos. Existen numerosos directorios y la jerarquía de los mismos puede tener incluso varios niveles de profundidad. Además se hace necesario conocer el nombre, localización y contenido de numerosos ficheros que se utilizan para configuraciones o clases del sistema, elementos sobre los que se harán análisis posteriores. También de directorios específicos,

como los empleados para hacer carga dinámica/automática de contenidos, siendo el caso de los directorios: ***include***, que se encuentran localizados dentro del lado servidor.

Para garantizar la gestión de forma manual, o sea la creación y mantención de un proyecto de Ksike, hay que poseer un nivel elevado de conocimiento y de experiencia empleando el marco de trabajo. Aun así, el desarrollo de software sobre Ksike, resulta un reto que requiere de asistencia documental y especializada, medios que actualmente no están lo suficientemente elaborados o al alcance de los desarrolladores.

Es una realidad que los desarrolladores no se aventuren a emplear este marco de trabajo, pudiendo utilizar otros como Symfony o Zend Framework. Los mismos, aunque poseen ciertas similitudes o incluso mayor complejidad estructural en cuanto a componentes físicos (ficheros y directorios), son más empleados. Esto ocurre porque las comunidades de desarrollo de estos *frameworks* han implementado varias extensiones y herramientas básicas que gestionan la creación de proyectos sobre IDEs como NetBeans y PHPStorm. Con el uso de estos recursos se agilizan los procesos de generación y mantenimiento de los sistemas que sobre estos marcos de trabajo se desarrollan. Esto deviene en que se subutilice la tecnología Ksike frente a otras, debido a la carencia de este último en cuanto a documentación actualizada así como herramientas que automaticen sus procesos. Estos elementos subliminarmente provocan que la misma quede en un segundo plano, lo que impide que se conozca, se potencie y se desarrolle.

1.2.2.2. Configuraciones

Como se mencionó anteriormente, los proyectos, aplicaciones, módulos de Ksike y el propio *framework*, son altamente configurables. Esta característica le proporciona un plus al marco de trabajo, pues al ser tan flexible puede ser adaptado a un gran número de escenarios sobre los cuales pueda extenderse eficientemente. Para garantizar esta característica: la flexibilidad, Ksike maneja numerosos parámetros de configuración que encapsula en dos grupos principales: enrutamiento (***router***) y carga (***loader***).

El enrutamiento puede estar configurado según espacios de nombre específicos como son: ***app***, ***plugins, lib, web*** y ***request***; que están asociados a la ruta física de la aplicación, los módulos del proyecto, bibliotecas y componentes externos así como la URL utilizada sobre la web respectivamente, según se muestra a continuación.

```
$config["router"]["app"]     = 'app/';
$config["router"]["plugins"]     = 'plugins/';
$config["router"]["lib"]     = 'lib/';
$config["router"]["web"]     = '';
```

Figura 1.2.5: Configuración de enrutamiento por espacios de nombre.

El espacio de nombre ***request***, constituye otra parte del enrutamiento que se realiza según las peticiones que se hagan al controlador frontal del *framework*. Esto quiere decir que cuando se construye una petición desde el lado cliente de un módulo determinado, el controlador frontal de Ksike necesita direccionar la petición exactamente hasta el proyecto, módulo y acción correspondiente.

Agréguese además que es necesario definir qué parámetros se envían en la petición y en qué formato se recibirá la respuesta. Las variables de configuración de enrutamiento se subdividen en dos categorías: claves (***key***) y alias. Concretamente las claves configurables son: ***proj, controller, action, params, outFormat, outInfop, OutOption*** y ***pattern,*** como se ilustra en la Figura 1.2.6.

```
er"]["request"]["key"]["proj"]          = false;
er"]["request"]["key"]["controller"]    = 'Main';
er"]["request"]["key"]["action"]        = 'index';
er"]["request"]["key"]["params"]        = '';
er"]["request"]["key"]["outFormat"]     = 'html';
er"]["request"]["key"]["outInfo"]       = 'void';
er"]["request"]["key"]["outOption"]     = 'none';
er"]["request"]["url"]["pattern"]       = 'controller/action/params/outFormat/outInfo/outOption';
```

Figura 1.2.6: Claves configurables para el enrutamiento de peticiones al controlador frontal de Ksike.

En cambio los 'alias' son: ***web*** y ***publisher***, con disímiles valores configurables. Estos a su vez, engloban otros muchos nomencladores de parámetros de configuración como son: ***uri, key*** y muchos otros que pueden divisarse en la Figura 1.2.7 a continuación.

```
16  $config["router"]["app"]     = 'app/';
17  $config["router"]["plugins"]= 'plugins/';
18  $config["router"]["lib"]     = 'lib/';
19  $config["router"]["web"]     = '';
20  $config["router"]["request"]["key"]["proj"]          = false;
21  $config["router"]["request"]["key"]["controller"]    = 'Main';
22  $config["router"]["request"]["key"]["action"]        = 'index';
23  $config["router"]["request"]["key"]["params"]        = '';
24  $config["router"]["request"]["key"]["outFormat"]     = 'html';
25  $config["router"]["request"]["key"]["outInfo"]       = 'void';
26  $config["router"]["request"]["key"]["outOption"]     = 'none';
27  $config["router"]["request"]["url"]["pattern"]       = 'controller/action/params/outFormat/outInfo/outOption';
28
29  $config["router"]["request"]["alias"]["web"]["url"]["pattern"]       = 'proj/controller/action/params/outFormat/outIn
30  $config["router"]["request"]["alias"]["web"]["key"]["controller"]    = 'Main';
31  $config["router"]["request"]["alias"]["web"]["key"]["action"]        = 'index';
32  $config["router"]["request"]["alias"]["web"]["key"]["outFormat"]     = 'js';
33  $config["router"]["request"]["alias"]["web"]["key"]["outInfo"]       = 'void';
34  $config["router"]["request"]["alias"]["web"]["key"]["outOption"]     = 'none';
35  $config["router"]["request"]["alias"]["web"]["key"]["proj"]          = 'atipic';
36
37  $config["router"]["request"]["alias"]["publisher"]["url"]["pattern"]       = 'params:type/outFormat';
38  $config["router"]["request"]["alias"]["publisher"]["key"]["controller"]    = 'Main';
39  $config["router"]["request"]["alias"]["publisher"]["key"]["action"]        = 'contents';
40  $config["router"]["request"]["alias"]["publisher"]["key"]["outFormat"]     = 'js';
41  $config["router"]["request"]["alias"]["publisher"]["key"]["outInfo"]       = 'void';
42  $config["router"]["request"]["alias"]["publisher"]["key"]["outOption"]     = 'none';
```

Figura 1.2.7: Fichero de configuración del servidor de Ksike (config.php).

Hasta este momento se han analizado las variables de configuración asociadas al enrutamiento, pero no pueden dejarse de analizar las de carga, quienes juegan un rol importante dentro de las funcionalidades de Ksike. Estas definen si el sistema carga automáticamente o de forma manual los elementos primarios (***base***), filtros (***filter***) y paquetes (***package***), estos sostienen toda la jerarquía de componentes funcionales del *framework*. Los filtros pueden ser de: interfaces (***iface***), control, recurso (***resource***) y decoración (***decorate***). Por otro lado los paquetes son más diversos y pueden ser de: configuración (***config***), mensajes de sucesos (***log***), sesiones (***session***), direccionamiento (***dir***), salida (***out***), inversión de control (***ioc***), controladores frontales (***front***) y errores (***error***).

La versión Ksike 1.1 Elephant brinda soporte para la construcción de aplicaciones web, lo que significa que posibilita el control sobre recursos para dos entornos distintos de ejecución: el cliente y el servidor. Esto implica que, aunque los conceptos que maneja el *framework* sobre ambos lados son muy similares, cada uno posee sus particularidades y además sus propias configuraciones, para servidor y cliente respectivamente.

En factores de configuración del lado servidor, Ksike posee más de cien parámetros distintos de configuración los que se gestionan a través del fichero **config.php**, ubicado en el directorio **core/cfg/** del núcleo del *framework*. En la misma localización se encuentra el fichero **config.js**, que es empleado para gestionar las configuraciones del cliente. Los parámetros de configuración son muy similares en aras de disminuir la complejidad conceptual del marco de trabajo, pero también tiene sus peculiaridades.

En este se establecen: espacio de nombre global del proyecto (**ns**), enrutamiento (***router***) y carga (***loader***). Este último engloba la configuración de tipo de carga (***loadType***) que puede ser sincrónica o asincrónica. También el grupo de las bases (***base***) incluye los patrones (***patterns***), asistentes (***helpers***) y manejadores (***handlers***). También agrupa los filtros (***filter***) y paquetes (***package***), elementos que tienen similares configuraciones con respecto a las del servidor pero comportamientos específicos para el cliente. Esto es debido a que concretamente, este último, constituye un entorno de ejecución diferente. Los elementos analizados le suman una veintena de variables de configuración a las más de cien anteriormente mencionadas para el servidor.

Esta enorme cantidad de elementos de configuración resulta ser uno de los mayores tabúes que presenta la tecnología, pues aquí reside el verdadero reto de los desarrolladores, quienes deben ser más que especialistas para introducirse en dichos ficheros y aventurarse a configurar manualmente los mismos, cambiando o manipulando información que puede resultar crucial para el funcionamiento óptimo de Ksike. Súmese además el hecho de que Ksike proporciona diversidad de formatos para la manipulación de la información asociada a las configuraciones (JSON, XML, PHP, JS, INI) los que tienen sus peculiaridades según se ilustra en la Figura 1.2.8.

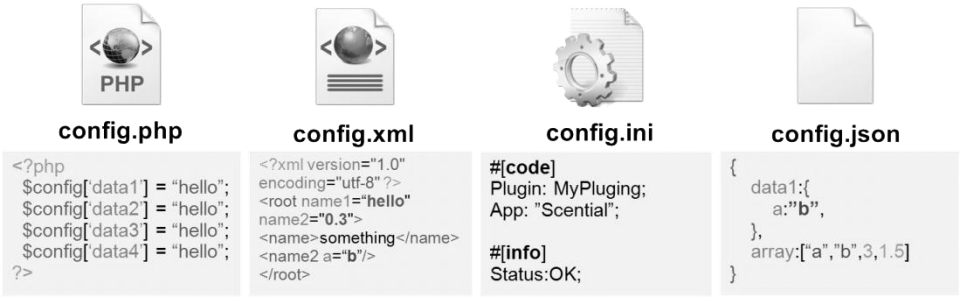


Figura 1.2.8: Formatos de archivos de configuración soportados por Ksike.

Concretamente el *framework* posee capacidad para el manejo de configuraciones para estos cinco tipos distintos de formatos. Debe conocerse que sus más de ciento treinta parámetros de configuración pueden estar en cualquiera o incluso en varios formatos al mismo tiempo de forma seccionada. A este punto, es común que crezcan en los desarrolladores varios factores de incertidumbre teniéndose que preguntar en ocasiones ¿Cuál uso? ¿Cómo lo uso?, temas que pueden ser mitigados haciendo uso de alguna herramienta que las gestione.

Una vez más, para evitar estos inconvenientes, es lógico que los desarrolladores echen a un lado a Ksike y utilicen otras tecnologías que faciliten la configuración ya sea a través de interfaces visuales o

mediante una CLI[21]. Lo cierto es que Ksike, que es un producto con múltiples ventajas y posibilidades, creado en la UCI, que además actualmente está en desarrollo, resulta estar en desuso y no se prueba ni se le da el correcto soporte.

1.2.2.3. Sistema de clases para JavaScript

El paradigma de POO *"(…) permite realizar grandes programas mediante la unión de elementos más simples, que pueden ser diseñados y comprobados de manera independiente del programa que va a usarlos. Muchos de estos elementos podrán ser reutilizados en otros programas (…) A estas 'piezas', 'módulos' o 'componentes', que interactúan entre sí cuando se ejecuta un programa, se les denomina objetos"* (García de Jalón, y otros, Abril 1998).

La POO brinda una atractiva lista de conceptos, dígase: objeto, método, propiedad, clase, encapsulación, agregación, composición, reusabilidad/herencia y polimorfismo. Estos están implementados en varios lenguajes y posibilitan un sinfín de oportunidades para el desarrollo de sistemas consistentes (Stefanov, Julio 2008). Entre los lenguajes que implementan el paradigma están, C++, Java, C#, PHP, Python entre muchos otros.

JavaScript no es un lenguaje que clásicamente esté orientado a objetos, sino un lenguaje de prototipado[22]. Esto significa que, a diferencia de los lenguajes clásicos, en los que se puede crear un objeto llamado "Pedro" de la clase "Persona", en los lenguajes prototipados orientados a objetos, se emplea un objeto existente "Persona" que es reutilizado como un prototipo para la creación de un nuevo objeto denominado "Pedro" (Stefanov, Julio 2008). En la Figura 1.2. se muestra un ejemplo de implementación de una clase "Persona" en los lenguajes Java, PHP y JavaScript respectivamente. Éste último está implementado según el Esquema de Clases de Ksike.

```java
class Persona
extends Object
implements IComparable
{
    private String nombre = "desconocido";

    public void Persona (String nombre) {
        this.nombre = nombre;
    }

    public String getNombre(){
        return this.nombre;
    }
}
```

Java

```php
<?php
    class Persona
    extends Object
    implements IComparable
    {
        private $nombre = "desconocido";

        public function Persona($nombre)
        {
            $this->nombre = $nombre;
        }

        public function getNombre(){
            return $this->nombre;
        }
```

PHP

```ksikejs
Kcl.class('Persona',
{
    extend: Object,
    implement: IComparable,
    properties : {
        nombre: "desconocido"
    },
    behavior: {
        construct : function(nombre)
            this.nombre = nombre;
        },
        getNombre : function(){
            return this.nombre;
        }
    }
});
```

KsikeJS

Figura 1.2.9: Clase 'Persona' implementada en varios lenguajes (KsikeJS significa Esquema de clases JS de Ksike, implementado en JavaScript).

En aras de abstraer a los desarrolladores de los conceptos del prototipado y atraerlos al paradigma clásico de POO, Ksike brinda un esquema propio de Programación Orientada a Objetos para JavaScript, que incluye los mismos conceptos vistos anteriormente pero se codifica de una manera particular. Esta aunque es muy similar a la brindada por los lenguajes clásicos, necesita ser conocida

[21] **CLI:** Siglas de *Command Line Interface*, en español Interfaz de Líneas de Comando.

[22] **prototipado** Está orientado a prototipos, que son objetos existentes.

previamente por los desarrolladores. Actualmente las clases de Ksike para JavaScript se crean y codifican de forma manual. Comenzarlas a codificar desde cero no es muy complicado pero pudiera ser un factor sobre el cual se cometen errores. Comprender este esquema y sus particularidades requiere cierto nivel de experiencia.

1.2.2.4. Enlazador de funciones

Uno de los elementos más novedosos dentro del conjunto que brinda el marco de trabajo Ksike es el recurso Linker o Enlazador de funciones. El mismo está inspirado en el Señales y Ranuras (*Signals & Slots*) del *Framework* Qt, que constituye una implementación del patrón observador-observado. Su funcionalidad principal consiste en enlazar, como bien lo dice el nombre, eventos o acciones entre módulos. Esto significa que cuando en un módulo A se ejecuta la función “a ()”, en correspondencia, el módulo B disparará la acción “b ()”, cómo se muestra en la Figura 1.2.8.

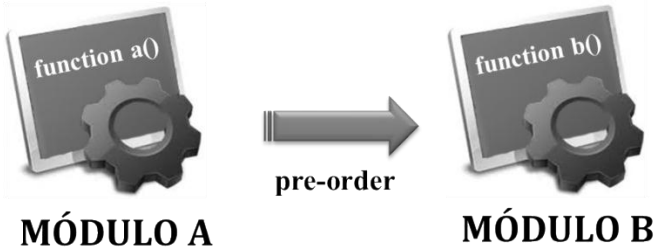


Figura 1.2.8: Representación gráfica del funcionamiento del Recurso Linker de Ksike.

Los modos de ejecución pueden ser: ***pre-order*** o ***post-order***. El primero significa que la función “b ()” se ejecutará antes de que se lleve a cabo la acción “a ()”, justo después de su invocación. El *post-order* representa el flujo normal de los eventos, en el cual la acción “b ()” depende circunstancialmente de la ejecución de “a ()”. Si las funciones no están emparentadas bajo ningún enlazador estas pueden ejecutarse normalmente. El Linker resulta ser un elemento novedoso para los sistemas desarrollados sobre la web, es por esto que hereda las características y conceptos de un marco de trabajo desarrollado para entornos de escritorio, Qt. En este marco de trabajo, para hacer uso del recurso señales y ranuras, existen algunas reglas fundamentales que se analizan a continuación.

Toda clase que herede de la **QObject** puede tener tantas señales como se desee aunque solo se puede emitir una señal desde la clase donde esta se encuentra implementada. También es posible conectar una señal con otra, amén de crear una cadena de señales. Otro elemento a tener en cuenta es que cada una de estas señales y ranuras pueden tener conexiones ilimitadas entre sí (Qt Project, 2012). Los términos “señal” y “ranura” no son más que nomencladores que se utilizan para aterrizar el concepto de enlace. La señal es la función disparadora desde la que parte el flujo de eventos. Las ranuras en cambio son todas aquellas que captan la señal y actúan en consecuencia. Como los conceptos empleados en los *frameworks* Qt y Ksike son similares, es evidente que su empleo también comparta cierta similitud.

Framework Qt

```
QObject::connect ( const QObject * sender, const char * signal, const QObject * receiver, const char * method);
```

Framework Ksike

```
$this->linker->connect("accion_a","Modulo_A","accion_b","Modulo_B",'post', null);
```

Figura 1.2.9: Implementación de los recursos "Signals & Slots" del *framework* Qt y "Linker" de Ksike.

Como puede evidenciarse en la Figura 1.2.9, no solo los conceptos son análogos en ambas tecnologías, sino también sus implementaciones. Su empleo en Ksike es muy intuitivo para aquellos desarrolladores experimentados en el *framework* Qt que deciden tomar partida en el desarrollo de aplicaciones web, aunque vale destacar que restan un par de elementos nuevos en cuanto a la forma de uso. A diferencia de Qt, en Ksike, el concepto se maneja a macro escala, lo que significa que este está dirigido a los módulos de un proyecto o aplicación y no a nivel de objeto. Teniendo en cuenta lo anterior, la implementación, tal y como se representa en la **¡Error! No se encuentra el origen de la eferencia.**1, se define en el método constructor de la clase servidora del módulo de aplicación, **Main.php**, quien tiene conocimiento de la existencia de los otros módulos, tal y como se explicó anteriormente.

La otra manera de usar los enlaces en Ksike es haciendo uso de ficheros de configuración, como se muestra en la Figura 1.2.12. Dichas configuraciones son asimiladas por el módulo aplicación al construirse, permitiendo la existencia de los enlaces una vez que esté cargado el sistema y garantizando además su correcto funcionamiento. El inconveniente con estas, básicamente es el mismo que el analizado en el sub epígrafe correspondiente a las configuraciones de Ksike, es engorroso manejar de forma manual los datos que están escritos sobre un archivo.

```
php
  $config["linker"]["pos"]["Navigation"]["zoomIn"][]  = array( "handle" => 0, "slot" => "update", "class" => "Map");
  $config["linker"]["pos"]["Navigation"]["zoomOut"][] = array( "handle" => 0, "slot" => "update", "class" => "Map");
  return $config;
```

Figura 1.2.12: Configuraciones asociadas al Enlazador de funciones de Ksike.

¿Qué sucedería si en una aplicación existen 50 enlaces? Evidentemente darle el correcto mantenimiento a un fichero de configuración de enlaces con tal cantidad de configuraciones puede ser agotador. La gestión de los enlaces puede resultar compleja, pues su definición a través de configuraciones o la implementación manual de estos dentro del módulo de principal del sistema, tendrían el mismo resultado, trabajo manual en exceso. Esto además puede ser un punto clave para cometer errores que provoquen fallas en el sistema implementado.

Si ya no es necesario el enlace hay que cerciorarse de eliminar su configuración o declaración pues, si no se trata correctamente, el sistema puede continuar lanzando los eventos que se han definido, pudiendo concretar resultados erróneos o incluso haciendo explotar la aplicación. Estos elementos sin lugar a dudas constituyen otra causa que se suma al conjunto mencionado anteriormente y que conlleva a efectos negativos en cuanto al empleo de la tecnología Ksike.

1.2.2.5. Publicaciones con Ksike

Una de las características elementales a tener en cuenta cuando se desarrollan aplicaciones informáticas sobre los entornos web es la seguridad. Esta tiene como objetivo garantizar la protección de la información, como recurso más valioso, bajo tres principios fundamentales: confidencialidad, integridad y disponibilidad. El primero hace referencia a los niveles de acceso a la información, garantizando que esta sea accesible solamente por el personal debidamente autorizado. El segundo principio, denota los permisos y operaciones que se deben realizar sobre determinada información, o sea que la información sea accedida por las personas autorizadas y de la forma autorizada. El último de los tres principios es el de disponibilidad, que se refiere a que los activos informáticos sean accedidos por las personas autorizadas en el momento requerido (Pfleeger, 2006).

Con el fin de garantizar la seguridad de los archivos de los proyectos desarrollados, varios *frameworks* como: Symfony brindan algunas estrategias. Estos recomiendan tener dentro de los proyectos un único directorio público: ***web/***. En él deben ubicarse solamente los archivos que necesitan los navegadores, tales como: hojas de estilos, los archivos JavaScript, las imágenes y los controladores frontales de las aplicaciones. Luego se delega la responsabilidad al servidor de aplicaciones a través de la configuración de un *host* virtual a este directorio, de manera tal que los usuarios solo acceden al contenido público restringiendo el acceso al resto de los directorios (Potencier, 2009) (Eguiluz, 2010).

