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 54 de la Revolución"

Frase (opcional)

Dedicatoria (opcional)

Agradecimientos (opcional)

Declaración de autoría

Declaro por este medio que yo, Yadir Hernández Batista con carné de identidad 89122539461, soy el autor de este trabajo y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales 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**

Resumen

El *framework* Ksike se muestra como una variante ideal para el desarrollo de aplicaciones para todo tipo de entornos y plataformas ya que, gracias 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 una manera sencilla para efectuar su aprendizaje o empleo, y más limitante aún, no cuenta con una herramienta que facilite el uso de sus capacidades.

En aras de garantizar un desarrollo ágil, eficiente 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, *Framework*, Entorno Integrado de Desarrollo, Ksike, Software.

ÍNDICE DE CONTENIDOS

ÍNDICE DE IMÁGENES

ÍNDICE DE TABLAS

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, las versátilmente empleadas en la medicina para la realización de operaciones complejas o los disímiles ejemplos de aplicaciones informáticas o software, que son hoy en día una compleja red de herramientas que se extiende a nivel global; todas 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 si misma*" (1), la cual ha aumentado considerablemente y constituye hoy día 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 y mejoran 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, ya que 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, además se rigen 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, lo que 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 sobre 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 tenor del cada vez mayor y más creciente número de marcos de trabajo orientados al desarrollo de software y las abismales diferencias entre los conceptos y filosofías planteados por cada uno; a finales del año 2010 un grupo de ingenieros de la Universidad de las Ciencias Informáticas (UCI) 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 (2).

Ksike propone numerosas ventajas entre las que se pueden 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[1], así como recursos para aplicaciones sobre dispositivos móviles; los cuales pueden ser

[1] **SIG** – Sistemas de Información Geográficos.

de gran aprovechamiento en los proyectos productivos del centro GEYSED[2] de la UCI y otros centros productivos de la misma entidad. Entre otros elementos, propone además generación de componentes para los entornos: *web*, *desktop* y móviles con alta reusabilidad entre ellos.

La Universidad de las Ciencias Informáticas de acuerdo a las características propias de su comunidad, la cual es precursora por excelencia del uso de tecnologías para el desarrollo de software, constituye actualmente uno de los ambientes más propicios en Cuba, para potenciar el desarrollo y empleo de nuevas tecnologías aplicadas al campo de la Informática. Al concebir el *framework* Ksike, en la propia UCI, se estimaba que el mismo fuera explotado a gran escala por los proyectos productivos y la comunidad de desarrollo de dicha entidad. La realidad es otra, pues a pesar de las disímiles bondades que brinda el marco de trabajo, su empleo no constituye un recurso a tener en cuenta por la comunidad de desarrollo y los proyectos productivos de la UCI para la construcción de soluciones informáticas.

Existen numerosos factores que atentan negativamente al desarrollo de Ksike, deviniendo en varias situaciones problemáticas. Actualmente la tecnología permanece con un bajo nivel de utilización por parte de la comunidad de desarrollo y los proyectos productivos de la UCI, aún cuando es un producto soberano y con excelentes prestaciones. A raíz de esto, puede inferirse además que al no emplearse el marco de trabajo el mismo no se potencia ni se desarrolla. Esta situación le atañe directamente al equipo de desarrollo del *framework*, quienes se esfuerzan por insertar su propuesta entre las soluciones factibles al desarrollo de software a emplear en la UCI. Teniendo en cuenta las deficiencias antes mencionadas se define como **problema a resolver:** ¿Cómo potenciar el empleo del *framework* Ksike como la 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 garantice mejorar el desarrollo del software que se sustenta en el *framework* Ksike. Se tiene como **objeto de estudio:** El proceso de desarrollo de IDE[3] orientados a *framework* de desarrollo y como **campo de acción**: La gestión automatizada de los procesos que intervienen en el desarrollo de software basado en Ksike.

[2] **GEYSED** – Centro de Desarrollo de soluciones para sistemas Geo-informáticos y de Señales Digitales.
[3] **IDE** – Acrónimo de *Integrated Development Environment* (Entorno Integrado de Desarrollo)

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. Validar 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. Realizar la validación de los componentes implementados.

En este punto el autor propone como **idea a defender** que**:** La utilización de una herramienta que automatice los procesos de desarrollo de software sustentado en el *framework* Ksike permitirá que dicho marco de trabajo sea asimilado con más facilidad por la comunidad de desarrollo y los proyectos productivos de la UCI.

Para la realización de la investigación se utiliza una **estrategia descriptiva** ya que se cuenta con bibliografía abundante referente a las ventajas del desarrollo de software sustentado en los marcos de trabajos orientados al desarrollo de aplicaciones informáticas, así como de las facilidades que brindan estos a la agilización en la obtención 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**:

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 pretende realizar 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 Integrados de Desarrollo para agilizar la obtención de soluciones informáticas. Para ello se realizará la observación de soluciones existentes de ese tipo que pueden ser útiles y aportar 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, rapidez en la obtención de soluciones y 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 correspondiente a los siguientes elementos:

- ✓ Una aplicación capaz de disminuir significativamente el tiempo 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 amplia documentación determinada por la metodología RUP[4], que facilitará la extensión de las funcionalidades por parte de miembros externos al equipo original.