Symfony2, al estar arquitectónicamente orientado al desarrollo de componentes, proporciona bajo la envoltura de cada *bundle*[23]*,* un directorio Resources/public/ en el cual se almacenan los datos que son públicos para los navegadores. Aunque como se mencionó anteriormente existe un solo directorio público en todo el proyecto (web), este marco de trabajo recomienda que los *bundles* de sus aplicaciones incluyan todos los archivos CSS, JavaScript e imágenes que con ellos se relacionan en el directorio *public* (Eguiluz, 2010). Estos archivos luego se copian o enlazan simbólicamente con el directorio público web, mediante la Interfaz de Línea de Comandos que provee el propio Symfony2, haciendo uso del comando:

```
$ php app/console assets:install
```

El mismo se encarga de automatizar el proceso de instalación de los archivos públicos de cada *bundle* en el directorio web. Anteriormente en el sub-epígrafe dedicado a las configuraciones de Ksike, se introdujo el término ***publisher*** que hacía referencia a las configuraciones iniciales del mecanismo de publicación del marco de trabajo. A diferencia de otros *frameworks*, Ksike posee un mecanismo de publicaciones que proporciona tres modos de publicación de los proyectos. Estos garantizan tres niveles de seguridad como se puede apreciar en la Tabla 1.2.1 a continuación y sobre la cual se dará una explicación seguidamente.

[23] **Bundle** Directorio a manera de módulo de aplicación, que contiene todo tipo de archivos dentro una estructura jerarquizada de directorios (Eguiluz, 2010)**.**

Publicación	Archivos públicos	Seguridad	Enrutamiento	Rendimiento
Total	Ksike + Proyecto	Mínima	Directo	Alto
Parcial	CAP[24] + Parte cliente	Parcial	Parcial	Medio
Limitada	CAP	Total	Indirecto	Bajo

Tabla 1.2.1: Relación de los modos de publicación de Ksike con los niveles de seguridad, los modos de enrutamientos y el rendimiento del sistema.

El modo de **publicación total** significa que todos los ficheros de los proyectos de Ksike serán públicos para los todos los usuarios. Esto garantiza, como se ha de suponer, el **mínimo de seguridad**, pues el código fuente, ficheros de configuración y demás estarán disponibles para todos los que accedan a la aplicación. Evaluando en términos de enrutamiento, puede decirse que, al ser totalmente público un proyecto, se gestiona el **enrutamiento directamente** por parte del servidor de aplicaciones, garantizando que las respuestas a las peticiones de los clientes, no consuman recursos extra del sistema, proveyendo un **elevado rendimiento** del mismo.

El modo de **publicación parcial** es muy similar a las recomendaciones propuestas por Symfony y Symfony2, en las que solo se ubica en un directorio la información que debe ser pública a los navegadores, o sea, todo el contenido de los directorios *client* analizados anteriormente y el PAC del proyecto. Este último actúa como intermediario entre las peticiones generadas en el proyecto y el controlador frontal de Ksike. De esta manera se protege el acceso al resto de la información del proyecto, garantizando un nivel de **seguridad parcial**. El **enrutamiento** ocurre también de forma **parcial** pues aún deben construirse peticiones para garantizar el acceso a otro tipo de información que quiera mostrarse a los clientes. Dichas peticiones requieren para su construcción y procesamiento varias operaciones por parte del intérprete de PHP por lo que causa que el **rendimiento** del sistema sea un poco menos eficiente, otorgándole una categoría media en comparación a la resultante de los otros dos modos de publicación.

Por último, el modo de **publicación limitada** sólo requiere que se haga público el PAC del proyecto pues este se encargará de gestionar, en conjunto con el controlador frontal de Ksike, las peticiones necesarias para acceder a todos los demás recursos del proyecto. Todos los recursos del cliente, por ejemplo: css, js e imágenes, serán obtenidos mediante la formulación de peticiones al controlador frontal de Ksike. Este se encargará de realizar una búsqueda dentro de una tabla *hash* donde se encuentran almacenadas las direcciones físicas de los recursos y luego dará respuesta a la petición. El resultado será almacenado en un archivo de caché para agilizar eventualmente la respuesta ante peticiones similares sobre un mismo recurso.
De esta forma se garantiza una **seguridad total**, pues los usuarios jamás tendrán conocimiento de la estructura física del proyecto, ni de la ubicación de los componentes del mismo, lográndose proteger la información. Como se describió anteriormente, el proceso se realiza mediante peticiones realizadas al

[24] **CAP** *Common Access Point.* Punto de Acceso Común (PAC).

controlador frontal de Ksike, por lo que puede notarse el **carácter indirecto** que posee el **enrutamiento** pues se le substrae esta tarea al servidor de aplicaciones, delegándole toda la responsabilidad al *framework*. Este modo requiere de mucho procesamiento por parte del intérprete de PHP, por lo que es el de más **bajo rendimiento** entre todos los modos propuestos por el mecanismo de publicación.

A los mecanismos de seguridad que proporciona Ksike se le suma además el mencionado anteriormente, correspondiente a la configuración de un host virtual, aumentándole la seguridad a los sistemas desarrollados. La situación problemática que existe con respecto a las publicaciones es en cuanto a las operaciones de desarrollo o mantenimiento que se realicen en los proyectos. Cada vez que se realicen cambios estructurales, sean agregados/eliminados recursos o se modifique el código fuente, es necesario actualizar de forma manual el contenido que será publicado, ya sea todo el proyecto, la parte de los clientes de todos los módulos o la tabla hash asociada al proyecto en cuestión, de acuerdo a cada modo de publicación analizado anteriormente.

A diferencia de Symfony, Ksike no cuenta con un CLI y una serie de comandos que automatizan sus procesos, por lo que es una realidad que su empleo para en el desarrollo de software no haga competencia a los primeros ante la preferencia de los desarrolladores. Evidentemente, haciendo uso de las herramientas que proporcionan los mencionados *frameworks* se agilizan los procesos de desarrollo, obtención y mantenimiento de software, quedando Ksike como un buen conjunto de teorías implementadas pero sin uso. Aunque este marco de trabajo posee un carácter académico, sus principios y recursos son novedosos pudiendo competir incluso con otros de su tipo, pero si no se utiliza el mismo por supuesto no se potencia ni se desarrollan sus conceptos, tampoco se mejora ante posibles errores que pueda presentar en su implementación.

1.2.2.6. Factores subjetivos

A grandes rasgos han sido analizadas varias situaciones problemáticas asociadas al empleo de Ksike en el desarrollo de software. Objetivamente existen un sinnúmero de procesos del marco de trabajo que contribuyen a disminuir psicológicamente la preferencia de los desarrolladores a emplearlo para construir sus sistemas. Actualmente el mismo no posee herramientas ni algún otro recurso que garantice automatizaciones para sus procesos más importantes, como son: gestión de proyectos, gestión de plantillas, manejo de dependencias, manejo de componentes externos, gestión de *logs* y tratamiento de errores.

A estos elementos hay que sumarle el hecho de que, por ser un conjunto de tecnologías recientes, no poseen la difusión en la comunidad ni documentación de desarrollo adecuado, por lo que la información que se asocia o explica sus principios y filosofías es bastante limitada. Actualmente cuenta solo con un manual de desarrollo para la versión 1.0 que se encuentra desactualizado. Quiere esto decir que, aunque los conceptos no han variado demasiado, su implementación sí en la mayoría de los casos, por lo que los ejemplos que aparecen ilustrados o explicados no son del todo funcionales,

debido a cambios que ha tenido la arquitectura e implementación del *framework*. Cabe aclarar que estos elementos han sido modificados a favor de garantizar la madurez evolutiva que ha alcanzado el marco de trabajo. Además se han elaborado dos artículos científicos que abordan escuetamente algunos elementos del funcionamiento de Ksike.

Otra de las situaciones existentes es que el mismo no ha sido lo suficientemente probado por los desarrolladores. Esto impide, en primer lugar que existan especialistas sobre el empleo de Ksike y en segundo lugar que actualmente no sean conocidas las verdaderas potencialidades del *framework.* Como otra vertiente asociada puede definirse que no existe la adecuada ejemplificación de su empleo, excepto por Geotrygon, aplicación informática que constituye un SIG implementado completamente sobre la arquitectura y recursos de Ksike (Geotrygon, desarrollo de SIG sobre el framework Ksike, 2012), y a lo sumo otros tres ejemplos muy básicos, orientados a demostrar el empleo de funciones específicas del marco de trabajo. Estos ejemplos son parte de la API que brinda el paquete de desarrollo y constituyen proyectos de prueba.

Ksike propone una nueva forma de enfocarse al desarrollo de aplicaciones para el entorno web, proponiendo un regreso de las filosofías de su precursor, el *desktop*. Como es eventualmente conocido, el ser humano reacciona ante las situaciones novedosas generalmente haciendo resistencia al cambio y Ksike no queda enajenado a este fenómeno. Debido a la falta de simpatía que crea el conjunto de principios propuestos por el mismo, se puede evidenciar un efecto negativo en cuanto a que las comunidades de desarrollo y los proyectos productivos de la UCI no emplean ni conocen el *framework* (Membrides Espinosa, 2010).

Se hace rechazo a la asimilación de esta nueva tecnología debido a que se necesitan estudiar muchos elementos para garantizar su correcta utilización por los desarrolladores. Esto deviene en que el desarrollo de aplicaciones sobre Ksike sea más lento o costoso que su contraparte haciendo uso de otras tecnologías que, sin ser más completas, han desarrollado algunas herramientas para automatizar sus procesos. Consecuentemente el mismo no se potencia ni se desarrolla, siendo una situación decadente avistada por su equipo de desarrollo, quienes apuestan a favor del uso de la tecnología por sus disímiles potencialidades y el carácter soberano que posee.

1.2.3. ¿Por qué una herramienta para Ksike?

Ksike es un marco de trabajo de código abierto y su desarrollo provee un ambiente propicio para la contribución de la comunidad, con estabilidad en el resultado. Al ser una solución surgida en la UCI posee también gran facilidad para el soporte, además el marco de trabajo es gratuito y está apto para ser empleado en el desarrollo de soluciones de software personalizadas. Siendo un independiente, permite que sea explotado a conveniencia, librando a sus empleadores del pago de licencias a agentes externos. Debido a que Ksike está liberado bajo licencia LGPL y es gratuito, todas las herramientas desarrolladas sobre dicha tecnología pueden ser fácilmente obtenidas. Claro está el hecho de que estas permiten ser usadas para crear proyectos comerciales. Y, obviamente, si se decide que no es la

tecnología más adecuada para los proyectos en desarrollo, no tiene impacto significativo, pues adquirirla está libre de impuestos y no cuesta absolutamente nada.

Además que tiene como meta desarrollar aplicaciones nativas para entornos móviles, aunque actualmente permite el desarrollo de aplicaciones web con interfaces para este tipo de dispositivos. Al ser desarrollado por profesionales cubanos que radican en la UCI, es posible crear una comunidad online que ante cualquier problema o duda siempre esté bien informada y dispuesta a ayudar. También es de validez analizar que los lenguajes de programación empleados para el desarrollo sobre este son muy conocidos, por lo que no debería resultar extra-difícil utilizarlo.

Construir un sistema de herramientas y editores visuales para los recursos de Ksike permite trabajar con más comodidad y fluidez a los desarrolladores, lo que posibilita asimilar la tecnología con mayor facilidad e intuición. Además está el hecho de que, usar un IDE completamente orientado a una tecnología en específico, que combine edición, depuración, gestión de proyectos, localización y herramientas de compilación o interpretación, brinda una garantía absoluta sobre el empleo de dicha tecnología. Tal es el caso de las tecnologías Qt, que aunque existen desde el año 1992 del pasado siglo, fueron conocidas y empleadas masivamente una vez que, diecisiete años más tarde, en el 2009, se creó el IDE Qt Creator (Ziller, 2009). Este es un entorno integrado que está dedicado específicamente al desarrollo sobre la tecnología Qt, aunque actualmente puede ser extendido y aplicado en varias tecnologías e incluye metas mucho más ambiciosas.

Basado en estos principios se decide enfocar la investigación sobre el objetivo de construir una herramienta que permita agilizar los procesos de obtención de soluciones sobre el *framework* Ksike. Dicha herramienta será bautizada a decisión del equipo desarrollo del marco de trabajo con el nombre de BHike, para continuar con las nomenclaturas de los productos que dicho equipo desarrolla.

1.2.4. Descripción general de los IDE

Los IDE están diseñados para maximizar la productividad de los programadores mediante el uso de una interfaz amigable e intuitiva que permite el empleo de herramientas, *frameworks* así como bibliotecas de clases y componentes. Estos elementos contribuyen a reducir la curva de aprendizaje y limitan los costes en tiempo para el desarrollo de sistemas informáticos. Son sin dudas herramientas altamente calificadas para la obtención de soluciones de software, aunque se han extendido empíricamente abarcando múltiples escenarios. Estos posibilitan un marco más amplio de funcionalidades que posibilitan darle seguimiento a funciones básicas de los sistemas operativos. En este acápite se pretende abordar algunos de los elementos fundamentales de los IDE.

1.2.4.1. Composición de los IDE

Muchos entornos de desarrollo modernos poseen, además de las partes antes mencionadas, un navegador de clases, un inspector de objetos y un diseñador de diagramas de jerarquía de clases, normalmente todo ello para su uso con el desarrollo de software orientado a objetos. También están

integrados por un sistema de control de versiones y varias herramientas para simplificar la construcción de una GUI. El límite en cuanto al conjunto de herramientas que puede o no poseer un entorno de desarrollo no está definido, por lo que, a modo de acotar el contenido a tratar en este documento, a continuación se explica las fundamentales. Para ello se emplea el esquema definido por el autor Carlos Blanco quien propone como partes fundamentales: editor de código fuente, compilador y/o intérprete, depurador y automatización de procesos.

Editor de código fuente

Carlos Blanco define que un editor de código fuente: *"(…) es un editor de texto del programa diseñado específicamente para la edición de código fuente de programas informáticos por los programadores. Puede ser una aplicación independiente o normalmente suele estar integrado (…)"* (Blanco, 2011). Estos están diseñados con características específicas que permitan simplificar y acelerar la entrada del código fuente, como son: el resaltado de sintaxis, que ocurre mientras se programa en tiempo real, avisando de inmediato de los problemas de sintaxis y el auto-completamiento. Los editores brindan también un conjunto de funcionalidades de acceso, que permiten ejecutar a un compilador, intérprete, depurador u otra herramienta que sea útil en el proceso de obtención de software (Blanco, 2011).

Compilador y/o Intérprete

Un compilador es un programa de computadora que transforma código fuente legible, comprensible por los desarrolladores, en código máquina de forma tal que pueda ser ejecutado por el CPU[25]. A este proceso se le denomina **compilación** (Bolton). También puede ser entendido como el proceso de transformación de código de un lenguaje de programación de origen (código fuente) a un lenguaje de destino, éste último que a menudo es codificado en binario, es conocido como lenguaje objeto. La razón más común por la que se realiza la compilación es para crear un ejecutable del programa escrito (Hernández, 2006).

Un intérprete es un programa informático capaz de analizar y ejecutar otros programas (Bolton). Los intérpretes difieren de los compiladores en que, estos últimos, traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema (Blanco, 2011), generalmente producen un binario ejecutable, que puede utilizarse cuando sea requerido. Por otro lado los intérpretes realizan el proceso de compilación o sea: análisis léxico, sintáctico, semántico y generación del código intermedio y luego ejecutan el resultado directamente. La interpretación se realiza a medida que sea necesaria, típicamente, instrucción por instrucción y normalmente no guardan el resultado de del proceso.

[25] **CPU** Siglas del inglés *Central Process Unity* (Unidad Central de Procesos).

Depurador

Un depurador (*debugger*) es un programa usado para probar y depurar, o sea detectar los errores de otros programas, denominados "programas objetivo". El código al ser examinado puede alternativamente estar corriendo en un ISS[26], una técnica que permite gran potencia en su capacidad de detenerse cuando son encontradas condiciones específicas, pero será típicamente algo más lento que ejecutando el código directamente. Algunos depuradores ofrecen dos modos de operación: la simulación parcial y la completa; para limitar este impacto.

Típicamente, los depuradores también ofrecen funciones más sofisticadas tales como ejecutar un programa paso a paso, detener la ejecución de un programa para examinar el estado actual en cierto evento o instrucción especificada por medio de un *breakpoint*, y el seguimiento de valores de algunas variables. Algunos depuradores tienen la capacidad de modificar el estado del programa mientras que está corriendo, en vez de simplemente observarlo.

Automatización de procesos

La automatización de manera dinámica es el acto de ejecutar secuencias de comandos o de generar una gran variedad de tareas que los desarrolladores de software hacen a diario, gracias a las características y herramientas que ofrece un IDE; incluyendo procesos como (Blanco, 2011):

- ✓ Compilación de código fuente a código binario.
- ✓ Embalaje del código binario.
- ✓ Pruebas de funcionamiento.
- ✓ Implementación de sistemas de producción.
- ✓ La creación de documentación y / o notas de la versión.

Todos estos elementos son de vital importancia para el desarrollo de herramientas versátiles y de múltiples aplicaciones como son los IDE, pero vale la pena destacar que entre los aspectos más importantes reside la implementación de los sistemas de producción, que proporcionan una estructura que agiliza la descripción, ejecución y el planteamiento de un proceso industrial. Esto quiere decir que son los que posibilitan el desarrollo de soluciones basado en los esquemas de proyectos, módulos, aplicaciones y otros, como los que residen en varios IDE como: NetBeans, Eclipse, Visual Studio, etc.

1.3. Conclusiones parciales

Los elementos mencionados en el acápite Situación Problemática incluyen ciertos niveles de complejidad conceptual, evidenciando que Ksike posee varios procesos que al realizarse de forma manual entorpecen el desarrollo de software. La automatización de procesos es un elemento fundamental de los IDE, por cuanto resulta viable desarrollar mecanismos que agilicen y faciliten la concepción de soluciones sobre el *framework*. Se evidencia además la necesidad de facilitar la asimilación del mismo en aras de otorgarle utilidad a la tecnología, pues actualmente está en desuso.

[26] **ISS** Siglas de *Instructions Simulation System,* en español Sistema de Simulación de Instrucciones.

CAPÍTULO 2: TENDENCIAS Y TECNOLOGÍAS

En este capítulo se realiza el análisis de las principales tendencias o soluciones desarrolladas, haciendo una descripción exhaustiva de las características principales de los IDE actualmente más populares y con mayores prestaciones. Además se estarán sintetizando de manera preliminar las principales herramientas, tecnologías y elementos metodológicos a utilizar durante todo el proceso de elaboración de la solución que se obtendrá al concluir la investigación.

2.1. Tendencias y alternativas

De acuerdo con mencionado en el Capítulo 1 acerca de las herramientas para la automatización de procesos de desarrollo de software, se han construido diversos IDE orientados a la concepción de aplicaciones para todo tipo de arquitecturas, plataformas o sistemas informáticos que responden a las necesidades de los desarrolladores para los ambientes *web* y *desktop*. Algunos de estos con el objetivo de lograr preferencia sobre el uso de una tecnología en particular, como es el caso de Qt-Creator, pero todos con el objetivo de garantizar rapidez y eficiencia en el desarrollo de software.

Las soluciones más elaboradas constituyen aplicaciones de entornos de escritorio, aunque existen tendencias a desarrollar herramientas web debido a las grandes ventajas que posee este tipo de entorno. Estas herramientas son denominadas WIDE[27] de acuerdo al medio donde se ejecutan: la web. En aras de localizar potencialidades, patrones, técnicas y facilidades que poseen los IDE y que puedan ser oportunamente empleados en el desarrollo de software sobre la plataforma Ksike, a continuación se exponen algunos de los más conocidos. Para mayor entendimiento se tendrán en cuenta las ventajas que presentan así como su nivel de aceptación por las comunidades de desarrollo en las que son empleados.

2.1.1. Entornos Integrados de Desarrollo (IDE)

NetBeans IDE: Está desarrollado completamente en el lenguaje Java usando la plataforma NetBeans. Soporta el desarrollo de todo tipo de aplicación Java: J2SE[28], web, EJB[29] y aplicaciones de móviles. Entre sus principales características se encuentra un sistema de proyectos basado en Apache Ant, control de versiones y refactorización. Su modularidad le ha permitido la potenciación de extensiones para el desarrollo de aplicaciones sobre los lenguajes de programación: Java, C/C++, Python, PHP, JavaScript así como para las aplicaciones orientadas a servicios SOA[30], incluyendo herramientas de esquemas XML[31], un editor WSDL[32] y un editor BPEL[33] para servicios web. Es una aplicación

[27] **WIDE** Acrónimo de *Web-based Integrated Development Environment*.
[28] **J2SE** Acrónimo de *Java 2 Platform Standard Edition*.
[29] **EJB** Acrónimo de *Enterprise JavaBeans*. Conjunto de API estándar para el desarrollo de aplicaciones.
[30] **SOA** Acrónimo de *Service Oriented Application* (Aplicaciones Orientadas a Servicios).
[31] **XML** Acrónimo de *Extensible Markup Language* (Lenguaje Extensible de Maquetado).
[32] **WSDL** Siglas de *Web Service Description Language* (Lenguaje de Descripción de Servicios Web).
[33] **BPEL** Siglas de *Business Process Execution Language* (Lenguaje de Ejecución de Procesos de Negocio).

multiplataforma que permite la creación de extensiones personalizadas (Oracle Corporation & afiliates, 2011).

Eclipse: Es un entorno integrado de desarrollo de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-ligero" basadas en navegadores. Este entorno ofrece una plataforma que ha sido para desarrollar IDE como JDT[34] y además emplea módulos para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permitiéndole trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y sistema de gestión de base de datos. La arquitectura basada en *plugins* permite escribir cualquier extensión deseada, pudiendo ser por ejemplo la gestión de la configuración. Además se provee soporte para Java y control de versiones (CVS) en el SDK[35] de Eclipse (Object Technology International, Inc., 2003).

Microsoft Visual Studio: Es un entorno de desarrollo integrado para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic.NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Visual Studio permite a los desarrolladores implementar todo tipo de aplicaciones, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Posee potentes mecanismos para la gestión de proyectos y povisto un esquema de plantillas que proporciona códigos auxiliares de proyecto así como elementos reutilizable y personalizable que aceleran el proceso de desarrollo. Además integra al MSBuild[36] que constituye la nueva plataforma de generación de Microsoft. La misma es completamente transparente en cuanto a la forma en que procesa y genera software, lo que permite a los desarrolladores montar y generar productos en un entorno de laboratorio en el que Visual Studio no está instalado (Microsoft).

Qt Creator: Es un IDE multiplataforma para C++ que forma parte de Qt SDK. Incluye un depurador visual y un editor de Interfaces Gráficas de Usuario. Qt Creator usa el compilador de C++ de la colección de compiladores de GNU en distribuciones de GNU/Linux y BSD. En Windows puede usar los compiladores MinGW o MSVC que forman parte de la propia instalación del producto. Qt Creator provee soporte para desarrollar y ejecutar aplicaciones de Qt para entornos de escritorio: Windows, Linux, FreeBSD y MacOS, así como para dispositivos móviles: Symbian, Maemo y Meego (Nokia). Entre las características fundamentales de las tecnologías Qt, se encuentra que es multiplataforma y además integra el recurso S&S, novedoso de la tecnología. Los principios que se definen en el IDE

[34] **JDT** Siglas de *Java Development Toolkit* (Paquete de Herramientas de Desarrollo para Java)
[35] **SDK** Siglas de *Software Development Kit* (Paquete de Desarrollo de Software).
[36] **MSBuild** Acrónimo de *Microsoft Build Engine* (Motor de Construcción de Microsoft).

para S&S pueden ser empleados en la solución para el desarrollo de las funcionalidades asociadas al Linker de Ksike.

2.1.2. Entornos Integrados de Desarrollo Web (WIDE)

Cloud9 IDE: Es un proyecto de software libre iniciado por Ajax.org, desarrollado sobre tecnología NodeJS por consiguiente en el lenguaje JavaScript. Propone brindar las mejores características tanto de los IDE como editores de código fuente como: NetBeans, Eclipse y Textmate, quienes definen arquitectura basada en *plugins.* Su centro de atención se basa en potenciar el desarrollo de JavaScript por lo que posee un conjunto de estándares para la integración en el desarrollo de cliente y servidor. Entre las características principales se puede mencionar que posee un editor de código altamente personalizado con reconocimiento de sintaxis, con soporte para: JavaScript, HTML, CSS y modos mezclados de código. Otro elemento importante a destacar es que integra un depurador para aplicaciones sobre NodeJS para el navegador Google Chrome (Cloud9IDE).

eXo IDE Cloud: Es una aplicación web que provee un entorno rico para el desarrollo de diversos contenidos, códigos fuente así como servicios. No requiere instalación adicional pues se ejecuta sobre los navegadores, posibilitando el acceso y manejo de ficheros desde cualquier lugar (eXo IDE, 2012). eXo IDE ofrece:

- ✓ Manejo con el Sistema de Archivos Remoto a través del Sistema Virtual de Ficheros que incluye: navegación, bloqueo de ficheros, búsquedas y versionado.
- ✓ Editor de código con reconocimiento sintáctico y características avanzadas como auto-completamiento de código con el editor WYSIWYG para HTML y Google Gadget.
- ✓ Soporte para varios lenguajes descriptivos, así como de programación como: JavaScript, HTML, XML, CSS, Java, Groovy, PHP, Ruby, JSP.
- ✓ Herramientas para el desarrollo de aplicaciones del lado cliente que incluyen tecnologías como Netvibes Widget, Google Gadgets, plantillas Groovy.
- ✓ Listo para usar en proyectos de Java, Java Spring y Ruby on Rails.
- ✓ Desarrollo, ejecución y depuración de aplicaciones del lado servidor con interacción hacia el cliente vía servicios REST[37].
- ✓ Sistema de Control de Versiones GIT con soporte incluido para la mayoría de las operaciones con servidores locales y remotos.

J2EE[38] define dos roles principales para el desarrollo de software: administradores y desarrolladores. Cada uno define casi las mismas funcionalidades excepto que los desarrolladores no pueden desplegar sus servicios REST en entornos comunes de ejecución, para estos se definen entornos de prueba (eXo IDE, 2012).

eXo IDE Cloud brinda soporte para los navegadores web:

[37] **REST** Acrónimo de *Representational State Transfer.*

[38] **J2EE** *Java 2 Platform Enterprise Edition* (Plataforma Java2 Edición Empresarial).

- ✓ Mozilla Firefox 3.6*
- ✓ Safari 5.0*
- ✓ Google Chrome 11.0*
- ✓ Internet Explorer 7.0*

De manera general antes de proseguir el estudio sobre las herramientas y tecnologías que se utilizarán, es necesario acotar los resultados fundamentales que abordaron las soluciones que se investigaron. Por un lado se encuentran varias herramientas de entornos de escritorio, las cuales poseen gran madurez en conceptos de desarrollo, arquitecturas potentes y altamente escalables como es el caso de NetBeans y Eclipse. Estos además poseen integración con múltiples lenguajes de programación e incluyen potentes compiladores para dichos lenguajes, aunque de manera general potencian una tecnología en particular, entre los que se destacan los *frameworks*: NetBeans, Qt y .NET. La tendencia fundamental de los mismos está dirigida al desarrollo de entornos de colaboración por lo que están enfocando sus herramientas sobre los conceptos asociados al control de versiones y *teamworking*[39]*.*

Por otro lado los WIDE son entornos novedosos y muy recientes que están basados en las buenas prácticas adquiridas por los equipos de desarrollo de los IDE clásicos, así como las referencias de los usuarios. Los mismos presentan interfaces mucho mejor logradas y amenas que sus progenitores de entornos escritorio. Cabe destacar que actualmente no poseen el mismo nivel de funcionalidades que los entornos *desktop* debido a que son recientes en la mayoría de los casos y sus objetivos no son los mismos en principio. Mientras que los IDE están guiados por los principios de garantizar un alto rendimiento de las aplicaciones sobre el uso de recursos de hardware de los equipos, los WIDE están totalmente enfocados en facilitar el acceso, la colaboración entre desarrolladores y los principios del *cloud computing*[40], aunque no dejan de la mano el rendimiento en la obtención de soluciones. Generalmente ofrecen soporte para software en entornos web y por consiguiente las los lenguajes HTML, CSS y JavaScript, pues aprovechan los recursos nativos que poseen las aplicaciones clientes sobre las cuales se ejecutan: el navegador web. Estos además están dirigidos al igual que los IDE en potenciar primariamente una tecnología particular como es el caso de Cloud IDE sobre NodeJS.

2.2. Metodología de Desarrollo de Software

Una metodología de desarrollo de software es aquella que hace posible la planificación, organización y desarrollo de un sistema o proyecto, independientemente de su temática o complejidad. Actualmente estas metodologías son una guía en el proceso de desarrollo de las aplicaciones informáticas, permitiendo que se obtengan resultados con la mayor calidad, rapidez y eficiencia posible, para evitar cometer errores futuros.

[39] ***Teamworking*** Término asociado al desarrollo de software ágil en entornos colaborativos.
[40] ***Cloud computing*** (Computación en la nube).

En el momento de comenzar una aplicación, se debe seleccionar primeramente la metodología de desarrollo a emplear, para lo cual existen distintos tipos. Las metodologías se clasifican en ágiles y pesadas o tradicionales. Estas últimas son más prácticas cuando el proyecto o aplicación a desarrollar es compleja y se tienen claros los requisitos de la misma. Además comprenden una definición detallada de los procesos y tareas a realizar, que sirven de apoyo, pues generan documentación suficiente para una mejor comprensión, en este caso se encuentra el RUP[41].

La versión ágil del proceso unificado de Scott Ambler AUP[42] resulta ser un híbrido de las metodologías XP y RUP. Este enfoque aplica técnicas ágiles como: TDD[43], AMDD[44], Gestión del Cambio Ágil y Refactorización de Base de Datos para mejorar la productividad, además integra la mayoría de los procesos definidos por RUP, heredando la robustez de la misma (Ambler, 2005). Está enfocada además a la interacción directa con el cliente, característica que hereda de la metodología XP. Desafortunadamente como no se cuenta con ningún cliente que guíe el proceso de captura de requisitos para la construcción de la solución que se propone al término de la presente investigación, no es aplicable la metodología AUP por lo que debe emplearse otra metodología que proponga un marco de trabajo más amplio.

De acuerdo a lo antes descrito debido a las necesidades para la solución del problema investigativo, se define como metodología a aplicar: RUP. La decisión anterior se debe a que desde un inicio están bien definidos los requisitos de software, sobre los que no ocurrirán cambios constantes, además se necesita de una documentación completa y detallada. No se puede dejar de mencionar que la misma sigue la línea de desarrollo del proyecto al cual pertenece, además de ser actualmente la más utilizada en la Universidad. A continuación se describe más detalladamente la metodología en cuestión.

2.2.1. RUP

RUP es una de las metodologías pesadas más conocidas y utilizadas. Posee suficiente robustez y precisión, por lo que permite tanto controlar como documentar muy bien el desarrollo del software, mitigando los riesgos que puedan existir en el proceso. Esto propicia que en todo momento exista un enfoque de trabajo bien estructurado, así como la asignación correcta de tareas y responsabilidades.

Para guiar el proceso de desarrollo del software emplea nueve flujos de trabajo organizados en cuatro fases de desarrollo. Las cuatro fases de desarrollo que definen el ciclo de vida son: (Jacobson, y otros, 2000)

Inicio: tiene como objetivo establecer la visión del proyecto o aplicación y lo que se quiere realizar.

Elaboración: es donde se define la arquitectura y los elementos base para la implementación.

Construcción: es donde se implementa y se va obteniendo una solución inicial parcial.

Transición: es donde se adquiere el producto acabado y definido, incluyendo su mantenimiento.

[41] **RUP** Acrónimo de *Rational Unified Process*, en español (Proceso Racional Unificado).
[42] **AUP** Acrónimo de *Agile Unified Process*.
[43] **TDD** Siglas de *Test Driven Development*, en español (Desarrollo Dirigido por Pruebas).
[44] **AMDD** Siglas de *Agile Model Driven Development*, en español (Desarrollo Dirigido por Modelado Ágil).

Las características que definen a RUP son:

- ✓ **Guiado por Casos de Uso:** los casos de uso (CU) son una técnica de captura de requisitos que representan funcionalidades del sistema, las cuales definen lo que el usuario desea obtener y permiten guiar todo el ciclo de vida de la aplicación o proyecto, para crear un resultado que satisfaga las necesidades reales del usuario. Además integran a todos los flujos de trabajo de RUP, sirviendo de punto de partida y de hilo conductor. En otras palabras, esta característica es la que permite establecer trazabilidad entre los artefactos que son generados en las diferentes actividades del proceso de desarrollo.
- ✓ **Centrado en la arquitectura:** la arquitectura es la organización o estructura de las partes más relevantes de un sistema, debido a que brinda una perspectiva clara y una visión común del mismo entre todos sus implicados. En esta se describen los procesos del negocio que son más importantes, teniendo en cuenta los elementos de calidad, rendimiento, reutilización y flexibilidad. En otras palabras, es la que permite entender bien el sistema y la toma de decisiones que indican cómo tiene que ser desarrollada la aplicación y en qué orden.
- ✓ **Iterativo e incremental**: permite dividir el trabajo en partes más pequeñas conocidas como fases de desarrollo, donde cada una se puede ver como una iteración, de la cual se obtiene un incremento que produce un crecimiento en el producto. Estas iteraciones son planificadas, a medida que se obtienen sus resultados, se van integrando. Esta característica permite ir mejorando los resultados, para lograr optimización y corrección de errores en el momento adecuado, perfeccionando así la aplicación final. En otras palabras, va iterando a medida que se van ejecutando las diversas fases, volviéndose a retocar para ser mejoradas o corregidas, mediante lo cual va incrementando la obtención de resultados y la optimización del sistema en general. Esto acarrea la disminución de los fallos cometidos en el desarrollo de la aplicación y que se adquiera calidad y experiencia (The Rational Unified Process: An Introduction, 2000).

2.3. Lenguaje de modelado

Un lenguaje de modelado es un lenguaje artificial que puede ser empleado para expresar información, conocimiento o sistemas en una estructura definida por un conjunto consistente de reglas, las cuales son la clave para garantizar la interpretación del significado de los componentes en la estructura. Los lenguajes de modelado pueden ser textuales o gráficos, estos últimos utilizan técnicas de diagramación a través de símbolos reconocibles que representan conceptos y líneas que conectan los conceptos relacionándolos. En la presente investigación debido a que la metodología seleccionada RUP, se apoya sobre el Lenguaje Unificado de Modelado, se utilizará el mismo para lograr la representación adecuada en cada una de las fases del desarrollo de software, del mismo se hará una descripción a continuación.

2.3.1. Lenguaje Unificado de Modelado (UML v2.0)

Es actualmente uno de los lenguajes de modelado más usado a nivel mundial por las grandes empresas productoras de software, pues permite visualizar, especificar, desarrollar y documentar los artefactos de la aplicación de forma eficiente y entendible (Object Management Group Inc., 2003). Básicamente UML es un instrumento gráfico que permite a los desarrolladores de una aplicación tener una visión de lo que desarrollarán, entender completamente la misma y mantenerla en mente mientras crean el sistema.

Para lograr lo antes descrito se emplean una serie de elementos llamados diagramas, los que representan las diferentes proyecciones del sistema según sea la fase de desarrollo en la que se encuentre, donde cada diagrama tiene fines distintos dentro del proceso de desarrollo. UML es un lenguaje fácil de aprender, bien descriptivo y permite documentar todo el proceso de creación del software, empleándose en todas sus etapas (Herramientas de desarrollo de ingeniería de software para Linux, 2005).

2.4. Base tecnológica

Como se definió anteriormente en el Capítulo 1 de este documento, los marcos de trabajo constituyen una base para el soporte del desarrollo de software. Es importante comprender que es necesario desarrollar una herramienta que responda, en calidad de IDE, al conjunto de necesidades de los desarrolladores de software que emplean Ksike. Esta debe ser desarrollada sobre tecnología libre, de fácil acceso e instalación, multiplataforma y extensible al mayor número de usuarios posibles. Además debe constituir un ejemplo demostrativo de las capacidades del propio Ksike, por lo que se define el desarrollo de una solución web, pues este entorno posibilita cumplir con los requerimientos del sistema mencionados anteriormente.

La herramienta que se propone además debe brindar un entorno amigable, de fácil interacción y que brinde una apariencia de entorno escritorio, garantizando el nivel de experiencia de los usuarios sobre los mismos. Debe mostrar una presentación profesional y sencilla por lo que se selecciona el *framework* Ext JS como tecnología a emplear para el desarrollo de interfaces, integrándola al API cliente de Ksike para obtener mejores resultados. De ambos se muestran más detalles a continuación.

2.4.1. Ksike

Para implementar la base tecnológica de la propuesta a desarrollar se selecciona Ksike, con el objetivo de aprovechar sus potencialidades en favor de demostrar sus aplicaciones y fomentar el uso del mismo. Además de hacer uso de sus favorables esquemas arquitectónicos que posibilitan la extensibilidad y flexibilidad de los componentes.

2.4.2. ExtJS

Ext JS es una biblioteca JavaScript para el desarrollo de aplicaciones web que poseen un alto nivel de interacción con los usuarios. Un sitio web que requiere un alto número de procesos y un gran flujo de

trabajo sería el ejemplo perfecto para emplear Ext JS (Frederick, y otros, 2008). Entre sus principales características:

- ✓ Provee un conjunto de componentes de fácil empleo y compatibilidad con varios navegadores web. Entre estos componentes se encuentran: ventanas, tablas, formularios.
- ✓ Posee buena integración con el EventManager, que permite registrar las interacciones del usuario.
- ✓ Emplea Ajax como medio de comunicación con el servidor.

2.5. Lenguaje de desarrollo

Un lenguaje de desarrollo o programación es aquel elemento dentro de la Informática que nos permite crear programas usando instrucciones y operadores que se rigen por reglas. En otras palabras, es el lenguaje que emplean los desarrolladores para que la computadora realice las acciones que desean. Actualmente existen diversos tipos de lenguajes de programación que pueden clasificarse según su nivel de abstracción: (alto, híbrido o bajo nivel), paradigma de programación que implementan: (estructurados, funcional, orientado a objetos, etc.), o según de ejecución (w*eb* o d*esktop*) entre otras clasificaciones. Algunos de estos lenguajes son PHP, C, Java y ASP.NET (Arias Marin, 2008).

Los *frameworks* mencionados anteriormente serán empleados en el desarrollo de BHike (nombre de la herramienta que se propone como IDE), para solventar las deficiencias subscritas al desarrollo de software sobre la plataforma Ksike, dígase Ksike y ExtJS están implementados en los lenguajes PHP versión 5.3 y JavaScript, por lo que el desarrollo de aplicaciones sobre esas tecnologías deberá realizarse usando los mismos lenguajes, de los cuales se ofrece una breve descripción a continuación.

2.5.1. PHP v5.3

Es un lenguaje de alto nivel con técnicas de Programación Orientada a Objetos, multiplataforma, robusto, sencillo de usar, rápido, integrable, excelente para crear aplicaciones web dinámicas y robustas. PHP es uno de los lenguajes de programación que permiten programar scripts del lado del servidor, insertados dentro del código HTML, con una gran cantidad de librerías de funciones y mucha documentación. PHP5 ofrece mejoras significativas con respecto a versiones anteriores, con una orientación a objetos similar a la de Java (Festival Latinoamericano de instalación de software Libre: PHP, 2006).

Resumiendo, se incita al empleo del lenguaje PHP porque el mismo posee varias funcionalidades que permiten desarrollar aplicaciones complejas con pocas implementaciones. Otro de los elementos a tener en cuenta es la flexibilidad que ofrece en cuanto a la generación de contenido HTML, lo que posibilita darle mayor aprovechamiento a las capacidades del servidor liberando al cliente de cargas por rendimiento de sus sistemas. Además de este elemento, se suma el hecho de que la versión 1.1 del *framework* Ksike, quien ha sido seleccionado como base tecnológica de la herramienta a construir, está programado en este lenguaje.

2.5.2. JavaScript

Provee el comportamiento, tercer pilar en el actual paradigma de las aplicaciones web, el cual define a las páginas web como entes consistentes de tres partes claramente distinguibles: el contenido (HTML), la presentación (CSS) y el comportamiento (JavaScript). El código JavaScript se ejecuta sobre un ambiente anfitrión, habitualmente los navegadores web, quienes son los más comunes pero no los únicos. Esto quiere decir que posee carácter multiplataforma y además puede ser empleado en la programación de todo tipo de aplicaciones y entornos (Stefanov, Julio 2008).

JavaScript es un lenguaje prototipado orientado a objetos que actualmente se puede ejecutar en el lado servidor, en aplicaciones escritorio y en medios enriquecidos (*rich media*), además es funcional en otra docena de aplicaciones como son:

- ✓ Creación de aplicaciones web.
- ✓ Codificar aplicaciones del lado servidor, tales como ASP, o por ejemplo, código ejecutable usando Rhino[45].
- ✓ Programar *scripts* para automatizar tareas en los escritorios de Windows, a través del *Windows Scripting Host*.
- ✓ Crear extensiones/*plugins* para aplicaciones de escritorio como son: Firefox, Chrome, Dreamweaver.
- ✓ Crear aplicaciones sobre Adobe Air, las cuales son ejecutables en ambientes de escritorio (Stefanov, Julio 2008).

2.6. Servidor Web

Un servidor web es un programa que, empleando el modelo cliente/servidor y el protocolo HTTP[46], brinda una salida al contenido desde las páginas web a sus usuarios, los que deben poseer equipos de cómputo con aplicaciones clientes que manejen sus peticiones e interpreten las respuestas. Todo equipo de cómputo que reside en internet, para brindar servicios web, debe tener un software servidor. Los dos más populares son IIS[47] y el más usado Apache, del cual a continuación se brindarán algunas características (Rouse).

2.6.1. Apache 2.2

Apache es el servidor web hecho por excelencia, su fácil configuración, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa. Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal, es una tecnología gratuita, de código abierto, es un servidor altamente configurable de diseño modular. Además ofrece una alta configurabilidad en la creación y gestión de *logs*. Apache permite la creación de ficheros de *log* a

[45] **Rhino** Motor de JavaScript escrito en Java.
[46] **HTTP** Acrónimo de *HyperText Transference Protocol* (Protocolo de Transferencia de HiperTextos).
[47] **ISS** *Microsoft's Internet Information Server* (Servidor de Información en Internet de Microsoft).

medida del modo y formatos que define el administrador del sistema, de este modo se puede tener un mayor control sobre lo que sucede en el servidor (Apache Software Foundation, 2011).