Finalmente, para dar cumplimiento a los objetivos específicos y a las tareas propuestas para la presente investigación, la **estructuración del contenido** del documento queda conformada de la siguiente forma:

Capítulo 1:"Fundamentos Teóricos de la Investigación". En este capítulo se tratarán los conceptos y definiciones asociados al problema identificado. Los mismos brindarán una mayor comprensión del objeto de estudio de la investigación. También se detallará más elaboradamente los elementos que constituyen la situación problemática imperante al uso de Ksike.

Capítulo 2:"Tendencias y tecnologías". En este capítulo se realizará una descripción exhaustiva de sistemas o aplicaciones para el desarrollo de software. Se harán los análisis correspondientes a las herramientas y tecnologías que se emplearán en el desarrollo de la solución propuesta.

Capítulo 3: "Análisis y Diseño del Sistema": En este capítulo se desarrolla el modelo de dominio, la especificación de los requisitos y la descripción de los casos de uso, el análisis y diseño de la aplicación. Se definen además los Modelos de Análisis y Diseño respectivamente, de acuerdo con las fases especificadas por la metodología RUP para el desarrollo de software. Además se describen los estándares y elementos principales que permitan confeccionar la arquitectura de la herramienta de desarrollo propuesta como solución.

Capítulo 4: "Implementación y Prueba". En este capítulo se describen los elementos necesarios para la implementación de la solución propuesta: diagramas del diseño y los estándares para el desarrollo de la Interfaz Gráfica de Usuario (GUI[5]). Se muestra la distribución del sistema en nodos mediante el diagrama de despliegue y 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 a desarrollar.

[4] **RUP** – Siglas de (*Rational Unified Process*) Metodología de desarrollo de software.

[5] **GUI** – siglas del inglés *Graphic User Interface.*

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS DE LA INVESTIGACIÓN

En el presente capítulo se abordarán 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 describirán detalladamente además, las principales características del *framework* Ksike. También se realizarán análisis más profundos sobre las causas que provocan la existencia de la situación problemática latente ante el uso de esta tecnología.

1.1. Conceptos asociados al dominio del problema

1.1.1. Aplicación informática

El término aplicación resulta familiar de cierta forma a todos los usuarios que tienen acceso a ordenadores, teléfonos celulares o cualquier dispositivo que posea software incluido, pero cabe hacer un análisis de su concepto, ya que la automatización de los procesos del *framework* Ksike está dirigida a garantizar mejoría en la concepción y desarrollo de aplicaciones informáticas. 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*" (3).

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 (4).

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.

Las aplicaciones difieren de los sistemas operativos, quienes se ejecutan para controlar los dispositivos de hardware de las computadoras y brindar un entorno propicio para la ejecución y utilización de dichos recursos que puedan ser empleados

[6] **RAE –** Real Academia Española.

por las aplicaciones. También son diferentes a las utilidades o *utilities*, software especializado en la realización de tareas de mantenimiento a los sistemas. El software de aplicación es aquel que hace que el computador coopere con el usuario en la realización de tareas típicamente humanas, tales como gestionar una contabilidad o escribir un texto.

Las aplicaciones informáticas están divididas en dos categorías: la primera, aplicaciones básicas, también conocidas como aplicaciones de propósitos generales que son empleadas en varias disciplinas. Entre ellas se incluyen: editores de texto, hojas de cálculo, sistemas de manejo de bases de datos y presentaciones. La otra categoría se refiere a las aplicaciones especializadas, también denominadas aplicaciones de propósitos específicos. Esta incluye miles de otros programas que están más estrechamente relacionados con actividades particulares como: editores de audio/video, programas para la creación de multimedia, páginas web y realidad virtual (5).

Figura 1: Tareas que pueden ejecutarse por aplicaciones informáticas y entornos de ejecución.

Las aplicaciones informáticas son de indudable utilidad en la sociedad actual, ya que durante varios años han ido evolucionando para lograr mayor vínculo con las personas, proveiendo mejores vías de comunicación hombre-máquina. Las mismas se ejecutan en una capa superior al software de sistema o sistemas operativos y, aunque utilizan recursos de los mismos, son totalmente independientemente de ellos. Hoy día residen en todo tipo de plataformas, ya sea: escritorio, web o móvil, como se muestra en la Figura 1 anteriormente. El último es más reciente que los anteriores, pero cabe destacar que, en cada uno de ellos, las prestaciones y capacidades actualmente son

muy similares por lo que pueden brindarse prácticamente los mismos servicios a los usuarios.

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 Interfaces Gráficas de Usuario (GUI), editor de código, un compilador y/o intérprete y un depurador (6). Margaret Rouse, agrega además que: Los IDE pueden ser aplicaciones independientes o pueden estar vinculadas como partes compatibles con algunas aplicaciones (7).

Citando un artículo publicado en ProgramacionDesarrollo.es, *blog* dedicado a la presentación, discusión y descripción de tecnologías de desarrollo en ámbito general, 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 de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto (…)*" (8).

De manera general puede resumirse que los IDE son aplicaciones de software 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 y la implementación de las aplicaciones. Se componen normalmente por:

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

Debe comprenderse además que los IDE 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. En el epígrafe 3.2 se abordarán al detalle algunas de las características principales de dichas herramientas de desarrollo.

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 Programación Orientada a Objetos (OOP), dígase la modularidad; según define Eugenia Bahit en su libro "OPP y MVC en PHP". 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 (…)*" (9).