2.7. Entorno de Desarrollo Integrado

En aras de agilizar el proceso de desarrollo de la solución propuesta y garantizando que se efectúe de una forma lo más entendiblemente posible, o sea, con formateo de código fuente, nomenclatura bien definida de los componentes obtenidos. Posibilitando además que se generen la menor cantidad de errores posibles durante la implementación de la herramienta se decide el empleo de uno de los IDE con mejores prestaciones actualmente y del que se agregan algunos elementos a continuación.

2.7.1. NetBeans v7.0.1

Se define a NetBeans debido a que el mismo es un IDE que posee soporte para varios lenguajes de alto nivel orientado a objetos como Java y PHP. Este también permite facilitar el empleo de varios *frameworks* de desarrollo tales como ExtJS y Symfony. Es bastante robusto por lo que las herramientas que posee permiten confort y comodidad en el desarrollo de software, siendo agradable para el usuario y sencillo de usar. NetBeans es un producto gratuito que no tiene restricciones de uso, ya que es de código abierto y tiene una comunidad que le ofrece soporte a nivel mundial (Oracle Corporation & afiliates, 2011).

2.8. Herramienta CASE

Los CASE[48] son un conjunto de instrumentos de asistencia en el desarrollo de programas informáticos, que se emplean desde la planificación, pasando por el análisis, diseño, hasta la generación del código fuente de los programas y la documentación. Este tipo de herramientas son altamente utilizadas en la actualidad por muchas empresas de producción de software para la automatización así como representación de los elementos fundamentales que componen el proceso de desarrollo de las aplicaciones o sistemas. Por tanto, aportan un alto valor económico y buenos resultados del producto a obtener (Herramientas de desarrollo de ingeniería de software para Linux, 2005).

Estas herramientas se relacionan estrechamente con la metodología de software a emplear y con el lenguaje UML, puesto que sirven de apoyo para la puesta en práctica de los mismos. Existen tres tipos de herramientas CASE: las de Alto Nivel, que son las que apoyan las fases iniciales del ciclo de vida del desarrollo del software, las de Bajo Nivel, que apoyan solamente las fases finales, y finalmente las de Cruzado de Ciclo de vida, que tienen lugar a lo largo de todo el proceso de desarrollo. Esta última clasificación es la más recomendada, debido a que asegura guiar y entender el proceso en todo momento, en este caso entran Visual Paradigm, Rational Rose entre otras (Herramientas de desarrollo de ingeniería de software para Linux, 2005).

[48] **CASE** Acrónimo de *Computer Aided Software Engineering*.

2.8.1. Visual Paradigm for UML v8.0 Enterprise Edition v5.0

Visual Paradigm for UML es una herramienta CASE cruzado de ciclo de vida multiplataforma, la cual pertenece a la categoría de software libre, brindando muchas facilidades, tales como buena documentación y soporte online. Además es una herramienta fácil de utilizar, con una interfaz agradable, que presenta soporte para la notación así como modelado de procesos de negocios y emplea una rápida respuesta con poca memoria utilizando moderadamente los tiempos del procesador, lo que le permite manejar grandes y complicadas estructuras de un proyecto en una forma más rentable para el sistema (Pressman, 2002).

También cuenta con un generador de mapeo de objetos-relacionales aplicable a la modelación de base de datos para los lenguajes de programación: Java, .NET, PHP entre otros. Esta herramienta es reconocida porque realiza de forma organizada tanto la diagramación visual como el diseño de proyectos, permitiendo así integrar y desplegar las aplicaciones (Herramientas de desarrollo de ingeniería de software para Linux, 2005).

Resumiendo, se selecciona esta herramienta para el diseño y modelado de la solución del problema de la investigación, pues la misma presenta numerosas ventajas que devienen convenientemente en varios procesos fundamentales del desarrollo de software. Varias de sus características validan esta afirmación, entre las que destacan: navegación entre código y modelo, sincronización de código fuente, generador de documentación y reportes UML PDF/HTML/MS Word, entorno de modelado visual, soporte completo de notaciones UML y aplicaciones web, diagramas de diseño automáticos sofisticados así como fácil instalación.

2.9. Conclusiones parciales

El estudio de las tendencias, tecnologías y el estado de posibles soluciones reveló la existencia de innumerables IDE que permiten la implementación de extensiones que pudieran enfocarse en automatizar los procesos de Ksike. Existen además disímiles guías y manuales que facilitan el desarrollo de las mismas, pudiéndose concretar una solución temprana al problema original. Desafortunadamente construir una extensión e incorporarla a un Entorno de Desarrollo existente, no garantiza el consumo de los recursos del *framework,* motivo por el cual no se puede probar dicha tecnología, ni detectar presuntos errores y deficiencias. La tendencia actual está orientada además, a brindar soporte al desarrollo de entornos colaborativos mediante los WIDE, por lo que la construcción de la herramienta estará basada en este tipo de entorno.

El mayor interés del equipo de desarrollo de Ksike está enmarcado en garantizar el correcto funcionamiento del marco de trabajo, así como su empleo y desarrollo. Debido a esto, se hace indispensable desarrollar una herramienta nueva, que posibilite sentar las bases para las futuras proyecciones de la tecnología. La misma debe permitir además crear un ambiente que garantice retroalimentar al marco de trabajo.

CAPÍTULO 3: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

En este capítulo se desarrolla un análisis de los principales conceptos asociados al negocio de Ksike en aras de elaborar el Modelo de dominio. También se realiza la especificación de varios requisitos funcionales y no funcionales que se deben tener en cuenta al desarrollar el sistema. Se definen además los Modelos de Casos de Usos del sistema, estándares así como otros elementos conceptuales que permitan seleccionar el tipo de aplicación y arquitectura que mejor se aplican a la solución.

3.1. Modelo del Dominio

Debido a que en el negocio asociado al *framework* Ksike, no existen un conjunto de operaciones, trabajadores, entidades u otros elementos que definan procesos del negocio asociado al marco de trabajo, se hace necesario realizar un modelo del dominio, el cual incluye un modelo conceptual. La descomposición del problema en conceptos u objetos individuales constituye el paso esencial de un análisis o investigación orientado a objetos. Un modelo conceptual explica a sus creadores los conceptos más significativos en un dominio del problema, así como sus relaciones y atributos. Este es el artefacto más importante a crear durante el análisis orientado a objetos (Larman).

3.1.1. Diagrama de conceptos

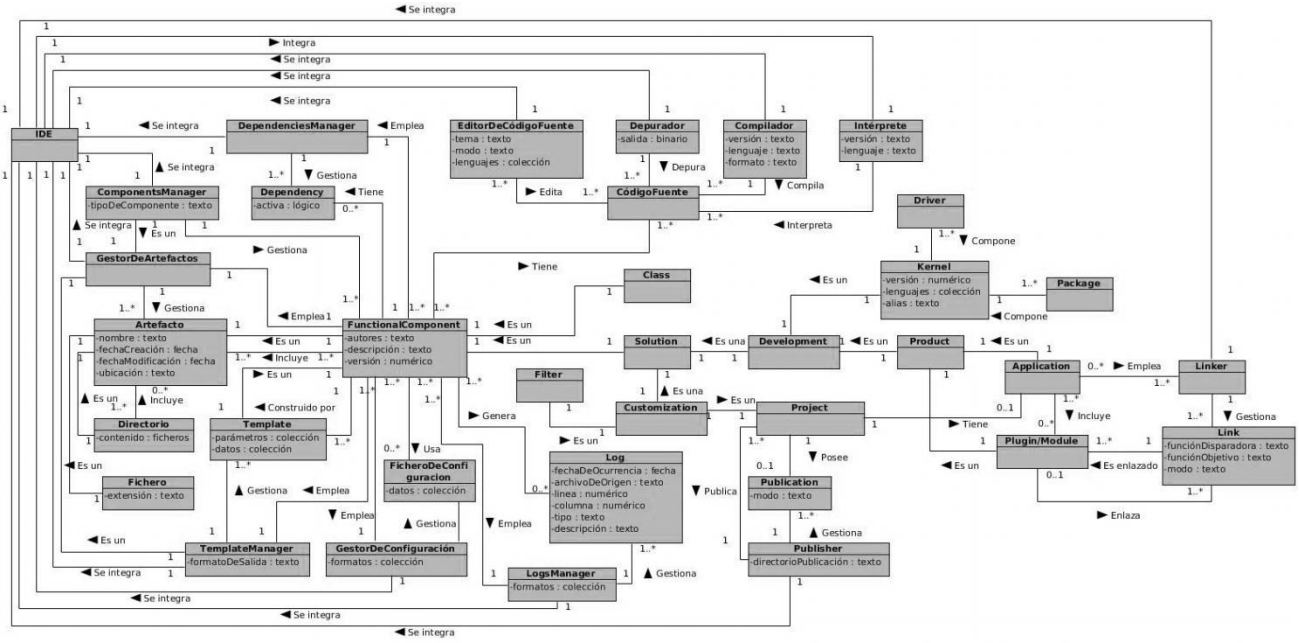


Figura 3.1.1: Modelo del dominio asociado al negocio de Ksike.

3.1.2. Descripción del diagrama

La Figura 3.1.1 mostrada anteriormente, refleja la interacción entre los conceptos esenciales asociados al negocio del *framework* Ksike y los mecanismos que desempeñan los principales roles en la gestión de los mismos. En esta ocupan un papel importante los artefactos, que pueden ser: ficheros,

directorios, plantillas o componentes funcionales. Las plantillas (*template*) son agrupaciones de artefactos que tienen como objetivo de servir de molde para la generación de componentes funcionales. Estos últimos, son artefactos que constituyen partes funcionales de software, pues brindan comportamientos y el marco elemental para programar la lógica de los sistemas que componen. Para garantizar lo anterior, deben tener en su composición código fuente asociado, pues mediante el mismo es que otorgan dichas capacidades a sus empleadores.

Los componentes funcionales son capaces además de usar ficheros de configuración y además generar *logs* o reportes de sucesos ante fallos, excepciones o notificaciones del sistema. Además pueden tener dependencias o interrelaciones unos con otros. Existen entre los componentes funcionales las clases y las soluciones, siendo las primeras la unidad más básica de implementación de lógica sobre el paradigma de POO. Las soluciones en cambio, definen recursos más estructurados y con fines mucho más ambiciosos, pues están formadas por varias clases y componentes.

Existen dos tipos fundamentales de soluciones, las orientadas a la configuración y las orientadas al desarrollo. Entre las configuraciones se relacionan los filtros y los proyectos, siendo las primeras, reglas de carga que se configuran para comportamientos específicos de *plugins*/módulos del sistema, mientras que los proyectos encapsulan recursos y configuraciones genéricas a un conjunto de componentes simultáneamente, entre los que se encuentra las aplicaciones. Estas al igual que los *plugins*/módulos son soluciones orientadas al desarrollo de productos finales, a diferencia de los drivers y los paquetes, que están orientadas al desarrollo del núcleo del marco de trabajo Ksike. Entre los *plugins*/módulos se establecen los enlaces que son gestionados por las aplicaciones, pues son estas las que disponen de la información, así como la responsabilidad necesarias para establecer el enlazamiento entre sus partes componentes, según define la arquitectura de Ksike.

Para gestionar los elementos antes mencionados, existen un conjunto de mecanismos que se encargan de gestionar: artefactos, componentes, plantillas, dependencias, configuraciones, *logs*, publicaciones, enlaces y código fuente. Todos estos se integran para conceptualizar un IDE. Estos mecanismos son herramientas que se encargan de propiciar un ambiente de colaboración ideal para agilizar los procesos de gestión anteriormente mencionados. La mayoría de estos son provenientes del negocio asociado al *framework* Ksike, pero se incluyen otros que constituyen componentes clásicos de los entornos de desarrollo integrado como son: editor de código fuente, depurador, intérprete y compilador, todos ellos analizados en el Capítulo 1 de este documento. Para más información refiérase al documento de Ingeniería: Modelo del Dominio, donde se definen más al detalle cada uno de los conceptos representados en la Figura 3.1.1.

3.2. Especificación de requisitos

Según Roger S. Pressman, la especificación del sistema es la base para la construcción del producto final pues describe los requisitos que debe este cumplir. Esta sirve como fundamento para la ingeniería del hardware, software, bases de datos y humana. Describe la función así como características de un

sistema de computación y las restricciones que gobiernan su desarrollo. La especificación delimita cada elemento del sistema. La Especificación del Sistema describe la información de datos y control que entran y salen del sistema (Pressman, 2002).

A continuación se describen los principales requisitos, funcionales y no funcionales que debe cumplir el sistema desarrollado. De un total de sesenta y un requisitos funcionales identificados, solo se mostrarán los más críticos. Para más información sobre los demás, referirse al documento ingenieril: Especificación de Requisitos.

3.2.1. Requisitos funcionales

RF 1. La herramienta debe proveer mecanismos para la gestión de artefactos (ficheros y directorios). Esta debe permitirle al usuario:

- **RF 1.1.** Crear fichero.
- **RF 1.2.** Crear directorio.
- **RF 1.3.** Abrir fichero.
- **RF 1.4.** Abrir directorio.
- **RF 1.5.** Modificar fichero.
- **RF 1.6.** Modificar directorio.
- **RF 1.7.** Eliminar fichero.
- **RF 1.8.** Eliminar directorio.

RF 2. La herramienta debe proveer mecanismos para la gestión de proyecto. Esta debe permitir:

- **RF 2.1.** Crear proyecto monolítico.
- **RF 2.2.** Crear multi-proyecto.
- **RF 2.3.** Modificar información de proyecto.
- **RF 2.4.** Abrir proyecto.
- **RF 2.5.** Abrir proyecto reciente.
- **RF 2.6.** Cerrar proyecto.
- **RF 2.7.** Eliminar proyecto.

RF 3. La herramienta debe proveer mecanismos para la gestión de aplicación. Esta debe permitir:

- **RF 3.1.** Crear aplicación.
- **RF 3.2.** Agregar aplicación existente.
- **RF 3.3.** Modificar información de aplicación.
- **RF 3.4.** Eliminar aplicación.

RF 4. La herramienta debe proveer mecanismos para la gestión de *plugin*/módulo. Esta debe permitir:

- **RF 4.1.** Adicionar nuevo *plugin*/módulo.
- **RF 4.2.** Modificar información de plugin/módulo.
- **RF 4.3.** Instalar plugin/módulo.
- **RF 4.4.** Eliminar plugin/módulo.

RF 5. La herramienta debe proveer mecanismos para la gestión de *driver*. Esta debe permitir:

- **RF 5.1.** Crear driver.
- **RF 5.2.** Modificar información de driver.
- **RF 5.3.** Exportar driver.
- **RF 5.4.** Instalar driver.
- **RF 5.5.** Desinstalar driver
- **RF 5.6.** Eliminar *driver*.

RF 6. La herramienta debe proveer mecanismos para la gestión de paquetes. Esta debe permitir:

- **RF 6.1.** Crear paquete.
- **RF 6.2.** Modificar información de paquete.
- **RF 6.3.** Instalar paquete.
- **RF 6.4.** Desinstalar paquete.
- **RF 6.5.** Eliminar paquete.

RF 7. La herramienta debe proveer mecanismos para la gestión de filtros. Esta debe permitir:

- **RF 7.1.** Crear filtro.
- **RF 7.2.** Eliminar filtro.

RF 8. La herramienta debe proveer mecanismos para la gestión de clases. Esta debe permitir:

- **RF 8.1.** Crear clase.
- **RF 8.2.** Eliminar clase.

RF 9. La herramienta debe proveer mecanismos para la edición de código fuente. Esta debe permitir:

- **RF 9.1.** Editar código fuente.
- **RF 9.2.** Guardar código fuente.

RF 10. La herramienta debe proveer mecanismos para la gestión de *logs*. Esta debe permitir:

- **RF 10.1.** Listar *logs*.
- **RF 10.2.** Guiar al origen del *log*.
- **RF 10.3.** Eliminar *logs*.

RF 11. La herramienta debe proveer mecanismos para la gestión de enlaces. Esta debe permitir:

- **RF 11.1.** Visualizar enlaces.
- **RF 11.2.** Crear enlace.
- **RF 11.3.** Activar enlace.
- **RF 11.4.** Desactivar enlace.
- **RF 11.5.** Modificar enlace.
- **RF 11.6.** Eliminar enlace.

RF 12. La herramienta debe proveer mecanismos para la gestión de la publicación de proyectos. Esta debe permitir:

- **RF 12.1.** Crear publicación total de proyecto.
- **RF 12.2.** Crear publicación parcial de proyecto.

RF 12.3. Crear publicación limitada de proyecto.

RF 12.4. Actualizar publicación.

RF 12.5. Eliminar publicación.

3.2.2. Requisitos no funcionales

3.2.2.1. Usabilidad

RNF 1. Fácil Navegación

El sistema podrá ser usado por personas con conocimientos básicos en el manejo de computadoras. Se emplearán componentes que indiquen al usuario el estado de los procesos que por su complejidad requieran de un tiempo de procesamiento apreciable.

RNF 2. Iconografía

Las funcionalidades principales del sistema estarán orientadas a iconos para un mayor reconocimiento por parte del usuario.

3.2.2.2. Fiabilidad

RNF 3. Fallos y errores

La herramienta posee un sistema de almacenamiento de fallos y errores que registra las anomalías del sistema en el directorio app/log.

RNF 4. Tiempo de reparación

El tiempo medio de reparación, en caso de fallos es de 7 días.

RNF 5. Disponibilidad

Tanto la información como las funcionalidades del sistema estarán disponibles siempre y cuando esté habilitado el servicio, por lo que el usuario podrá acceder a ellas sin problemas.

3.2.2.3. Eficiencia

RNF 6. Tiempo de respuesta

El tiempo de respuesta puede variar entre 0 - 30 segundos, de acuerdo a la cantidad de información a procesar, entre mayor cantidad de información mayor será el tiempo de procesamiento.

3.2.2.4. Soporte

RNF 7. Mediante Negociación

La aplicación recibirá mantenimiento en el período de tiempo determinado por el equipo de desarrollo y los clientes.

3.3. Descripción del sistema

3.3.1. Definición de los actores del sistema

Un actor del sistema es un agente externo que interactúa con el sistema en pos de obtener un resultado esperado. El sistema cuenta con los actores que se especifican a continuación en la Tabla 3.3.1:

Actor	**Descripción**
Desarrollador	Es el usuario que tendrá acceso a todas las funcionalidades del sistema.

Tabla 3.3.1: Listado de actores del sistema BHike.

Como se define en la tabla anterior, el sistema propuesto posee un solo actor denominado desarrollador, que es el encargado de desencadenar todas las operaciones y procesos del sistema. Este actuará en consecuencia de las acciones realizadas por el actor.

3.3.2. Listado de casos de uso

Una vez recopilados los requisitos y los actores, el ingeniero del software o analista, puede crear un conjunto de escenarios que identifiquen una línea de utilización para el sistema que va a ser construido. Estos escenarios, conocidos también como casos de uso, facilitan una descripción de cómo se usará el sistema. En general, un caso de uso es, simplemente, una descripción del papel que desempeña un actor mientras interactúa con el acontecer del sistema (Pressman, 2002). Enfatizando esta definición al marco de la solución que se describe en este documento, se ha identificado que el desarrollador puede ejecutar varias operaciones sobre el sistema, las cuales se muestran a continuación en la Tabla 3.3.2.

Referencia a Requisitos	**Nombre del Caso de Uso**	**Prioridad**
RF: 1.1/1.2/1.3/1.4/1.5/1.6/1.7/1.8	Gestionar Artefactos	Critico
RF: 2.1/2.2/2.3/2.4/2.5/2.6/2.7	Gestionar Proyecto	Critico
RF: 3.1/3.2/3.3/3.4	Gestionar Aplicación	Critico
RF: 4.1/4.2/4.3/4.4	Gestionar *Plugin*/Módulo	Critico
RF: 5.1/5.2/5.3/5.4/5.5/5.6	Gestionar *Driver*	Secundario
RF: 6.1/6.2/6.3/6.4/6.5	Gestionar Paquetes	Secundario
RF: 7.1/7.2	Gestionar Filtros	Secundario
RF: 8.1/8.2	Gestionar Clases	Secundario
RF: 9.1/9.2	Editar Código Fuente	Critico
RF: 10.1/10.2/10.3	Gestionar *Logs*	Critico
RF: 11.1/11.2/11.3/11.4/11.5/11.6	Gestionar Enlaces	Secundario
RF: 12.1/12.2/12.3/12.4/12.5	Gestionar Publicaciones	Crítico

Tabla 3.3.2: Listado de Casos de Uso.

3.3.3. Diagrama de casos de uso

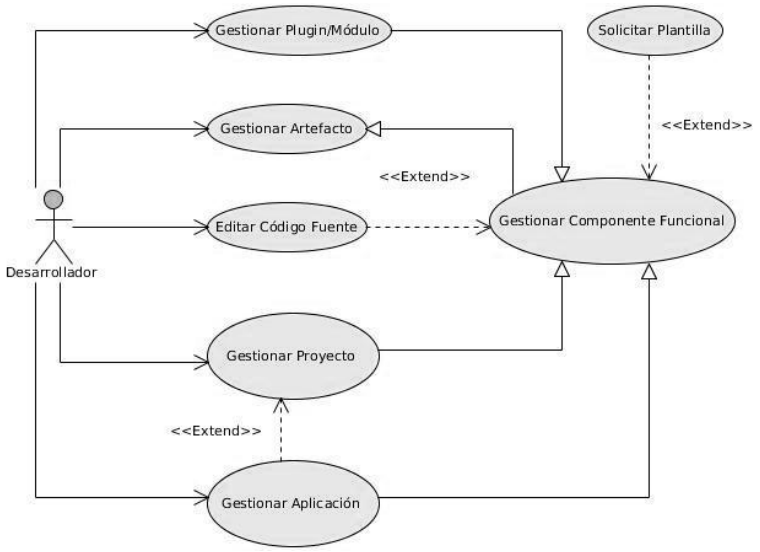


Figura 3.3.1: Diagrama de casos de usos del sistema.