Otros autores con respecto al término módulo definen que: La estructura de los sistemas informáticos "*(…) se constituye de componentes, módulos o piezas de código que nacen de la noción de abstracción, cumpliendo funciones específicas e interactuando entre sí con un comportamiento definido (…) los componentes se organizan de acuerdo a ciertos criterios, que representan decisiones del diseño (…)*" (10).

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 (…)*" (11).

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 como se ilustra en la Figura 2 a continuación.

Figura 2 - Esquema abstracto de composición modular de una aplicación informática.

Para obtener una mejor comprensión sobre el término módulo, es necesario analizar otros dos conceptos, componente y plug-in, que en las disciplinas de Ingeniería Informática comparten estrecha relación con el concepto de módulo anteriormente definido y 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 y los principios que orientan su diseño y evolución. Debe quedar claro que en esta definición el concepto de "componente" es genérico e informal (12).

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 (13). Que sea una unidad de composición y no 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 *plug-in* 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 *plug-in*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 *plug-in* que tiene la capacidad de procesar y mostrar el contenido de los archivos Adobe Flash, este es procesado por el *plug-in* en caso de que no existiera, el objeto simplemente no se mostrará al usuario final. La ausencia del *plug-in* apropiado no constituye un impedimento para mostrar el contenido de la página web (14).

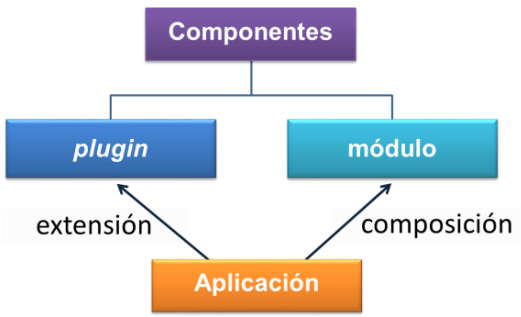

Figura 3 - Los *plugins* constituyen extensiones de las aplicaciones y los módulos partes componentes de las mismas. Ambos son componentes de software.

A modo de resumen se pueden identificar a los módulos o *modules*, así como a los *plug-in, plugin* o extensiones como tipos particulares de componentes de software, el primero como parte intrínseca de un programa informático y a los *plugins* como a extensiones de las funcionalidades de una aplicación que pueden o no existir sin afectar el funcionamiento de la misma, como anteriormente se ilustra en la Figura 3.

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 contenido. Sirven para evitar la creación constante de contenidos similares, permitiendo la replicación de estos (15).

La referencia online de Visual Studio en el apartado de Plantillas de Visual Studio MSDN: *Template Reference* define que: Las plantillas proporcionan elementos reutilizables y personalizables que aceleran el proceso de desarrollo ya que los usuarios no tienen que crear nuevos proyectos y elementos desde cero. "*(…) proporcionan a los usuarios un punto de partida para empezar a crear proyectos o ampliar proyectos actuales. Las plantillas de proyecto proporcionan los archivos necesarios para un tipo de proyecto determinado (…)*" (16).

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, según se ilustra en la Figura 4. Las plantillas 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.

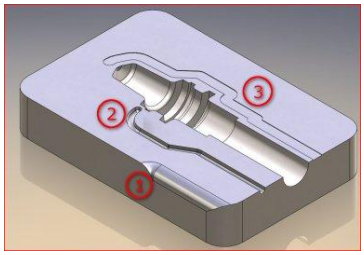

Figura 4 - Plantilla o molde para la realización de un componente mecánico.

A partir de la plantilla pueden así mismo diseñarse y fabricarse nuevas plantillas. 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 plantillas incluirán variables y posiblemente unos pocos operadores lógicos para permitir una mejor adaptabilidad de la misma.

1.1.5. Marco de trabajo

La palabra inglesa *framework*, puede traducirse al español como marco de trabajo, pero en términos generales se define según el diccionario MacMillan como (17): Un sistema de reglas, leyes, principios, acuerdos etc., que establecen la forma en que funciona un determinado elemento. Una estructura que brinda soportes de forma particular.

Otros autores definen que, un *framework*, en el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado (18). 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 y que, además, pueden ser fácilmente depurados y probados; el autor establece tres puntos de vistas específicos para el análisis de los *frameworks* como: contenedores, modelos arquitectónicos y guías metodológicas para el desarrollo de software (19). A continuación se explicará cada uno de ellos.

Contenedor

Vistos como un contenedor, los *frameworks* son una forma de re-empaquetar una función o un conjunto de funciones (relacionadas o no) para alcanzar uno o más de los siguientes objetivos:

- Simplicidad de uso.
- Consistencia en interfaz.
- Mejora a las funcionalidades básicas.
- Recogida de procesos discretos en una asociación lógica (un objeto).

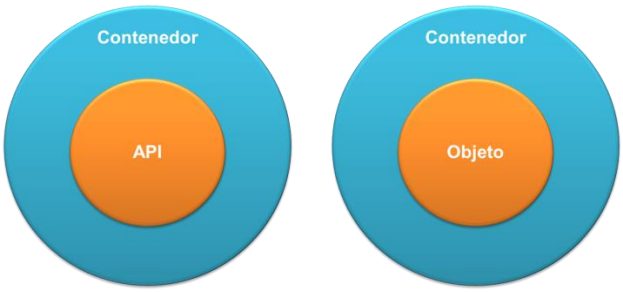

Figura 5 - Un contenedor que proporciona valor añadido.