La Figura 3.3.1 muestra 7 de un conjunto de 15 casos de uso que fueron identificados para agrupar los requisitos que debe cumplir la herramienta. Los mismos fueron descritos anteriormente en la sección de requisitos funcionales y se enfocan en resolver los problemas latentes asociados a las dificultades que influyen directamente en el empleo de Ksike. Debido a la complejidad que poseen estos casos de uso, al gran número de funcionalidades que deben ser implementadas para garantizar una respuesta adecuada a las necesidades planteadas, la cantidad de desarrolladores de la herramienta y el tiempo en el que debe ser entregado el prototipo funcional, no es posible llevar a cabo el ciclo completo para todos los casos de uso identificados. Es por esto que se priorizarán los que tienen mayor impacto en el resultado final a mano de los usuarios y se dejará una base documental de los que deben describirse e implementarse en versiones posteriores.

Centrándose en el objetivo principal de la presente investigación, que es proveer un marco para agilizar el desarrollo de software con el *framework* Ksike, entre los casos de usos que serán priorizados se encuentran: Gestionar Artefacto, Gestionar Proyecto, Gestionar Aplicación, Gestionar *Plugin*/Módulo y Editar Código Fuente. También se analizarán algunos otros casos de uso que tienen repercusión directa sobre la gestión de artefactos y componentes funcionales que define el marco de trabajo, como son: Solicitar Plantilla y Gestionar Componente Funcional. Entre ellos se selecciona el CUS Gestionar Proyecto para ser descrito a continuación de forma extendida, aunque para obtener más información sobre los demás casos de uso, se recomienda remitirse al artefacto de Ingeniería: Modelo de Casos de Uso del Sistema.

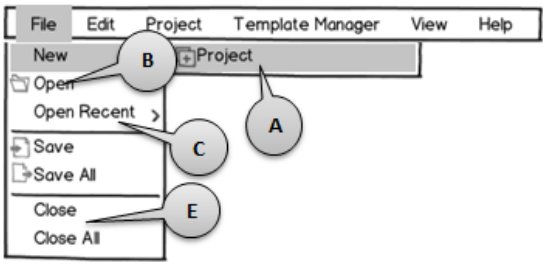
3.3.4. Descripción extendida de un caso de uso

Caso de Uso:	Gestionar Proyecto
Actor:	Desarrollador
Propósito:	Esta funcionalidad se lleva a cabo con el objetivo de manipular la información asociada a los proyectos permitiendo al usuario la creación de nuevas

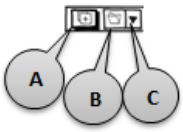
	proyectos, así como: modificar, cerrar o eliminar las ya existentes.
Resumen:	El caso de uso inicia cuando el usuario selecciona la opción correspondiente entre las que se describen a continuación y concluye cuando el sistema guarda los datos y notifica los resultados.
Precondiciones:	No tiene
Referencias	RF 9, RF 10, RF 11, RF 12, RF 13, RF 14, RF 15
Prioridad	Crítico

Flujo Normal de Eventos

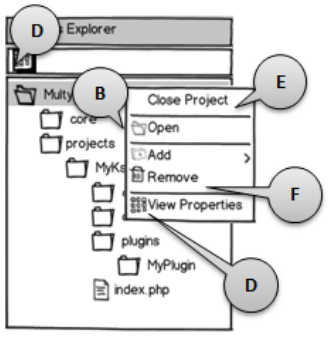
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso se inicia cuando el usuario selecciona la opción crear (**A**), abrir (**B**), abrir reciente (**C**), modificar información (**D**), cerrar (**E**) o eliminar (**F**) proyecto en el menú principal, en la barra de herramientas principal, en el menú del navegador de soluciones o en la sección de la página de inicio [Ver Interfaz 7, Interfaz 8, Interfaz 9 e Interfaz 10 respectivamente].	**2.** Brinda las opciones de: – Si selecciona **A**, ver Sección "Crear Proyecto". – Si selecciona **B**, ver Sección "Abrir Proyecto". – Si selecciona **C**, ver Sección "Abrir Proyecto Reciente". – Si selecciona **D**, ver Sección "Modificar Información". – Si selecciona **E**, ver Sección "Cerrar Proyecto". – Si selecciona **F**, ver Sección "Eliminar Artefacto" del caso de uso Gestionar Artefacto. **3.** El caso de uso termina cuando se guardan los datos que se generen y se notifique al usuario los resultados.



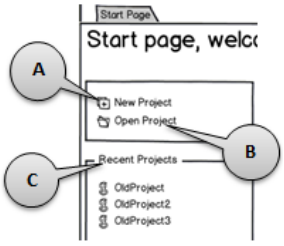
Interfaz 7: Barra de Menú principal.



Interfaz 8: Sección de la barra de herramientas principal.



Interfaz 9: Explorador de Artefactos.

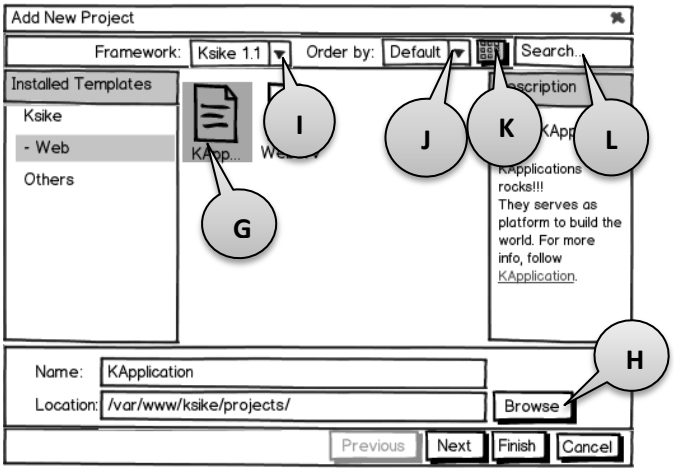


Interfaz 10: Sección de la página de inicio.

Sección "Crear Proyecto"

Flujo Normal de Eventos

Acción del Actor	**Respuesta del Sistema**
1. Selecciona la opción **A** del menú principal, la barra de herramientas principal, el menú del navegador de soluciones o la sección de la página de Inicio. [Ver Interfaz 7, Interfaz 8 e Interfaz 10].	**2.** Solicita la información de las plantillas que se encuentran alojadas en el repositorio de plantillas. **3.** Muestra la ventana "*Add New Project*", otorgándole la posibilidad al usuario de escoger qué tipo de proyecto desea generar, a partir de una plantilla existente.
4. Selecciona la plantilla **G** base que desea.	**5.** Muestra una descripción de la plantilla seleccionada. Brinda además nombre y ubicación predefinidos. **6.** Si la plantilla seleccionada no requiere datos adicionales se habilita el botón "*Finish*".
7. Modifica los datos (nombre y ubicación) en el formulario principal y presiona el botón "*Finish*". [Ver Interfaz 11].	**8.** Valida los datos, procesa la información y crea el proyecto. **9.** Automáticamente ejecuta la acción "Abrir Proyecto" y termina el caso de uso.

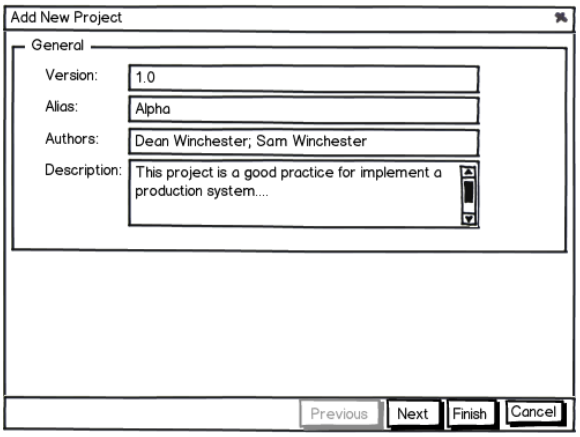


Interfaz 1: Ventana Adicionar Nuevo Proyecto.

Flujos Alternos

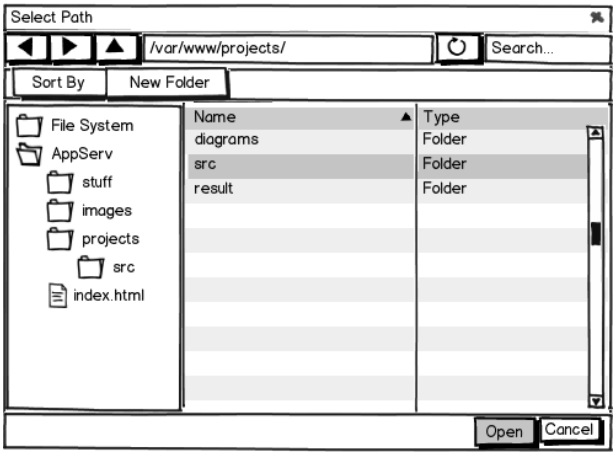
Acción del Actor	**Respuesta del Sistema**
1. Deja el nombre o la ubicación en blanco.	**2.** Muestra un mensaje indicando que hay campos requeridos en el formulario. **3.** El flujo alterno se remite al paso 7 del flujo normal de eventos.
	1. Si la plantilla seleccionada requiere datos adicionales, el sistema muestra los botones: "*Next*" y "*Previous*" y deshabilita el botón "*Finish*" si los datos requeridos son obligatorios.
2. Presiona el botón "*Next*". [Ver Interfaz 11].	**3.** Muestra tantos campos y formularios como especifique la plantilla

	seleccionada. [Ver Interfaz 12].
4. Introduce los datos requeridos.	**5.** Valida los datos, procesa la información y si no son requeridos datos adicionales se habilita el botón "*Finish*".
6. Presiona el botón "*Finish*". [Ver Interfaz 12].	**7.** Crea el proyecto y automáticamente ejecuta la acción "Abrir Proyecto" y termina el caso de uso.



Interfaz 2: Panel de datos personalizados de una plantilla.

1. Selecciona una dirección diferente para el proyecto **H**.	**2.** Muestra una ventana para navegar por el sistema de archivos. [Ver Interfaz 13].
3. Navega hasta el directorio deseado y presiona el botón "*Open*".	**4.** Actualiza el valor de la ubicación del proyecto, valida los datos, procesa la información y si no son requeridos datos adicionales se habilita el botón "*Finish*".
5. Presiona el botón "*Finish*". [Ver Interfaz 11].	**6.** Automáticamente ejecuta la acción "Abrir Proyecto" y termina el caso de uso.

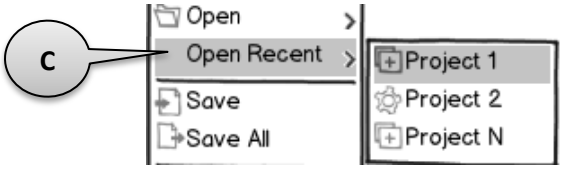


Interfaz 3: Ventana de navegación en el sistema de directorios.

1. Selecciona un nuevo valor en "*Framework*" **I** [Ver Interfaz 11].	**2.** Filtra las plantillas compatibles con la versión del *framework* seleccionado y las muestra. Se dirigen las acciones al paso # 4 del flujo normal de eventos.

1. Selecciona un valor diferente de ordenación de las plantillas **J** [Ver Interfaz 11].	**2.** Ordena las plantillas según el criterio seleccionado y se dirigen las acciones al paso # 4 del flujo normal de eventos.
1. Cambia el modo de visualización de plantillas **K** [Ver Interfaz 11].	**2.** Varía la visualización de los datos de la plantilla entre icono o detalle, según la selección del usuario. Se dirigen las acciones al paso # 4 del flujo normal de eventos.
1. Modifica los datos en el formulario principal.	**2.** Valida los datos y si están incorrectos muestra un mensaje de notificación.
1. Presiona el botón "*Cancel*" o el botón cerrar de la ventana [Ver Interfaz 11].	**2.** Interrumpe el proceso y termina el caso de uso.
Sección "Abrir Proyecto"	
Flujo Normal de Eventos	
1. Selecciona la opción **B** [Ver Interfaz 11].	**2.** Muestra una ventana, que permite navegar por el sistema de archivos y permite al usuario localizar el proyecto que desea abrir [Ver Interfaz 13].
3. Navega hasta el directorio donde se encuentra el proyecto y presiona el botón "*Open*" o presiona doble *click* sobre el proyecto.	**4.** Verifica que el proyecto seleccionado es válido. **5.** Almacena el proyecto en el registro de proyectos recientes. **6.** Carga toda la información asociada al mismo, así como su composición física en el Explorador de Artefactos [Ver Interfaz 9]. **7.** Concluye el caso de uso.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
1. Presiona el botón "*Cancel*" o el botón cerrar ventana.	**2.** Cierra la ventana y detiene la ejecución del caso de uso.
Sección "Abrir Proyecto Reciente"	
Flujo Normal de Eventos	

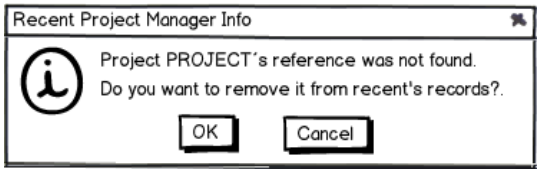
1. Despliega el menú **C** [Ver Interfaz 7, Interfaz 8 e Interfaz 10] y presiona *click* sobre uno de los elementos mostrados. [Ver Interfaz 14].	**2.** Verifica que el proyecto seleccionado existe. **3.** Actualiza el proyecto en el registro de proyectos recientes. **4.** Carga toda la información asociada al mismo, así como su composición física en el Explorador de Artefactos [Ver Interfaz 9]. **5.** Concluye el caso de uso.



Interfaz 4: Menú "Abrir Projectos Recientes".

Flujos Alternos

1. El usuario despliega el menú **C** [Ver Interfaz 7, Interfaz 8 e Interfaz 10] y presiona *click* sobre uno de los elementos mostrados. [Ver Interfaz 14].	**2.** El sistema verifica que el proyecto seleccionado existe. En caso de que no sea encontrado, muestra un mensaje indicando el problema y brinda la opción de eliminar la referencia [Ver Interfaz 15].
3. Presiona el botón "*OK*" o "*Cancel*".	**4.** Si el usuario presiona "*OK*", elimina la referencia del proyecto de los registros recientes y termina el caso de uso. En caso contrario concluye el caso de uso sin cambios en los registros de proyectos recientes.

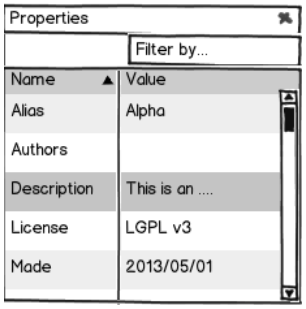


Interfaz 5: Ventana de confirmación de eliminación de la referencia de un proyecto reciente.

Sección "Modificar Información"

Flujo Normal de Eventos

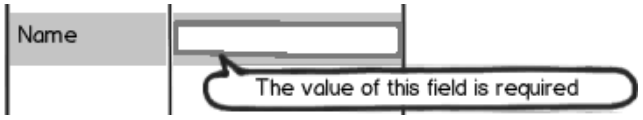
Acción del Actor	**Respuesta del Sistema**
1. Presiona *click* sobre el botón de propiedades o la opción "*View Properties*" **D** [Ver Interfaz 9].	**2.** Muestra el panel "*Properties*" que posee una tabla que contiene la información asociada al proyecto y que puede ser editada por el usuario [Ver Interfaz 6].
3. Modifica los parámetros que desea siempre que los mismos sean editables.	**4.** Verifica que los datos introducidos por el usuario son válidos
5. Al concluir presiona la tecla *Enter*.	**6.** Hace persistente la información. **7.** Notifica al usuario de la realización de los cambios y concluye el caso de uso.



Interfaz 6: Panel de propiedades.

Flujos Alternos

Acción del Actor	**Respuesta del Sistema**
1. Deja campos en blanco.	**2.** Muestra un mensaje indicando que los valores son requeridos y remite las acciones al paso # 3 del flujo normal de eventos.



Interfaz 7: Panel de propiedades con valores en blanco.

Sección "Cerrar Proyecto"

Flujo Normal de Eventos

Acción del Actor	**Respuesta del Sistema**
1. Presiona *click* derecho sobre un proyecto abierto en el navegador de soluciones y selecciona la opción "*Close*" o "*Close Project*" **E** [Ver Interfaz 7 e Interfaz 9].	**2.** Guarda el estado actual de las operaciones del proyecto y cierra el mismo.

Sección "Eliminar Proyecto"

Flujo Normal de Eventos

Acción del Actor	**Respuesta del Sistema**

1.	Presiona *click* derecho sobre un proyecto abierto en el Explorador de Artefactos [Ver Interfaz 9] y selecciona la opción "*Remove*" **E**.	**2.**	Muestra una ventana de confirmación.
3.	Presiona el botón "*Yes*".	**4.**	Elimina la información y contenido asociado al proyecto destruyendo su persistencia. De esta forma concluye el caso de uso.
Flujos Alternos			
Acción del Actor		**Respuesta del Sistema**	
3.	Presiona el botón "*No*".	**4.**	Detiene la ejecución del caso de uso.
Poscondiciones			

Tabla 3.3.3: Descripción extendida del caso de uso: Gestionar Proyecto.

3.4. Sistema de Plantillas

Como se ha venido analizando a lo largo de este documento, el trabajo con el *framework* Ksike consecuentemente está precedido de varios elementos que afectan el empleo de la tecnología a mano de los desarrolladores. Para darle una solución al conjunto de procesos que pueden ser automatizados, se ha diseñado un sistema que permita enfocar el desarrollo de pequeñas soluciones que puedan ser acopladas luego en un marco integrado. Además se ha enfocado la intención de la herramienta en desarrollo, a que apunte en dirección a construir un mecanismo que permita agilizar los procesos de creación y mantenimiento de los principales componentes del marco de trabajo. Para lograr los objetivos propuestos, se ha hecho un análisis de los mecanismos que poseen otros IDE dirigidos a automatizar el proceso de creación de componentes.

Como resultado de la investigación se ha obtenido que la mayoría de los IDE: Eclipse, NetBeans, Aptana, Qt-Creator y Visual Studio, convergen en la implementación de un sistema de generación de componentes basados en plantillas. Este sistema está descrito detalladamente en la documentación asociada a las Referencias de esquema de plantillas de Visual Studio (Microsoft), las cuales fueron empleadas para conformar un esquema propio (H&T[49]) que es aplicable al desarrollo de componentes de Ksike. Las plantillas, como se definieron al inicio del presente trabajo en el Capítulo 1, son elementos que se emplean para agilizar el proceso de instanciación o clonación de copias idénticas de un ente. Las mismas brindan una propuesta genérica de un componente y requieren de un conjunto de datos que entrada que permiten obtener una personalización del contenido que se necesita replicar, tal y como se ilustra a continuación.

[49] **H&T** Siglas de *Handlers and Templates* (Manejadores y Plantillas).

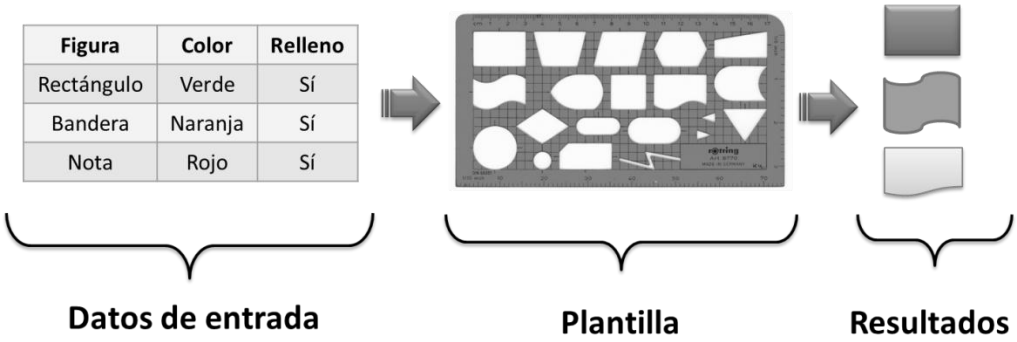


Figura 3.4.1: Ejemplo básico del uso de plantillas.

El esquema de plantillas consiste principalmente en un fichero XML que contiene un conjunto de información y relaciones con otros ficheros útiles para generar varios tipos de artefactos. Esta información está dividida en cuatro secciones fundamentales: TemplateData, TemplateContent, WizardExtension y WizardData. La primera sección posee información descriptiva, que le permite al usuario ubicarse y saber cuál es el propósito de la plantilla. TemplateContent a su vez describe la información taxonómica del componente que genera la plantilla y define además relaciones con otros elementos que permiten un mayor nivel de personalización del resultado (Microsoft).

Como se aprecia en la Figura 3.4.1, un paso fundamental en el uso de las plantillas es la recolección de los datos que se necesitan para concretar una instancia, pues de este proceso depende la diversidad de resultados que pueden ser obtenidos. De esta forma puede definirse que se obtendrá tanta diversidad en los resultados, como valores diversos se inserten en la entrada. Es por esto que, si se requieren varios datos, la sección WizardExtension asociada al esquema de plantillas, permite la incorporación de un asistente personalizado para garantizar la entrada de los valores que son necesarios para generar el componente. Asociado a este está la sección WizardData, que se emplea para acumular información que sea necesaria para el funcionamiento de un asistente personalizado.

3.5. Conclusiones parciales

El esquema de plantillas que se desarrolló permite diversidad en la generación de componentes de software, por lo que es aplicable a varios escenarios, extendiendo las capacidades de la herramienta, no solo a la automatización de los procesos asociados al framework Ksike, sino de cualquier otra que se necesite integrar. Varias de las funcionalidades descritas deben ser atendidas en estado temprano en aras de otorgarle vitalidad al marco de trabajo. Esto significa que actualmente se prevé el desarrollo de varias funcionalidades básicas de los IDE, pero debido a que no se cuenta con suficiente personal ni recursos para la construcción de la herramienta, no se podrá desarrollar en este momento una solución más abarcadora sobre los procesos del *framework*.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

En este capítulo se describen los elementos necesarios para la implementación de la solución propuesta. Se presenta el diseño de la arquitectura y Diagrama de Clases del Diseño respectivamente, de acuerdo con las fases especificadas por la metodología RUP para el desarrollo de software. Se muestra la distribución del sistema en nodos mediante el diagrama de despliegue, así como la organización de los componentes y las relaciones lógicas entre ellos a través del diagrama de componentes, quedando así conformado el modelo de implementación. Además se realiza el diseño de los casos de prueba para el sistema que se propone.

4.1. Diseño de la arquitectura

Uno de los principales pasos a seguir en la fase de Inicio (*Inception*) según RUP, es la identificación de los requisitos del sistema que se quiere construir. Habiendo realizado anteriormente este proceso, es el momento oportuno para definir el diseño de la arquitectura del sistema. "*El diseño de la arquitectura de un sistema es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes forman el sistema, cómo se relacionan entre ellos y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados como: seguridad, disponibilidad, eficiencia o usabilidad*" (de la Torre LLorente, y otros, 2010).

Según la Guía de Arquitectura N-Capas orientada al Dominio (de la Torre LLorente, y otros, 2010), el primer paso para diseñar la arquitectura es la identificación del tipo de sistema se quiere construir. En esa bibliografía se mencionan varios tipos de aplicaciones, entre las que se encuentran: aplicaciones móviles, de escritorio, RIAs[50], aplicaciones de servicios, aplicaciones web, entre otras. En el caso particular del sistema que se diseña para darle cumplimiento al objetivo trazado en la presente investigación y, según los requisitos descritos el tipo de aplicación que se pretende construir es una RIA, de las cuales se hará una breve descripción a continuación.

4.1.1. Aplicaciones enriquecidas de Internet

Las RIA son aplicaciones web diseñadas para proveer las mismas características y funcionalidades usualmente asociadas a las aplicaciones de entorno de escritorio. Generalmente separan el procesamiento a través de las redes de Internet, dividiendo la interfaz de usuario y sus actividades asociadas de la manipulación de los datos y las operaciones en el servidor. Los RIA generalmente se ejecutan sobre los navegadores web y no requieren instalaciones de software en el lado cliente para ejecutarse (Rouse).

[50] **RIA** Acrónimo de *Rich Internet Application,* en español (Aplicaciones Enriquecidas de Internet).

4.1.2. Estilos arquitectónicos

Luego de haber identificado el tipo de sistema o aplicación que se quiere construir, el siguiente paso es identificar los estilos arquitectónicos o arquitecturales sobre los cuales debe diseñarse la arquitectura base del sistema. Los estilos arquitectónicos, son un conjunto de principios que en forma de marco de trabajo abstracto, promueven el particionado y reutilización de los componentes para una familia de sistemas. Los estilos arquitectónicos brindan varios beneficios de los cuales el más importante es que proporcionan un lenguaje común en cuanto a la forma de diseñar el sistema (Microsoft, 2009).

Es necesario conocer además que existen varios estilos arquitectónicos y que los mismos están asociados a distintos categorías según los puntos de enfoque en los cuales se centran. Es necesario también entender que los sistemas generalmente responden a más de un estilo arquitectónico, por lo que para mayor comprensión a continuación en la Tabla 4.1.1 se brinda un ejemplo de algunos estilos arquitectónicos y las categorías a las que se les asocia.

Categoría	**Estilo arquitectónico**
Comunicación	SOA, Bus de mensajes
Despliegue	Cliente/Servidor, N-*Tier*
Dominio	DDD[51]
Estructura	Basada en componentes, Orientada a Objetos

Tabla 4.1.1: Estilos arquitectónicos y categorías.

Como pudo apreciarse en la conceptualización de los sistemas RIA, estos responden a un estilo arquitectónico casi por defecto: cliente/servidor. Además de este, el desarrollo de aplicaciones con el *framework* Ksike se guía fundamentalmente sobre el estilo arquitectónico basado en componentes. A estos dos estilos arquitectónicos se le suma también el de presentación desacoplada, para definir una triada de la cual se harán breves descripciones a continuación.

4.1.2.1. Arquitectura cliente/servidor

El estilo arquitectónico cliente/servidor define una relación entre dos sistemas como se muestra en la **¡Error! No se encuentra el origen de a referencia.**. Entre los múltiples beneficios que brinda este estilo arquitectónico están la seguridad, pues los datos se almacenan en el servidor que generalmente ofrecen mayor seguridad. Permite el acceso centralizado a los datos que están almacenados en el servidor lo que facilita su acceso y actualización (de la Torre LLorente, y otros, 2010).

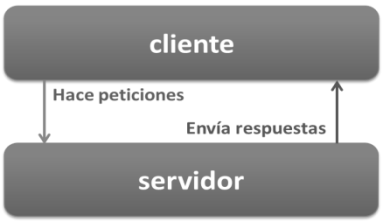


Figura 4.1.1: Modelo de la arquitectura cliente/servidor.

Aplicando sus principios fundamentales, el sistema queda dividido en dos aplicaciones principales y una red que las conecta. Brinda mayor seguridad sobre los datos por lo que agiliza las labores de mantenimiento, actualización y soporte.

[51] **DDD** Siglas de Domain Driven Design, arquitectura orientada al dominio.

4.1.2.2. Basado en componentes

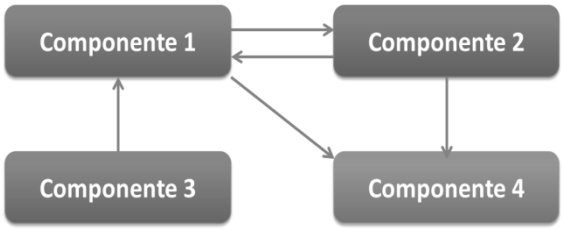


Figura 4.1.2: Modelo de arquitectura basada en componentes.

"*El estilo de arquitectura basado en componentes, describe un acercamiento al diseño de sistemas como un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver el problema*" (de la Torre LLorente, y otros, 2010). Este estilo arquitectural guía el diseño de las aplicaciones a partir de componentes individuales enfatizando en la descomposición del sistema en partes más pequeñas con interfaces bien definidas que exponen métodos, eventos y propiedades como se muestra a continuación.

Entre los principales beneficios que brinda se encuentra, la facilidad de despliegue, pues se pueden sustituir componentes por su nueva versión sin afectar a otros componentes o al sistema. Además la reducción de costos, pues se pueden usar componentes de terceros para abaratar los costos de desarrollo y mantenimiento. Otros de los principios fundamentales son que permite la reducción de la complejidad del sistema y la reusabilidad de los componentes desarrollados.

4.1.2.3. Presentación desacoplada

El estilo presentación separada o desacoplada indica cómo debe realizarse el manejo de las acciones del usuario, la manipulación de la interfaz y los datos de la aplicación, como se ilustra en la **¡Error! o se encuentra el origen de la referencia.**. Este estilo permite separar los componentes de la interfaz del flujo de datos y de la manipulación. Entre sus principales características se encuentran las facilidades que brinda para realizar las pruebas pues se pueden probar los comportamientos de forma individual lo que facilita también el soporte y mantenimiento de las aplicaciones. Otra de las ventajas que ofrece es que permite a los diseñadores crear la interfaz gráfica, mientras los desarrolladores programan la lógica para su funcionamiento.

Figura 4.1.3: Modelo del estilo arquitectónico Presentación Desacoplada.

ExtJS es la biblioteca de componentes que está definida por los requisitos de tecnología para la construcción de las vistas de presentación. Su empleo permite implementar el estilo arquitectónico presentación desacoplada sin mayores contratiempos. Esto constituye una gran ventaja que se incluye como una de las pautas a seguir para desarrollar el diseño de la arquitectura de la herramienta que se desarrollará.

4.2. Modelo del diseño

4.2.1. Diagrama de clases del diseño

El diseño arquitectónico propuesto consta de un módulo principal que se encarga de proveer un marco para la comunicación entre componentes así como la inyección de vista. Entre las principales características de BHike está el hecho de que constituye una aplicación de tipo RIA pues requiere de alto nivel de interactividad de los usuarios con el sistema. A diferencia de las aplicaciones web comunes las RIA no siguen el modelo navigacional descrito por las especificaciones clásicas de la Web 1.0, que basaba su funcionamiento en peticiones sobre ***Html Forms***. Estos últimos realizan envíos de información <<*submit*>> a una página servidora que se encarga de procesar los datos, para luego redireccionar el flujo hacia una nueva página cliente, iniciando un nuevo ciclo de navegación.

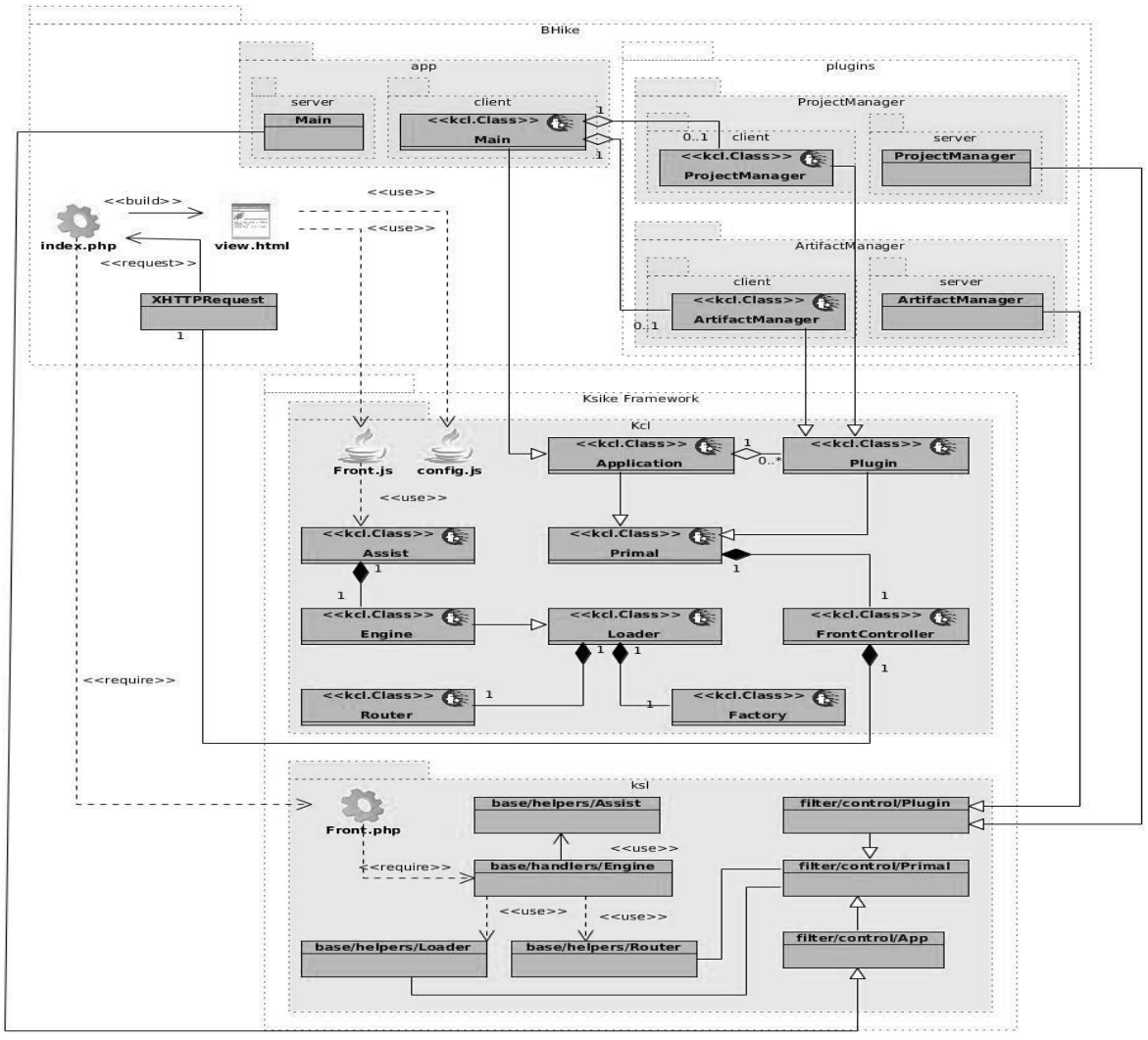


Figura 4.2.1: Diagrama de clases del diseño de BHike (Vista General).

En el modelo presentado anteriormente en la Figura 4.2.1 se evidencia el diseño de una aplicación web. En esta, el flujo de navegación se inicia por la página servidora **index.php** que se encarga de construir la página cliente **view.html**. Se denota además la existencia de una arquitectura basada en componentes, entre los cuales se encuentran: el módulo de aplicación (***app***), el módulo de gestión de

componentes funcionales (**ProjectManager**) y el módulo de gestión de artefactos (**ArtifactsManager**). Cada uno de estos se aferra a la arquitectura cliente-servidor haciendo uso de varios recursos propuestos por el *framework* Ksike, haciéndose hincapié en las clases ***kcl.Class***[52] y las clases servidoras que aparecen en el modelo.

También es el caso del diagrama de clases del diseño asociado al Caso de Uso: Gestionar Componente Funcional, que se ilustra en la Figura 4.2.2, quien generaliza varios Casos de Uso como son: Gestionar Proyecto, Gestionar Aplicación, Gestionar *Plugin*/Módulo y otros que fueron descritos con anterioridad. Este diagrama de clases, es aplicable al flujo descrito de forma extendida en el epígrafe 3.3.4 del capítulo anterior.

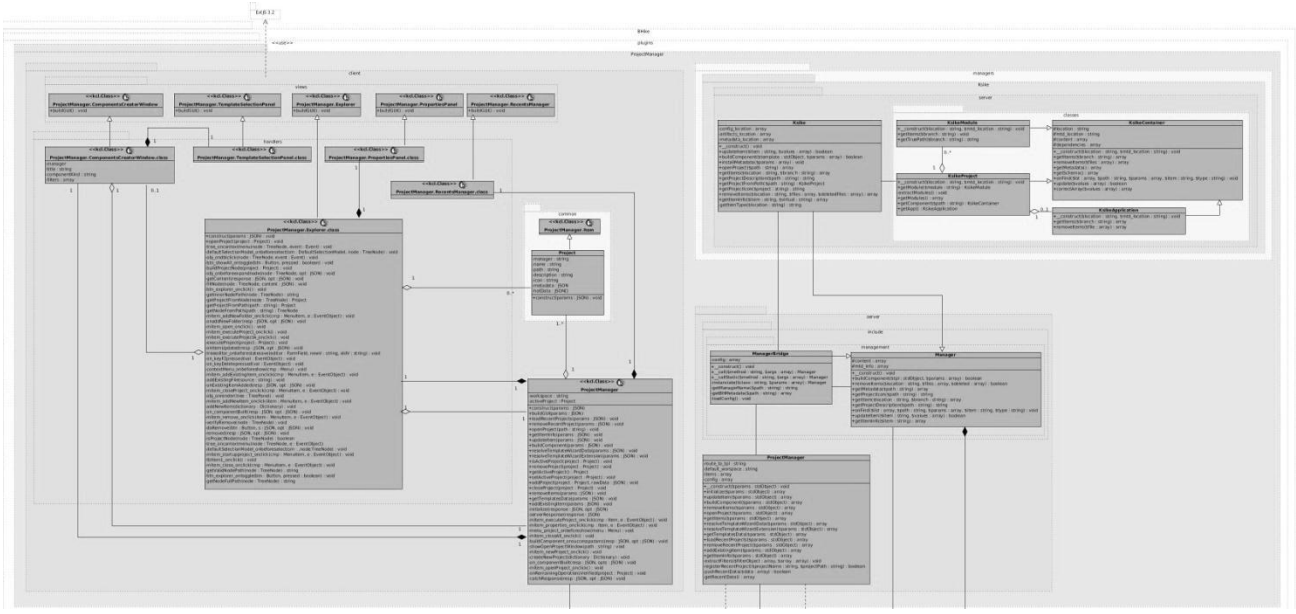


Figura 4.2.2: Diagrama de clases del diseño del CU Gestionar Componente Funcional.

Como se ha analizado anteriormente en otros casos, el diagrama está dividido en un paquete cliente y uno servidor según define el modelo arquitectónico que se sigue. El cliente implementa el patrón Presentación Desacoplada por lo que la pueden encontrarse varias clases asociadas a la Vista de Presentación (VP), Lógica de Presentación (LP) y Lógica de Negocio (LN). Dentro de las clases asociadas a la VP, se encuentran las clases: ComponentsCreatorWindow, TemplateSelectionPanel, PropertiesPanel, Explorer y RecentsPanel, todas implementadas con el esquema del *kcl.class*.

Por otra parte implementando la LP asociada a las vistas definidas se encuentran las clases: ComponentsCreatorWindow.class, TemplateSelectionPanel.class, PropertiesPanel.class, Explorer.class y RecentsManager.class respectivamente, estas últimas ofrecen una extensión de las capacidades de las VP. Para completar el trípode definido en el sub-paquete asociado al cliente, asociados a la LN se encuentran las clases ProjectManager.Item, ProjectManager.Project y la clase controladora ProjectManager. Esta última extiende de la clase Plugin de la API cliente de Ksike y está relacionada por composición con las clases: ComponentsCreatorWindow.class y

[52] **Kcl.Class:** Esquema de clases de la API de JavaScript propuesta por Ksike.

RecentsManager.class. Además constituye el principal ente de control del lado cliente, que se asocia al módulo Gestión de Componentes Funcionales.

El sub-paquete servidor contiene las clases servidoras asociadas al módulo, que se encargan de dirigir la LN para darle respuestas a las solicitudes del cliente. En este se localizan las clases ProjectManager, ManagerBridge y Manager. La primera extiende de la clase Plugin del API servidora que propone Ksike y para delegar el procesamiento agrega la clase ManagerBridge que funciona como puente entre cada tipo de gestor para ordenar la realización de las operaciones requeridas como puede apreciarse en la Figura 4.2.3 a continuación.

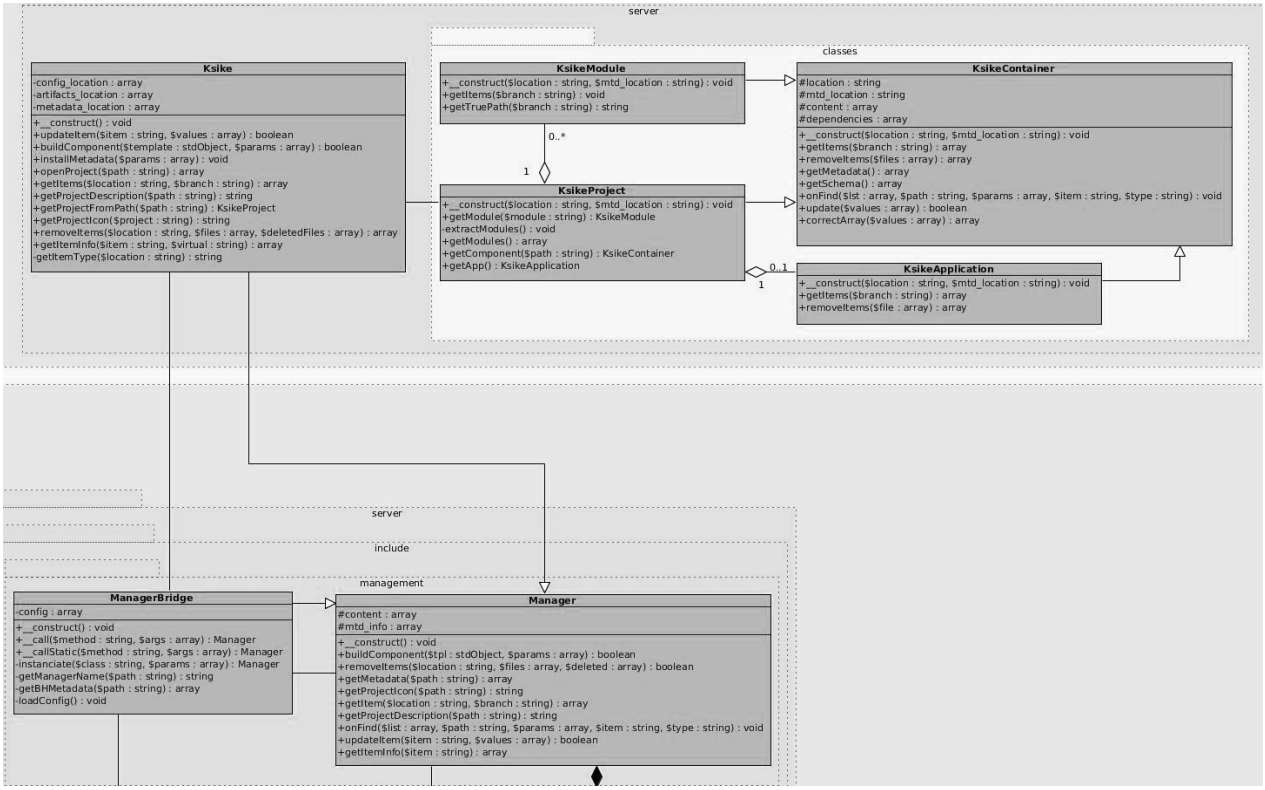


Figura 4.2.3: Gestor de los proyectos Ksike (lado servidor).