Los *frameworks* llevan todo el trabajo pesado de las funciones primitivas o fundamentales abstrayendo el uso de los recursos básicos de los sistemas. Además proveen un conjunto nuevo de funcionalidades. Emplean API[7], objetos e incluso otros contenedores que realizan operaciones totalmente nuevas. En lugar de añadir, restar o gestionar recursos, ellos modifican comportamiento existente (19).

Arquitectura

Una arquitectura es un estilo o patrón que incorpora elementos específicos de diseño. Obviamente un marco de trabajo orientado al desarrollo de aplicaciones de software necesita tener un diseño. Su arquitectura está separada de la colección de partes que implementa y de la aplicación de una metodología de implementación. En esencia, una arquitectura implementa relaciones entre objetos, herencia, contenedores, intermediarios, colecciones, etc. Las arquitecturas tienen la característica de que pueden ser ignoradas o reemplazadas, al menos al principio de un proyecto. También definen una interfaz que permite la utilización de sus recursos, como se ilustra en la Figura 6. Apropiándose de estas características, los *frameworks* son muy útiles porque crean una estructura (colección de objetos) reutilizable que proporcionan funcionalidades mejoradas (19).

Figura 6 - Una arquitectura que asocia/contiene varios objetos
(La interfaz es el contenedor de la arquitectura)

[7] **API –** (*Application Programming Interface*) Interfaz de Programación de Aplicaciones.

Metodología

Las metodologías, como lo definen los propios términos – método – forma de realizar una actividad y – logía – análisis científico, diseñado, consistente y respetable; son guías para la realización de un conjunto de procesos. Según el diccionario se definen como: un cuerpo de prácticas, procedimientos y reglas empleados por quienes trabajan en una disciplina o materia (19).

Los *frameworks* generalmente implementan una metodología que, a diferencia de las arquitecturas que tratan con las relaciones entre objetos, establecen normas para la interacción de los procesos, Figura 7, siendo la primera una relación pasiva mientras que la segunda una actividad. Muchas de las metodologías garantizan la comunicación entre objetos, el manejo de la persistencia de datos y las respuestas a eventos de usuario (19).

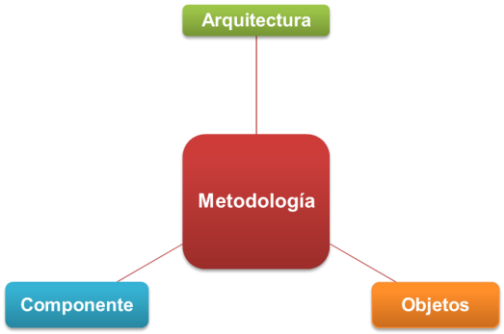

Figura 7 - Una metodología define la interacción entre arquitectura, componentes y objetos.

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 iniciar a programar 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 cómo la duración del mismo (20).

Continuando con las definiciones descritas dentro de las disciplinas de la Ingeniería de Software (ISW), en este caso en particular de las empleadas por el *Project Management Institute* (PMI), el PMBOK 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 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"* (21).

Ambas definiciones de manera muy general proponen que los proyectos pueden ser aplicables a todo tipo de disciplinas no solo las de Ingeniería Informática, sino a cualquiera donde se conceptualice. Comúnmente se entiende por proyecto la planificación que consiste en un conjunto de actividades que se encuentran interrelacionadas y coordinadas. La razón del mismo consiste en alcanzar objetivos específicos dentro de los límites que imponen un presupuesto con la calidad requerida en un lapso de tiempo previamente definido. Empleándose para su gestión la aplicación de conocimientos, habilidades, herramientas y técnicas en función de satisfacer los requisitos del mismo, apuntando a lograr un resultado único surgido como respuesta a una necesidad (2).

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. Donde intervienen ciertas particularidades tales como: la estructura jerárquica de sus entes, tipos de archivos, ficheros de configuración, etc (2).

La MSDN[8] brinda un concepto aterrizado al IDE *Visual Studio* que es de gran interés para este trabajo de investigación, ya que 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, todo con el objetivo de ayudar a organizar y realizar tareas comunes con los elementos que se están desarrollando (22). "*El resultado de un proyecto puede ser un programa ejecutable (exe), un archivo de biblioteca dinámica (dll) o un módulo, entre otros*" (22).

[8] **MSDN –** Acrónimo de *MicroSoft Developer Network* (Red de Desarrolladores de Microsoft).

1.2. Objeto de Estudio

1.2.1. Descripción general

En el escritorio o *desktop*, por ser un ambiente más naturalizado y además 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[9] y tecnologías AJAX[10] para establecer la comunicación entre cliente y servidor, se han creado un conjunto nuevo de filosofías y marcos de trabajo. Los mismos están dirigidos por ende al desarrollo de aplicaciones específicas para web. La forma de desarrollo ha variado o se ha perfeccionado consecuentemente, haciendo diferentes actualmente los principios e implementación de soluciones para ambos entornos, web y escritorio.

Ksike surge como necesidad de obtener una plataforma para la construcción de aplicaciones orientadas a entornos web, aprovechando las potencialidades del desarrollo orientado a ambientes escritorios y además, enfocado 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. Partiendo de la idea de que potencia el desarrollo de productos de software basados en el paradigma de OOP[11] y por ende todas las ventajas que este provee (2).