Entre los gestores, como se ilustra en la Figura 4.2.3, se encuentran la clase Manager y Ksike quien es una especialización de la primera, lo que le permite a BHike incluir varios tipos de gestores de forma que pueda ser extensible a otras definiciones de proyecto, no solo para Ksike, sino para algún otro que necesite ser definido. Para más información se recomienda hacer una lectura del Artefacto de Ingeniería: Modelo del diseño.

4.2.2. Patrones de diseño

Para la elaboración de los diagramas de clases del diseño debido a la complejidad del mismo y a varias situaciones problemáticas asociadas a este proceso, fueron tomados en consideración disímiles patrones de diseño que serán analizados brevemente a continuación. Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones

entre ellos. Describe una estructura comúnmente recurrente en la comunicación de los componentes que resuelve un problema general de diseño en un contexto particular (Larman).

4.2.2.1. Patrones GRASP[53]

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (Prácticas de Software). Dentro de los utilizados en el desarrollo del sistema se encuentran los siguientes:

Experto en información: Este patrón responde al principio básico de asignación de responsabilidad (Larman). El mismo se aplica en varias clases y puede evidenciarse en la clase Ksike, mostrada en la Figura 4.2.3. Esta tiene asignada la responsabilidad de gestionar la información sobre los proyectos, aplicaciones y módulos de Ksike pues es esta clase, la que contiene la información necesaria para gestionar los componentes mencionados.

Creador: El patrón creador responde a la interrogante de ¿Quién crea? Para esto asigna a una clase B la responsabilidad de instanciar a la clase A, si cumple con alguna de las siguientes condiciones (Larman):

- B contiene A.
- B agrega A.
- B registra A.
- B utiliza A estrechamente.

Este patrón se puede presenciar en la relación que existe entre las clases KsikeProject y KsikeApplication o KsikeProject y KsikeModule, mostrado anteriormente en el diagrama de la Figura 4.2.3, pues la clase KsikeProject agrega a las clases KsikeApplication y KsikeModule.

Controlador: El patrón controlador propone una respuesta a la interrogante de: ¿Quién administra un evento del sistema? Para lograr esto, se le asigna la responsabilidad a una clase que se defina como una de las siguientes opciones (Larman). Este es el caso de la clase KsikeProject quien actúa como fachada para el acceso a las operaciones de las instancias de KsikeApplication y KsikeModule, como puede evidenciarse en la Figura 4.2.3 mostrada anteriormente.

Bajo Acoplamiento: Este patrón brinda una solución al problema de ¿Cómo lograr que en caso de producirse una modificación en alguna de las clases, se tenga la mínima repercusión posible en el resto de ellas? Para garantizar esto, el patrón se define como una medida de la fuerza con que una clase está conectada a las demás (SlideShare). El *driver* BHBuild fue desarrollado para disminuir el acoplamiento del sistema, pues cada una de las tareas que en este se implementan garantizan la aplicación del patrón bajo acoplamiento al resto del sistema. Para más detalles ver el Artefacto de Ingeniería: Modelo del diseño en la página 12.

[53] **GRASP:** Acrónimo de *General Responsability Assignament Software Patterns*, en español Patrones Generales de Software para Asignación de Responsabilidades.

Alta Cohesión: El patrón Alta Cohesión es un principio que se debe tener presente en todas las decisiones de diseño, es la meta principal que ha de buscarse en todo momento. La cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase, además de que una alta cohesión garantiza que clases con responsabilidades estrechamente relacionadas no realicen un trabajo enorme y que cada elemento del diseño debe realizar una labor única dentro del sistema (SlideShare). Se aplica también en el *driver* BHBuild para la realización de varias tareas como son la creación, copia, eliminación y demás operaciones que se realizan sobre ficheros

Polimorfismo: Este patrón se emplea cuando existen variaciones en el comportamiento según el tipo de objeto que se evalúe. Por consiguiente define la asignación de responsabilidades mediante el uso de operaciones polimórficas a los tipos en los que varía el comportamiento (Larman). La clase BHBuild_Task_DeployTemplateContent del *driver* BHBuild implementa chequeos sobre la variación del tipo y asigna la responsabilidad haciendo uso de métodos polimórficos como se puede apreciar a continuación en la Figura 4.2.4.

```
foreach($tplContent as $deployable)
    if($deployable instanceof BHBuild_Model_IDeployable) {
        if($deployable instanceof BHBuild_Model_File)
            $deployable->setPath($this->destination.$deployable->getTargetFileName());
        else
            $deployable->setPath($this->destination);
```

Figura 4.2.4: Aplicación del patrón polimorfismo en la clase BHBuild_Task_DeployTemplateContent.

Indirección: Este patrón asigna la responsabilidad a un objeto intermedio para que medie entre otros componentes o servicios, de modo que no se acoplen directamente. En el diseño presentado se evidencia en la clase ManagerBridge quien funciona de intermediario entre las clases ProjectManager y cada uno de los gestores implementados: Manager y Ksike, como se ilustra en las Figuras 4.2.2 y 4.2.3.

4.2.2.2. Patrones GOF[54]

Los patrones GOF son una serie de buenas prácticas asociadas al diseño de software que brinda un conjunto de tres clasificaciones fundamentales según definen los autores del libro *Design Patterns.* Las mismas se dividen en creacionales, estructurales y de comportamiento, de los cuales fueron empleados varios en el diseño del sistema propuesto.

Entre los patrones creacionales se emplearon:

***Builder*:** El patrón constructor abstrae la construcción de un objeto complejo de su representación, de forma tal que el mismo proceso de construcción pueda crear diferentes representaciones. Entre las ventajas asociadas se encuentran que el mismo reduce el acopamiento, permite un mayor control en el proceso de creación de una instancia e independiza el código que construcción de la representación

[54] **GOF:** Acrónimo de *Gang of Four,* en español Pandilla de los cuatro.

(Gamma, y otros, 1997). El patrón define cuatro agentes fundamentales vinculados al proceso de construcción de objetos: Director, Constructor abstracto, Constructor concreto y Producto. Ocupan estos roles las clases: BHBuild_Executer, Builder, BHBuild_Pattern_Builder y BHBuild_Task respectivamente como se aprecia en la sección C del diagrama de clases asociado a la realización del Caso de Uso Gestionar Componente Funcional representado en el Modelo del diseño.

***Singleton*:** Este patrón garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único (Gamma, y otros, 1997). En el diagrama de clases del diseño asociado a la gestión de componentes funcionales, todos los gestores implementan un *Singleton*, pues solo una instancia es requerida para procesar la información asociada a los proyectos.

Entre los patrones estructurales se emplearon:

Adapter: El patrón adaptador se utiliza para transformar una interfaz A en otra B, de tal modo que una clase que no pudiera utilizar la interfaz A, haga uso de ella a través de B. Esto garantiza que varias clases cooperen entre sí, pues de otra manera no podrían hacerlo debido a la incompatibilidad de sus interfaces (Gamma, y otros, 1997). La implementación de *drivers* en Ksike constituye básicamente en definir una interfaz adaptadora para acceder a un conjunto de clases o interfaces independientes del marco de trabajo, esa responsabilidad recae en la clase BHBuild del *driver* de mismo nombre.

Bridge: El patrón puente desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente sin necesidad de alterar por ello la otra. Este se aplica en varias situaciones en las que se desea evitar un enlace permanente entre la abstracción y su implementación. Además si existe la restricción de que, tanto las abstracciones como sus implementaciones deben ser extensibles por medio de subclases y los cambios en la implementación de una abstracción no deben impactar en los clientes, es decir, su código no debe tener que ser recompilado (Gamma, y otros, 1997). En el Módulo de Gestión de Componentes Funcionales se puede evidenciar el empleo de una clase con función de puente, como es el caso de ManagerBridge, quien resulta ser el agente abstracto mientras que las clases Manager y Ksike son las implementaciones del mismo.

Entre los patrones de comportamiento se emplearon:

Chain of Responsability: El patrón cadena de responsabilidades evita acoplar el emisor de una petición a su receptor, dando a más de un objeto la posibilidad de responder a una petición. Para ello, se encadenan los receptores y la petición pasa a través de la cadena hasta que es procesada por algún objeto. Este patrón es utilizado a menudo en el contexto de las interfaces gráficas de usuario donde un objeto puede contener varios objetos. Según el ambiente de ventanas genera eventos, los objetos los manejan o los pasan (Gamma, y otros, 1997). En la llamada que se efectúa sobre la

notificación de los módulos del sistema a través del método “notify” cuando se ordena al módulo Main realizar notificaciones a los módulos ante la ocurrencia de un evento como se ilustra en la Figura 4.2.5.

```
notify : function(comp){
    for(var i in std.mod)
        if(std.mod[i].notified != null)
            std.mod[i].notified(comp);
}
```

Figura 4.2.5: Ejecución de una llamada en cadena.

Observer: El patrón observador define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes (Gamma, y otros, 1997). Este patrón suele observarse en los *frameworks* de interfaces gráficas orientados a objetos, en los que la forma de capturar los eventos es suscribir *listeners* a los objetos que pueden disparar eventos. El *framework* empleado para el diseño de la interfaz gráfica ExtJS, permite subscribir varias funciones a uno o múltiples eventos asociados a los componentes visuales. Esta característica es aprovechada por varios módulos del sistema que están a la expectativa del evento *click* asociado al botón Guardar que provee la barra de herramienta principal del módulo de aplicación.

4.3. Diagrama de despliegue

Según el sitio web www.visual-paradigm.com, el diagrama de despliegue se emplea para modelar el aspecto físico de un sistema de software orientado a objetos. Estos permiten modelar, en una vista estática, la configuración del sistema en tiempo de ejecución y visualizar la distribución de los componentes de una aplicación. En la mayoría de los casos, los diagramas de despliegue modelan la configuración del hardware junto con los componentes de software que se alojan en este (Visual Paradigm). Este diagrama es de gran utilidad para los sistemas que poseen diversos entornos de ejecución, por lo que es aplicable a las soluciones informáticas que presentan arquitectura de sistemas distribuidos y cliente-servidor como es el caso de la solución que se construye.

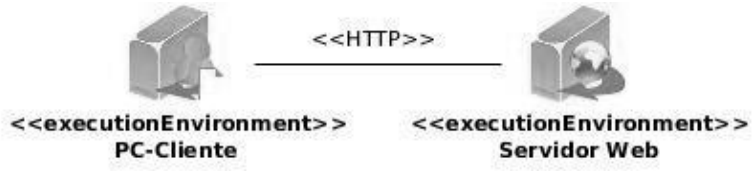


Figura 4.3.1: Diagrama de despliegue de BHike.

La Figura 4.3.1 mostrada anteriormente, refleja la existencia de los principales entornos de ejecución que requiere BHike para su correcto funcionamiento, los cuales son descritos a continuación:

Nodo PC Cliente: Es el entorno a través del cual los clientes del sistema se conectan y operan con el mismo. Este computador deberá contar con un navegador similar a Chromium versión 9+ o algún otro que cumpla con los estándares W3C e incluya soporte para CCS3 y HTML5.

Nodo Servidor Web: Este deberá tener instalado como Sistema Operativo: Windows XP o superior o recomendablemente GNU/Linux Ubuntu 11.04 o superior. Además deberá contar con el servidor de aplicaciones web Apache 2.0 o superior y el módulo PHP 5.3 o superior.

4.4. Diagrama de componentes (DC)

El DC ayuda a modelar el aspecto físico del sistema, ilustrando la arquitectura de los componentes de software, que pueden ser de: tiempo de ejecución, ejecutables, así como de código fuente y las relaciones entre ellos (Visual Paradigm). La herramienta BHike está formada por varios paquetes de componentes que emplean, en su gran mayoría, los recursos que provee el *framework* Ksike tal y como se aprecia en la Figura 4.4.1. Entre los más notorios se encuentran el paquete ***core, lib, app*** y ***plugin***. El paquete ***core*** contiene código fuente que especializa el comportamiento de Ksike según las necesidades de la herramienta, mientras que en ***lib*** se incluye el código fuente de las bibliotecas o componentes externos que se integran al sistema. El fichero ***index.php*** por otra parte, contiene el código fuente asociado al PAC definido para la aplicación.

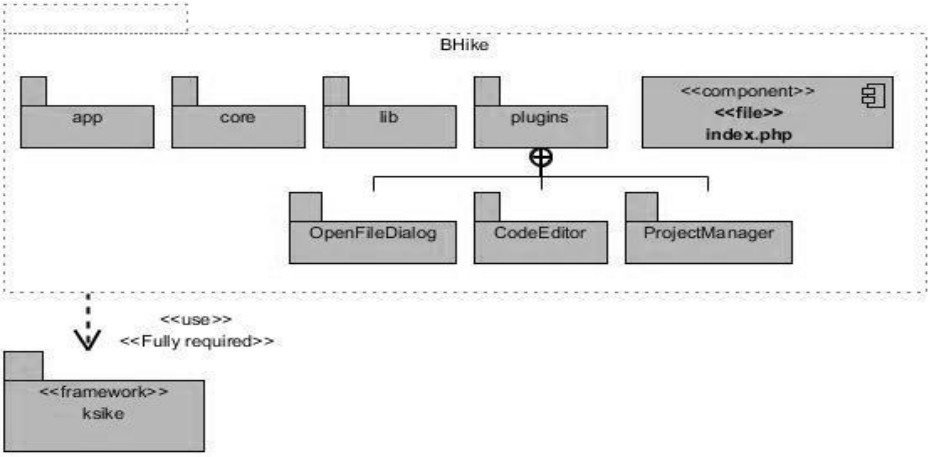


Figura 4.4.1: Diagrama de paquetes de BHike (Vista General).

En el caso particular, tanto del paquete ***app*** como de los sub paquetes contenidos en ***plugin*****:** OpenFileDialog, CodeEditor y ProjectManager, definen una estructura física muy similar, lo cual se debe a que los mismos son módulos de Ksike, como se refleja en la Figura 4.4.2 a continuación:

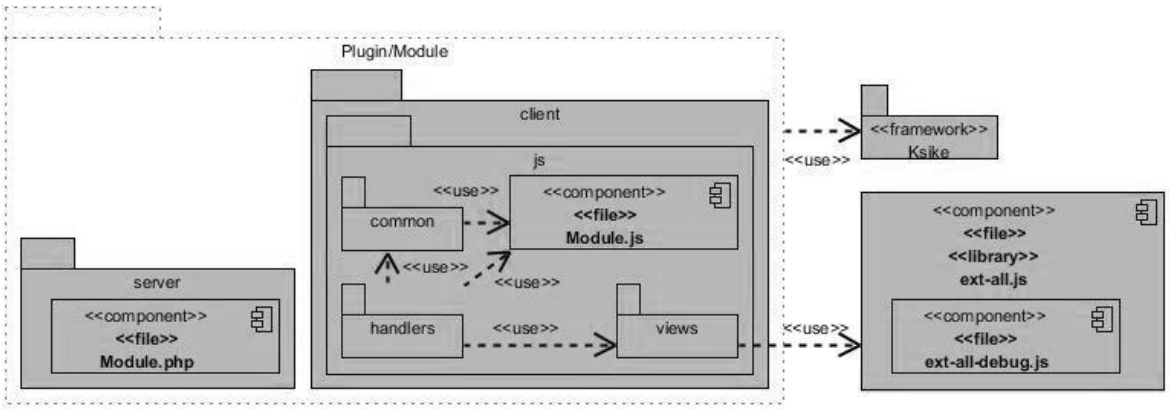


Figura 4.4.2: Diagrama de componentes de plugins/módulos de Ksike.

Cada módulo de BHike contiene un paquete ***server*** que posee un fichero de código fuente ***(Module).php***[55] que es el componente principal del lado servidor y además un paquete ***client***. Este último a su vez posee un paquete ***js*** que contiene el código fuente en el lenguaje JavaScript asociado al lado cliente. El mismo está distribuido en los sub-paquetes: ***common***, ***handlers*** y ***views***. En ***common*** se almacena el código fuente asociado al negocio del lado cliente; los ficheros contenidos en este paquete usan recursos de funcionalidades del fichero ***(Module).js***[56] al igual que el paquete ***handlers***.

En este se agrupa el código fuente las clases encargadas de manejar la lógica de presentación que están conformados por pares a razón de ***View.class.js***[57] **-** ***View.js***[58], de forma tal que cada componente de vista (***View.js***) tiene asociado un manejador (***View.class.js***). Por último es imprescindible mencionar que los módulos hacen uso del *framework* Ksike, así como las vistas emplean los componentes visuales de la biblioteca ExtJS.

4.5. Pruebas del sistema

El objetivo de realizarle pruebas al sistema es revelar la existencia de operaciones incorrectas o incompletas que afectan el funcionamiento del mismo. Las pruebas se pueden ejecutar durante toda la construcción del sistema, pero mayormente son realizadas cuando termina la implementación para probar lo que se ha realizado. Entre los casos de prueba se puede distinguir dos tipos comúnmente utilizados:

Pruebas de caja negra: verifican el resultado de la interacción entre los usuarios y el sistema, comprobando que se cumplan las pre-condiciones y pos-condiciones especificadas para cada caso de uso siguiendo la secuencia de acciones previstas para el mismo.

Pruebas de caja blanca: son las encargadas de comprobar el comportamiento de las interacciones de los componentes internos del sistema.

En el caso del sistema que se entrega como culminación de esta investigación, se propone que se le realice pruebas de caja negra a cada caso de uso para que se pueda apreciar que cumplen con las precondiciones y pos-condiciones especificadas para cada uno. Debido a que varios de los casos de uso presentados en el presente trabajo son similares, a continuación se muestra el caso de prueba realizado para el CUS Gestionar Proyecto.

4.5.1. Descripción general

El caso de uso se inicia cuando el usuario selecciona la opción correspondiente en la interfaz principal, por alguno de los caminos descritos en el Modelo de Casos de Uso del Sistema y concluye cuando el sistema guarda los datos y notifica los resultados.

[55] ***(Module).php*** Fichero que recibe el nombre del módulo y es el componente fundamental del lado servidor.
[56] ***(Module).js*** Fichero que recibe el nombre del módulo y es el componente fundamental del lado cliente.
[57] ***View.class.js*** Ejemplo de componente de software asociado a la lógica de presentación que se almacena en el paquete *handlers*.
[58] ***View.js*** Ejemplo de componente de software asociado a las vistas de presentación que se almacena en el paquete *views.*

4.5.2. Secciones a probar

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad	Flujo Central
SC-1: "Crear Proyecto"	EC-1.1: "Crear proyecto con éxito".	El usuario selecciona la opción *Add New Project* y el sistema muestra la interfaz de usuario correspondiente para la introducción de los siguientes datos: • Nombre • Ubicación	Crear Nuevo Proyecto: 1. El usuario selecciona la plantilla de proyecto. 2. El sistema muestra la descripción de la plantilla. 3. El usuario introduce los datos del proyecto. 4. Presiona el botón *Finish*. 5. El sistema crea el proyecto. 6. El sistema abre el proyecto en el explorador de artefactos.
	EC-1.2: "Crear proyecto sin éxito"	El usuario deja campos vacíos.	Crear Nuevo Proyecto: 1. El sistema muestra un mensaje indicando que hay campos requeridos en el formulario y deshabilita el botón *Finish*.
	EC-1.3: "Crear proyecto cancelado"	El usuario selecciona la opción *Add New Project* y el sistema muestra la interfaz de usuario correspondiente.	Crear Nuevo Proyecto: 1. El usuario presiona el botón *Cancel*. 2. El sistema cancela la creación del proyecto y cierra la ventana *Add New Project*.
SC-2: "Abrir Proyecto"	EC-2.1: "Abrir proyecto con éxito".	El usuario selecciona la opción *Open Project*, navega hasta la dirección donde se encuentra alojado el proyecto y presiona el botón *Open* o presiona doble *click* sobre el proyecto.	Abrir Proyecto: 1. El sistema muestra una ventana de navegación por el sistema de ficheros. 2. El usuario busca y selecciona el proyecto deseado. 3. Presiona el botón *Open*. 4. El sistema almacena el proyecto en el registro de proyectos recientes. 5. Carga la información del proyecto en el Explorador de Artefactos.
	EC-2.2: "Abrir proyecto cancelado".	El usuario selecciona la opción *Open Project*.	Abrir Proyecto: 1. El sistema muestra una ventana de navegación por el sistema de ficheros. 2. El usuario presiona el botón *Cancel*. 3. El sistema cancela la navegación y cierra la ventana *Open Project*.