Su objetivo fundamental consiste en potenciar la construcción ágil de software, permitiéndole al desarrollador concentrarse en la lógica de la aplicación y abstraerse de la comunicación cliente-servidor. Asumiendo como proyección la ruptura de las barreas entre el desarrollo de aplicaciones orientadas a entornos web y de escritorio. Este proyecto se inició a mediados del mes de julio del 2010. Constituye el reflejo del interés profesional de sus integrantes en aras de lograr el objetivo propuesto. 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 (2).

[9] **HTTP** – *Hyper Text Transfer Protocol* (Protocolo de Transferencia de Hiper Texto).
[10] **AJAX** – Acrónimo de *Asyncronic JavaScript and XML*(JavaScript y XML Asincrónicos).
[11] **OOP**– *Object Oriented Programming* (Programación Orientada a Objetos, POO).

Figura 8 - Logo del *framework* Ksike.

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

- ✓ API en el lenguaje JavaScript para el desarrollo del lado cliente.
- ✓ API en el lenguaje PHP[12] para el desarrollo del lado servidor.
- ✓ Módulo para el mapeo de bases de datos con el ORM Doctrine.
- ✓ Módulo para la gestión de bases de datos de PostgreSQL.
- ✓ Patrones de Diseño: Singleton, Observer, Factory.

Como valor añadido además al paquete de esta versión, se encuentran 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.
- ✓ Driver para el manejo de información sobre ficheros en los formatos JSON[13], XML[14], PHP, JS[15], INI[16].
- ✓ Integra el motor de plantillas Twig

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

A pesar de las disímiles bondades abordadas anteriormente, el desarrollo de software sustentado en Ksike no constituye un hecho realista 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, el mismo no cuenta con los mejores niveles de divulgación, distribución y documentación, emplear el *framework* resulta engorroso porque el mismo posee varios procesos que complejizan el trabajo de los desarrolladores y que constituyen en

[12] **PHP**- (*Hypertext Pre-processor*) Pre-procesador de Hipertexto.
[13] **JSON**-(*JavaScript Object Notation*) Notación de Objetos de JavaScript.
[14] **XML**-(*Extensible Markup Language*) Lenguaje de Maquetado Extensible.
[15] **JS** – Siglas de JavaScript.
[16] **INI** –Extensión de ficheros de configuración de aplicaciones de Sistemas Operativos Windows.

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. Esquema de organización de Ksike

Ksike propone dentro de su marco, filosofías organizativas para proyecto, aplicación y módulo, que están dirigidas a describir la composición física de las soluciones informáticas que se desarrollan haciendo uso del propio marco de trabajo. El *framework* permite manejar bajo un proyecto, aplicación o módulo, 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 proyectos, aplicaciones y módulos de Ksike presenten una compleja estructura de directorios, ficheros de configuración, clases y otros recursos** sobre la cuál se hace una descripción a continuación.

El esquema de organización propuesto por Ksike para proyectos necesita de los directorios ***core***, ***lib***, ***plugins***; como se puede observar en la Figura 9. Además requiere un fichero ***index.html*** o ***index.php*** que es el Punto de Acceso Común (PAC), 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 proyecto sobre el marco de trabajo. El segundo 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 módulos del sistema en desarrollo (2).

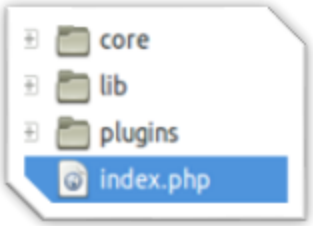

Figura 9 - Estructura de un proyecto Ksike.

Por otro lado, las aplicaciones y 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 10. 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.

Figura 10 - Estructura de una aplicación y un módulo de Ksike.

Adentrándose en la estructura de directorios del lado cliente, como se puede apreciar en la Figura 11, 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[17] 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.

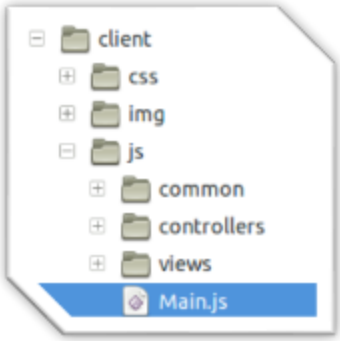

Figura 11 - Estructura interna del directorio *client*.

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 del lado cliente. En el directorio js se almacena además todas las clases JavaScript que se empleen para la programación de la lógica del lado cliente.

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

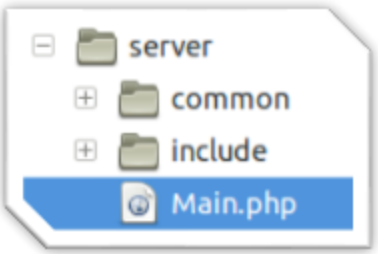

Figura 12 - Estructura interna del directorio *server*.

[17] **CSS –** Siglas de *Cascade Style Sheets* (Hojas de Estilos en Cascada).

(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, sin mencionar que es 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.

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. También es necesario conocer sobre los nombres, localizaciones y funciones específicas de los directorios y ficheros que el *framework* utiliza para su correcto funcionamiento, como por ejemplo los empleados para hacer carga dinámica/automática de contenidos; este es el caso de los directorios: ***include***, que se encuentran localizados dentro del lado servidor. Aún 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, 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 extensiones y herramientas básicas que gestionan la creación de proyectos, aplicaciones y módulos sobre Entornos Integrados de Desarrollo 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 y, subliminarmente provoca 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* en sí mismo, 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 agrupa 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***; que están asociados a la ruta física de la aplicación, los módulos del proyecto, bibliotecas y componentes externos y la URL[18] 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 13 - Configuración de enrutamiento por espacios de nombre.