SC-3: "Abrir Proyecto Reciente"	EC-3.1: "Abrir Proyecto Reciente con éxito"	El usuario selecciona uno de los proyectos del panel *Recent Projects*.	Abrir Proyecto Reciente: 1. El sistema verifica la existencia del proyecto. 2. El sistema carga la información del proyecto seleccionado en el Explorador de Artefactos.
	EC-3.2: "Abrir Proyecto Reciente sin éxito"	El usuario selecciona uno de los proyectos del panel *Recent Projects*.	Abrir Proyecto Reciente: 1. El sistema verifica la existencia del proyecto y no logra ubicarlo. 2. El sistema informa al usuario que el proyecto no existe y brinda la posibilidad de eliminar la referencia del panel *Recent Projects*. 3. El usuario presiona el botón *OK*. 4. El sistema elimina la referencia.
SC-4: "Modificar Información de Proyecto"	EC-4.1: "Modificar información con éxito."	En el Explorador de Artefactos el usuario selecciona un proyecto y ppresiona *click* sobre el botón de propiedades o la opción "*View Properties"* del menú contextual.	Modificar Información de Proyecto: 1. El sistema muestra el panel "*Properties*" que posee una tabla que contiene la información asociada al proyecto. 2. El usuario modifica los parámetros que desea siempre que los mismos sean editables y presiona la tecla *Enter*. 3. El sistema hace persistente la información y notifica al usuario de la realización de los cambios.
	EC-4.2: "Modificar información sin éxito."	En el Explorador de Artefactos el usuario selecciona un proyecto y ppresiona *click* sobre el botón de propiedades o la opción "*View Properties"* del menú contextual.	Modificar Información de Proyecto: 1. El sistema muestra el panel "*Properties*" que posee una tabla que contiene la información asociada al proyecto. 2. El usuario deja campos en blanco. 3. Muestra un mensaje indicando que los valores son requeridos y brinda al usuario

			la posibilidad de corregir la deficiencia.
SC-5: "Cerrar Proyecto"	EC-5.1: "Cerrar Proyecto con éxito."	En el Explorador de Artefactos el usuario selecciona la opción *Close Project* sobre un proyecto abierto.	Cerrar Proyecto: 1. El sistema guarda el estado actual de las operaciones del proyecto y cierra el mismo.
SC-6: "Eliminar Proyecto"	EC-6.1: "Eliminar Proyecto con éxito."	En el Explorador de Artefactos el usuario selecciona la opción *Remove* sobre un proyecto abierto.	Eliminar Proyecto: 1. El sistema brinda al usuario una ventana de confirmación con las opciones de *Yes* y *No*. 2. El usuario presiona el botón *Yes*. 3. El sistema elimina la información del proyecto. 4. El sistema informa el éxito de la operación.
	EC-6.2: "Eliminar Proyecto cancelado."	En el Explorador de Artefactos el usuario selecciona la opción *Remove* sobre un proyecto abierto.	Eliminar Proyecto: 1. El sistema brinda al usuario una ventana de confirmación con las opciones de *Yes* y *No*. 2. El usuario presiona el botón *No*. 3. El sistema cancela la operación.

Tabla 4.5.1: Diseño de Caso de Prueba del CUS Gestionar Proyecto.

4.5.3. Descripción de variables

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	Nombre	Campo de Texto	No	Nombre requerido para generar un nuevo proyecto.
2	Ubicación	Campo de Texto	No	Ubicación requerida para alojar un proyecto nuevo.

Tabla 4.5.2: Descripción de las variables asociadas al diseño de pruebas del CUS Gestionar Proyecto.

4.5.4. Matriz de datos

4.5.4.1. SC-1: Crear Proyecto

ID del Escenario	Nombre	Ubicación	Respuesta del sistema	Resultado de la prueba
EC-1.1	V	V	Se crea el proyecto	Satisfactorio
	V	I	El sistema muestra un mensaje de error.	Satisfactorio
	I	V	El sistema muestra un mensaje de error.	Satisfactorio
	I	I	El sistema muestra un mensaje de error.	Satisfactorio

Tabla 4.5.3: Matriz de datos de la sección Crear Proyecto.

4.6. Resultados de las pruebas

Los Diseños de Casos de Pruebas del sistema mostrados anteriormente están definidos según las especificaciones del método de particiones equivalentes. Luego de elaborados, los mismos fueron aplicados a cada uno de los Casos de Uso implementados, con el objetivo de identificar posibles irregularidades en la herramienta. Después de haberse realizado 2 iteraciones se arrojó la existencia de 35 no conformidades, entre las que se listan las fundamentales a continuación:

- ✓ Inexistencia de notificaciones ante campos en blanco.
- ✓ Botones sin funcionalidad.
- ✓ Validaciones no funcionales ante entradas de datos incorrectos.
- ✓ Nombre de paneles y ventanas que no coincidían con los descritos en el Modelo de Casos de Uso.
- ✓ No actualización de los componentes visuales ante cambios en la información.
- ✓ Inexistencia de funcionalidades claves del sistema.

Gracias a la realización satisfactoria de las pruebas al sistema, se pudieron identificar y eliminar las deficiencias detectadas garantizando que la solución cumpliera con los requisitos funcionales descritos originalmente. Este elemento se comprobó con la realización de una tercera iteración que no reveló nuevas no conformidades, tal y como se ilustra a continuación en la Figura 4.6.1:

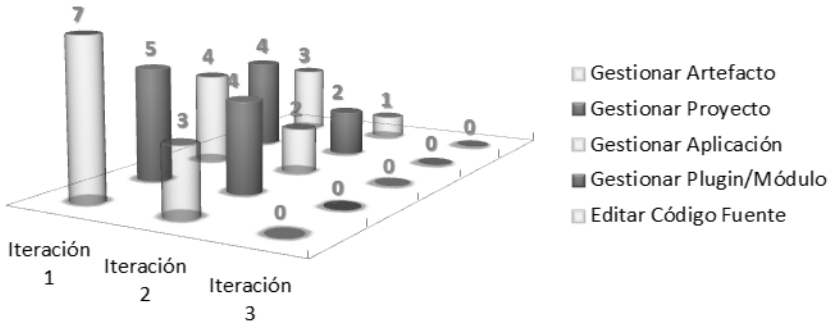


Figura 4.6.1: Gráfico de no conformidades por casos de uso por iteraciones de prueba.

4.7. Conclusiones parciales

En el presente capítulo se han arribado a varias conclusiones importantes, entre ellas se destaca la necesidad de garantizar la consistencia del sistema ante posibles cambios de equipo de desarrollo o de enfoque de solución. Para lograr esto, es imprescindible apoyarse en un diseño consistente que responda al mayor número posible de modificaciones sin afectar el funcionamiento íntegro de la aplicación. Otro de los elementos a tener en cuenta es que: los procesos de Ksike serán automatizados parcialmente en el transcurso de tiempo, por lo que el sistema debe proveer una arquitectura que se flexibilice ante la adición de nuevas funcionalidades. El sistema además debe brindar la posibilidad al usuario de especificar sus propios componentes, que en gran medida puede ser aplicable para automatizar los procesos sobre otras tecnologías. Es indispensable además, hacer uso exhaustivo de varios patrones de diseño en aras de fomentar la reutilización y disminuir el acoplamiento de los componentes desarrollados.

CONCLUSIONES GENERALES

El desarrollo de la presente investigación se obtuvo como resultado una herramienta para automatizar los procesos asociados al desarrollo de software que se sustenta en el *framework* Ksike. La misma, aunque aún está en un estado básico, implementa varios de los requisitos funcionales que fueron identificados para darle solución a la situación problemática. Además se evidenció que el desarrollo de mecanismos que posibiliten la automatización de los procesos fundamentales de Ksike, debe ser un proceso paulatino y complejo, pues la tecnología está aún en estado de evolución. Se ha construido una plataforma que ofrece un ambiente común para la integración de nuevas funcionalidades basándose en la arquitectura de componentes.

El desarrollo de la aplicación estuvo guiado por una metodología de desarrollo de software que posibilitó la generación de una documentación abundante y muy necesaria para el equipo de desarrollo del *framework,* pues esto permite suplir una de las limitantes que posee dicha tecnología: la falta de constancia documental. Es la primera vez además que se realiza una investigación a fondo enfocada en identificar los procesos automatizables de Ksike. De manera general se le dio cumplimiento al objetivo planteado y actualmente la herramienta se encuentra a disposición de la comunidad de desarrollo de la Universidad de las Ciencias Informáticas.

RECOMENDACIONES

BHike está lejos aún de convertirse en un IDE, pues no contempla los procesos de interpretación, compilación ni depuración asociados a los lenguajes de programación que emplea el *framework* Ksike, es por esto que se recomienda brindarle soporte en ese sentido. Enfatizando directamente sobre los procesos del marco de trabajo, se han documentado en el presente trabajo varios que deben ser automatizados entre los que se encuentran: Gestión de *Logs* y Errores, Gestión de Publicaciones, Gestión de Enlaces, Gestión de Configuración y Gestión de Dependencias. De igual manera se ha identificado la oportunidad potencial de integrar a BHike las soluciones Lycan y Caxtor desarrolladas en el departamento DATEC de la facultad 6 de la Universidad de las Ciencias Informáticas. Estas últimas, son soluciones web enfocadas al diseño de componentes visuales para las tecnologías ExtJS, que integradas a BHike, amplían el marco operacional del mismo, aumentando su usabilidad en la comunidad de desarrollo de la UCI.

BIBLIOGRAFÍAS

Ambler, Scott. 2005. AmbySoft. *El Agile Unified Process (AUP).* [En línea] AmbySoft, 2005. [Citado el: 20 de Marzo de 2013.] http://www.ambysoft.com/unifiedprocess/agileUP.html.

Apache Software Foundation. 2011. Apache 2.2. *Visión general de las nuevas funcionalidades de Apache 2.0.* [En línea] Apache Software Foundation, 2011. [Citado el: 23 de Noviembre de 2012.] http://httpd.apache.org/docs/2.0/new_features_2_0.html.

Arias Marin, Marvin David. 2008. Definición de lenguaje de programación. Tipos. Ejemplos. *catedraprogramacion.* [En línea] 16 de Octubre de 2008. [Citado el: 29 de Noviembre de 2011.] http://catedraprogramacion.foroactivo.net/t83-definicion-de-lenguaje-de-programacion-tipos-ejemplos.

2012. Asignatura: Entornos de programación. *Entornos de programación: Concepto, funciones y tipos.* [En línea] 27 de 11 de 2012. http://lml.ls.fi.upm.es/ep/entornos.html.

Bahit, Eugenia. *OOP y MVC en PHP.*

Blanco, Carlos. 2011. CarlosBlanco.pro. *Entornos de Desarrollo Integrado (IDE´s) – Introducción.* [En línea] 8 de Abril de 2011. [Citado el: 8 de Enero de 2013.] http://carlosblanco.pro/category/prog-apli/.

Bolton, David. About.com. [En línea] [Citado el: 8 de Enero de 2013.] http://cplus.about.com/od/glossary/g/gloscompiled.htm.

—. About.com. *Definition of Interpreter.* [En línea] About.com Guide. [Citado el: 8 de Enero de 2013.] http://cplus.about.com/od/introductiontoprogramming/g/interpreterdefn.htm.

Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel. 2004. *Arquitecturas de Software, Guía de estudio.* 2004.

CAMACHO, ERIKA, CARDESO, FABIO y NUÑEZ., GABRIEL. 2004. *ARQUITECTURAS DE SOFTWARE GUÍA DE ESTUDIO.* 2004.

Clifton, Marc. 2003. Code Project. *What Is A Framework?* [En línea] Code Project, 3 de Noviembre de 2003. [Citado el: 11 de Diciembre de 2012.] http://www.codeproject.com/Articles/5381/What-Is-A-Framework.

Cloud9IDE. github social coding. [En línea] https://github.com/ajaxorg/cloud9; http://www.cloud9ide.com.

Codebox. CODEBOX. *Glosario.* [En línea] Codebox. [Citado el: 11 de Diciembre de 2012.] http://www.codebox.es/glosario.

Córdoba, Fernando García. 2004. *La tesis y el trabajo de tesis.* México : Limusa, 2004. 968-18-6235-X.

de la Torre LLorente, César, y otros. 2010. *Guía de Arquitectura N-Capas Orientada al Dominio con .net 4.0 (Beta).* Madrid : Krasis Consulting, 2010. 978-84-936696-3-8.

Deitel. 2006. *Java TM How to Program, Seventh Edition.* New Jersey : Prentice Hall, 2006. 9780136085676.

EcuRed. EcuRed. [En línea] [Citado el: 6 de Diciembre de 2012.] http://www.ecured.cu/index.php/Aplicaci%C3%B3n_inform%C3%A1tica.

Eguiluz, Javier. 2010. *Desarrollo web ágil con Symfony2.* Madrid, España : s.n., 2010.

Espinosa, Antonio Membrides y Calderín, Angel Rilder Salazar. 2008. *Soluciones Libres para Aplicaciones Multimedia basadas en C++.* La Habana : UCI, 2008.

eXo IDE. 2012. *eXo IDE User Guide.* [pdf] 2012.

Festival Latinoamericano de instalación de software Libre: PHP. **Herrera, Ing. Gerardo. 2006.** Venezuela : FLISOL, 2006.

Floristán, Liset Cabrera. 2012. *Módulo de Comunicación para el Sistema de Control de Flotas.* La Habana : UCI, 2012.

Frederick, Shea, Ramsay, Colin y Blades, Steve. 2008. *Learning Ext JS.* [pdf] Birmingham, Mumbai : Packt Publishin, 2008. ISBN 978-1-847195-14-2.

Gamma, Erich, y otros. 1997. *Design Patterns Elements of Reusable Object-Oriented Software.* s.l. : KevinZhang, 1997.

García de Jalón, Javier, Rodríguez, José Ignacio y Sarriegui, José María. Abril 1998. *Aprenda C++ como si estuviera en primero.* San Sebastían : Universidad de Navarra, Abril 1998.

Geotrygon, desarrollo de SIG sobre el framework Ksike. **Membrides Espinosa, Antonio, y otros. 2012.** Habana, Cuba : Publicaciones UCI, 2012.

Gutiérrez., Jorge Antonio Díaz. 2009. *Desarrollo de un IDE libre y multiplataforma para la creación de componentes visuales de ActionScript para Software Educativo: codeDraw.* La Habana : UCI, 2009.

Hardz Website. 2008. PHP HiPeRTeXTo PRe-PRoCeSaDo. *Blog de WordPresscom.* [En línea] 7 de Febrero de 2008. [Citado el: 18 de Noviembre de 2011.] http://hardz.wordpress.com/2008/02/07/php-hipertexto-pre-procesado/.

Hernández, Fernando López. 2006. *Compilar y depurar aplicaciones con herramientas de programación de GNU.* Madrid : s.n., 2006.

Herramientas de desarrollo de ingeniería de software para Linux. **Giraldo, Luis y Zapata, Yuliana. 2005.** s.l. : Monitoria de Ingesoft, 2005.

IT Business Edge. Webopedia. *integrated development environment.* [En línea] IT Business Edge. [Citado el: 08 de Diciembre de 2012.] http://www.webopedia.com/TERM/I/integrated_development_environment.html.

Jacobson, I., Booch, G. y Rumbaugh, J. 2000. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2000.

Jimenez, Monica Talledo. 2008. *GUÍA DE LOS FUNDAMENTOS PARA LA DIRECCIÓN DE PROYECTOS (GUÍA DEL PMBOK) Cuarta edición.* Newtown Square, Pennsylvania 19073-3299 EE.UU : Project Management Institute, 2008. 978-1-933890-72-2.

Kulak, Daryl y Guiney, Eamonn. 2004. *USE CASES REQUIREMENTS IN CONTEXT Second Edition.* Boston : Pearson Education, Inc., 2004. 0-321-15498-3.

Larman, Craig. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.*

Macmillan. [En línea] Macmillan Publishers. [Citado el: 11 de Diciembre de 2012.] http://www.macmillandictionary.com/dictionary/british/framework.

Martinez Alarcón, David y Pupo Leyva, Iliannis. 2010. *Caxtor: constructor de aplicaciones web.* Habana, Cuba : Grupo Editorial Ediciones Futuro, 2010.

Membrides Espinosa, Antonio. 2010. *Manual de Desarrollo del Framework Ksike.* Habana : s.n., 2010.

Microsoft. 2009. *Composite Application Guidance for WPF and Silverlight.* 2009.

—. 2009. *Microsoft Application Architecture Guide 2nd Edition.* 2009. 9780735627109.

—. Microsoft Home Page. [En línea] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/professional/overview.

—. 2011. MSDN. *Proyectos como contenedores.* [En línea] Microsoft, 2011. [Citado el: 14 de 12 de 2012.] http://msdn.microsoft.com/es-es/library/s17bt45e(v=vs.80).aspx.

—. Referencia del esquema de plantillas de Visual Studio. [En línea] http://msdn.microsoft.com/es-es/library/xwkxbww4(v=vs.80).aspx.

Nokia. Qt Creator IDE and tools. [En línea] [Citado el: 13 de Noviembre de 2011.] http://qt.nokia.com/products/developer-tools.

Object Management Group Inc. 2003. *Unified Modeling Language Specification.* Masachussets : OMG, 2003. v1.5.

Object Technology International, Inc. 2003. *Eclipse Platform.* 2003.

Oracle Corporation & afiliates. 2011. NetBeans. [En línea] 2011. [Citado el: 18 de Noviembre de 2011.] http://netbeans.org/community/releases/70/.

Palacio, Juan. 2008. *ScrumManager: Gestión de proyectos.* s.l. : SafeCreative, 2008. 0808180903131.

Pfleeger, Charles P. 2006. *Security in computing.* 2006. 978-0-13-239077-4.

Potencier, Fabien. 2009. *El Tutorial Jobeet.* 2009.
Prácticas de Software. Prácticas de Software. *Experiencias sobre la Ingeniería y Management del Software.* [En línea] [Citado el: 26 de Abril de 2013.] http://www.practicasdesoftware.com/.
Prashant y Gupta, Sarika. Mayo 2012. *Simplifying Use Case Models Using CRUD Patterns.* s.l. : International Journal of Soft Computing and Engineering , Mayo 2012. 2231-2307.
Pressman, Roger. 2002. *Ingeniería del Software: Un enfoque práctico.* España : McGraw-Hill Companies, 2002. 5ta Edición.
ProgramacionDesarrollo.es y editorbfb. 2011. ProgramacionDesarrollo. *Qué es un entorno de desarrollo integrado, IDE.* [En línea] BlogsFarm Network SL, Febrero de 2011. [Citado el: 08 de Diciembre de 2012.] http://programaciondesarrollo.es/que-es-un-entorno-de-desarrollo-integrado-ide/.
Proyecto Lycan. Lycan GENESIS - Component Builder. [En línea] [Citado el: 24 de Septiembre de 2012.] http://comunidades.uci.cu/projects/lycan-ide.
Qt Project. 2012. Qt Project. *Cómo Usar Señales ("Signals") y Ranuras ("Slots")?* [En línea] Qt Project, 17 de Agosto de 2012. [Citado el: 12 de Febrero de 2013.] http://qt-project.org/wiki/How_to_Use_Signals_and_Slots_Spanish.
Real Academia Española. DICCIONARIO DE LA LENGUA ESPAÑOLA - Vigésima segunda edición. [En línea] RAE. [Citado el: 6 de Diciembre de 2012.] http://lema.rae.es/drae/.
Reynoso, Carlos y Kiccillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.
Rouse, Margaret. SearchSoftwareQuality. [En línea] WhatIs.com. [Citado el: 06 de Diciembre de 2012.] http://searchsoftwarequality.techtarget.com/definition/.
Sileika, Rytis. 2010. *Pro Python System Administration.* New York : Apress, 2010. 978-1-4302-2605-5.
SlideShare. Ingeniería en Sistemas de Información. Diseño de sistemas. [En línea] [Citado el: 3 de febrero de 2013.] Disponible en: http://www.slideshare.net/jpbthames/patrones-para-asignar-responsabilidades-grasp.
Stefanov, Stoyan. Julio 2008. *Object-Oriented JavaScript. Create scalable, reusable high-quality JavaScript applications, and libraries.* Birmingham, B27 6PA, UK : Packt Publishing, Julio 2008. 978-1-847194-14-5.
Szyperski, Clemens Alden, Gruntz, Dominik y Murer, Stephan. 2002. *Component Software - Beyond Object-Oriented Programming – Second Edition.* s.l. : Addison-Wesley / ACM Press, 2002. 0-201-74572-0.
TechTarget y Rouse, Margaret. 2007. SearchSoftwareQuality. *integrated development environment (IDE).* [En línea] WhatIs.com, Febrero de 2007. [Citado el: 08 de Diciembre de 2012.] http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment.
TechTerms.com. techterms.com. *Template.* [En línea] TechTerms.com. [Citado el: 11 de Diciembre de 2012.] http://www.techterms.com/definition/template.
The Rational Unified Process: An Introduction. **Kruchten, P. 2000.** s.l. : Addison Wesley, 2000.
Universia. Desarrollo Profesional. *Las universidades como incubadoras de empresas.* [En línea] Universia.es. [Citado el: 02 de Marzo de 2013.] http://desarrollo-profesional.universia.es/programas-de-trainee/universidades-emprendedoras/universidades-incubadoras-empresas/.
Virtual Paradigm International. 1999-2011. Visual Paradigm. *What VP-UML Provides.* [En línea] 1999-2011. [Citado el: 23 de Septiembre de 2011.] http://www.visual-paradigm.com/product/vpuml/provides/.
Visual Paradigm. VP Gallery. [En línea] http://www.visual-paradigm.com/VPGallery/diagrams/Deployment.html.
Vizcaíno, Aurora, García, Félix Oscar y Caballero, Ismael. *Una Herramienta CASE para ADOO: Visual Paradigm.* Ciudad Real : Universidad de Castilla-La Mancha.
Ziller, Eike. 2009. Qt Blog. *Qt Creator 1.0 is out.* [En línea] Qt-project.org, 3 de Marzo de 2009. [Citado el: 18 de Febrero de 2013.] http://blog.qt.digia.com/blog/2009/03/03/qt-creator-10-is-out/.