La otra parte del enrutamiento 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***, ***pattern***, como se ilustra en la Figura 14.

```
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 14 - Claves configurables para el enrutamiento de peticiones al controlador frontal de Ksike.

En cambio los 'alias' son: ***web***, con varios elementos de configuración y ***publisher***, también con disímiles valores configurables. Estos a su vez, engloban otros muchos nomencladores de parámetros configurables como son: ***uri***, ***hey***, y muchos otros que pueden divisarse en la Figura 15 a continuación.

[18] **URL** – Siglas de *Uniform Resource Locator* (Localizador Uniforme de Recursos)

```
$config["router"]["app"]     = 'app/';
$config["router"]["plugins"]= 'plugins/';
$config["router"]["lib"]     = 'lib/';
$config["router"]["web"]     = '';
$config["router"]["request"]["key"]["proj"]          = false;
$config["router"]["request"]["key"]["controller"]    = 'Main';
$config["router"]["request"]["key"]["action"]        = 'index';
$config["router"]["request"]["key"]["params"]        = '';
$config["router"]["request"]["key"]["outFormat"]     = 'html';
$config["router"]["request"]["key"]["outInfo"]       = 'void';
$config["router"]["request"]["key"]["outOption"]     = 'none';
$config["router"]["request"]["url"]["pattern"]       = 'controller/action/params/outFormat/outInfo/outOption';

$config["router"]["request"]["alias"]["web"]["url"]["pattern"]       = 'proj/controller/action/params/outFormat/outIn
$config["router"]["request"]["alias"]["web"]["key"]["controller"]    = 'Main';
$config["router"]["request"]["alias"]["web"]["key"]["action"]        = 'index';
$config["router"]["request"]["alias"]["web"]["key"]["outFormat"]     = 'js';
$config["router"]["request"]["alias"]["web"]["key"]["outInfo"]       = 'void';
$config["router"]["request"]["alias"]["web"]["key"]["outOption"]     = 'none';
$config["router"]["request"]["alias"]["web"]["key"]["proj"]          = 'atipic';

$config["router"]["request"]["alias"]["publisher"]["url"]["pattern"]         = 'params:type/outFormat';
$config["router"]["request"]["alias"]["publisher"]["key"]["controller"]      = 'Main';
$config["router"]["request"]["alias"]["publisher"]["key"]["action"]          = 'contents';
$config["router"]["request"]["alias"]["publisher"]["key"]["outFormat"]       = 'js';
$config["router"]["request"]["alias"]["publisher"]["key"]["outInfo"]         = 'void';
$config["router"]["request"]["alias"]["publisher"]["key"]["outOption"]       = 'none';

//.....................................................
$config["loader"]["cfg"]["load"]                    = "auto";
$config["loader"]["filter"]["iface"]["Module"][]    = 'Security';
$config["loader"]["filter"]["iface"]["Module"][]    = 'Error';
$config["loader"]["filter"]["control"]["Module"][]  = 'Primal';
$config["loader"]["filter"]["control"]["Main"][]    = 'App';
$config["loader"]["filter"]["control"]["Plugin"][]  = 'Plugin';
$config["loader"]["filter"]["resource"]["Main"][]       = 'Linker';
```

Figura 15 - 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***), flitros (***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 e Inyección de Dependencia (***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, como se ilustró anteriormente en la Figuras 15 y a continuación en la Figura 16; 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.

```
var config = {
    "ns" : "std",
    "router": {
        "mode": "none",  //... secure|none|partial
        "p2c": "../../", //... relative (../)| url (index.php/publisher/)
        //"c2p": "tools/BHike/"
        "lib" : "lib/"
    },
    "loader": {
        "cfg": {
            "loadType": "syncrhonic"
        },
        "base": [
            'patterns/Kcl.Class',
            'patterns/Kcl.Singleton',
            'patterns/Kcl.Observed',
            'patterns/Kcl.Factory',
            'helpers/Kcl.Router',
            'helpers/Kcl.Loader',
            'handlers/Kcl.Engine'
        ],
        "filter": [
            'resource/format/Kcl.Format',
            'resource/format/Kcl.Format.JSON',
            'resource/bridge/Kcl.Bridge',
            'resource/bridge/Kcl.Bridge.Ajax',
            'control/Kcl.Primal',
            'control/Kcl.Plugin',
            'control/Kcl.App'
        ],
        "package": [
            'front/Kcl.FrontController',
            'error/Kcl.Error'
        ],
        "other": [
            'Main'
        ]
    }
}
```

Figura 16 - Fichero de configuración del cliente de Ksike (config.js).

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, el grupo de las bases (***base***); que 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 de la gran variedad ficheros de configuración el hecho de que Ksike proporciona diversidad de formatos para la manipulación de la información

asociada a las configuraciones (JSON[19], XML[20], PHP[21], JS[22], INI[23]) los que tienen sus peculiaridades según se ilustra en la Figura 17.

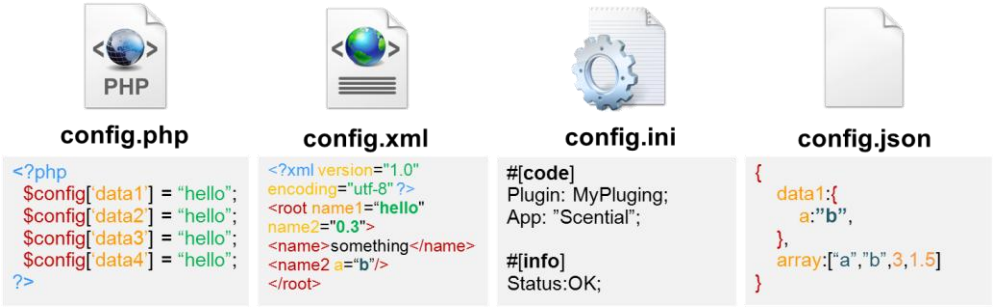

Figura 17 - Formatos de archivos de configuración soportados por Ksike.

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?

Una vez más, para saltarse 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 Interfaz de Línea de Comandos (CLI[24]). Lo cierto es que Ksike que es un producto con muchísimas ventajas y posibilidades, actualmente en desarrollo además y creado en la cantera de la UCI, resulta estar en desuso y no se prueba ni se le da el correcto mantenimiento.

1.2.2.3. Sistema de Clases de JavaScript

"La Programación Orientada a Objetos (OOP) 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" (23).

Muchas veces hay que utilizar varias instancias o ejemplares análogos de un determinado objeto, por ejemplo las ventanas de las aplicaciones de la PC, varios

[19] **JSON**-(*JavaScript Object Notation*) Notación de Objetos de JavaScript.
[20] **XML**-(*Extensible Markup Languaje*) Lenguaje de Maquetado Extensible.
[21] **PHP**- (*PHP Hypertext Pre-processor*) Pre-procesador de Hipertexto.
[22] **JS** – Siglas de JavaScript.
[23] **INI** –Extensión de ficheros de configuración de aplicaciones de Sistemas Operativos Windows.
[24] **CLI** – Siglas de *Command Line Interface* (Interface de Línea de Comandos).

usuarios, clientes, cuentas de banco, etc. *"La definición genérica de estos objetos análogos se realizar mediante la clase. Así, una clase contiene una completa y detallada descripción de la información y las funciones que contendrá cada objeto de esa clase"* (23).

La Programación Orientada a Objetos (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 (24). Entre los lenguajes que implementan el paradigma de la OPP 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[25]. 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" (24). En la Figura 18 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

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

KsikeJS

Figura 18 – 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, que aunque es muy similar a la brindada por los lenguajes clásicos, necesita ser conocida 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

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

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. De modo tal 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 19.

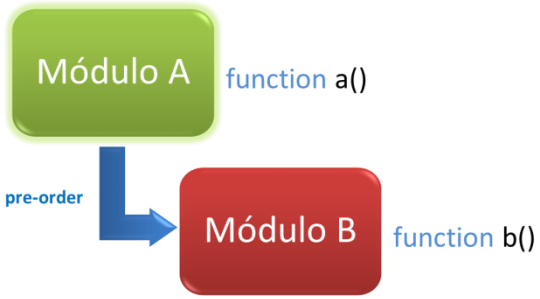

Figura 19 - Representación gráfica del funcionamiento del Recurso Linker de Ksike.

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 desees. Solo se puede emitir una señal desde la clase donde esta se encuentra implementada. 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 (25). 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. Cómo 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 20 - Implementación de los recursos "Signals & Slots" del *framework* Qt y "Linker" de Ksike.

Como puede evidenciarse en la Figura 20, 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. Es por lo que la implementación tal y como se representa en la Figura 21, 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.

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

Figura 21 - Implementación realizada con el Enlazador de Ksike.

La otra manera de usar los enlaces en Ksike es haciendo uso de ficheros de configuración, como se muestra en la Figura 22. 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 22 - 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. Una vez más el asunto de las configuraciones puede resultar complejo o la otra variante, implementar manualmente los enlaces

dentro del módulo de principal del sistema, pues el resultado sería el mismo, exceso de trabajo manual.

Si ya no es necesario el enlace, hay que cerciorarse de eliminar su configuración o declaración, pues sino 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 (26).

Con el fin de garantizar seguridad de los archivos de los proyectos desarrollados, varios *frameworks* para el desarrollo de aplicaciones como: Symfony y Symfony 2 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 (27) (28).

Symfony2, al estar arquitectónicamente orientado al desarrollo de componentes, proporciona bajo la envoltura de cada *Bundle*[26], un directorio Resources/public/ en el cual se almacenan los datos que son públicos para los navegadores. Aunque como se

[26] **Bundle –** Directorio a manera de módulo de aplicación, que contiene todo tipo de archivos dentro una estructura jerarquizada de directorios (28)**.**

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 los ellos se relacionan en el directorio *public* (28). 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 a continuación y sobre la cuál se dará explicación seguidamente.

Publicación	Archivos públicos	Seguridad	Enrutamiento	Rendimiento
Total	**Ksike + Proyecto**	**Mínima**	**Directo**	**Alto**
Parcial	**PAC[27] + Parte cliente**	**Parcial**	**Parcial**	**Medio**
Mínima	**PAC**	**Total**	**Indirecto**	**Bajo**

Tabla 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 estará disponible 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, provea un **elevado rendimiento al sistema**.

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 los PAC del proyecto quien actúa como intermediario entre las peticiones generadas en el proyecto y el controlador frontal de Ksike. De esta

[27] **PAC –** Punto de Acceso Común, en inglés (CAP, *Common Access Point*).

manera se protege el acceso al resto de la información del proyecto, garantizando un nivel de **seguridad parcial**. El **enrutamiento** ocurre de forma **parcial** también 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 **mínima publicación**, 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, como se ilustra en la Figura 23. El resultado será almacenando en un archivo de caché para agilizar eventualmente la respuesta ante peticiones similares sobre un mismo recurso.

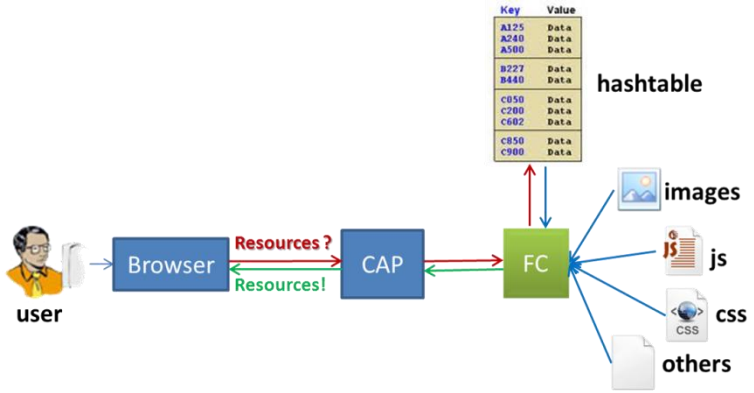

Figura 23 - Proceso de acceso a los recursos bajo el modo de mínima publicación.

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 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, como se representa a continuación en la Figura 24. 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.

```
1  #MiProyectoKsike
2  <VirtualHost *:80>
3      DocumentRoot    "/Proyectos/MiProyectoKsike/web"
4      DirectoryIndex app.php
5      ServerName     miproyecto.ksike.local
6      <Directory "/Proyectos/MiProyectoKsike/web">
7          AllowOverride All
8          Allow from All
9      </Directory>
10 </VirtualHost>
```

Figura 24 - Fichero de configuración de un virtual host en Apache Server.

Debido a que Ksike no posee un CLI con comandos que automaticen algunos de sus procesos al igual que Symfony y Symfony 2, es una realidad que su empleo para en el desarrollo de software no haga competencia a estos a los ojos 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, aplicados al desarrollo de software, 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 adecuada en la comunidad de desarrollo y la documentación 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. Lo que impide, en primer lugar que existan especialistas sobre el empleo de Ksike y en segundo lugar a 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 (29), 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 subutilizan el *framework* (2).

Se hace rechazo a la asimilación de esta nueva tecnología ya que se tienen que estudiar muchos elementos para garantizar su utilización por desarrolladores avanzados. Lo que deviene a que el desarrollo de aplicaciones sobre Ksike sea más lento o costoso que su contraparte haciendo uso otras tecnologías que, sin ser más completas, ya 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.

Actualmente la metodología de desarrollo que sigue equipo de Ksike, se basa en técnicas de captura de requisitos tales como "estudio de sistemas homólogos", así como elementos identificados basados en la experiencia en el desarrollo tanto de productos web como desktop. Esto permite que el *framework* se actualice constantemente, pues es posible integrar otros marcos de trabajo existentes, con mayor nivel de madurez y, delegarle a los mismo, responsabilidades específicas como manejo de la seguridad, como es el caso de Ulogin. Así se orienta el desarrollo del *framework* en garantizar la integración con otros marcos de trabajo.

1.2.3. ¿Por qué desarrollar una herramienta nueva?

Ksike es 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 que garantiza soberanía e independencia tecnológica, pues el marco de trabajo está programado desde cero. Siendo un producto soberano e 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 GPL y es gratuito, todas las herramientas desarrolladas sobre dicha tecnología son gratuitas por lo que 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 sirve la tecnología para los proyectos en desarrollo, no tiene ningún significado en costos pues adquirirla está libre de impuestos y no cuesta absolutamente nada.

Ksike es una tecnología multi-plataforma, por lo que es posible desarrollar aplicaciones que funcionen tanto en Mac, Linux como en Windows, aprovechando los componentes ya programados. Además que tiene como meta el desarrollar aplicaciones nativas para entornos móviles, aunque ya es posible el desarrollo de aplicaciones web con interfaces para este tipo de dispositivos. Al ser desarrollado y empleado en la UCI es posible crear una comunidad online que ante cualquier problema o duda siempre este

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 más cómoda y fluidamente a los desarrolladores logrando 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 (30). Un Entorno de Desarrollo Integrado que está dedicado específicamente al desarrollo sobre la tecnología Qt, aunque actualmente está muy extendido e incluye metas mucho más ambiciosas.

1.3. Conclusiones parciales

La conceptualización de un problema constituye el primer paso y a la vez el más importante para encontrarle una solución al mismo. Teniendo como basamento teórico las experiencias acumuladas por el proyecto Qt, queda como meta seguir sus pasos y desarrollar una herramienta que automatice los procesos del *framework* Ksike en aras de garantizar que su asimilación por parte de las comunidades de desarrollo y los proyectos productivos de la UCI. Es de presumir que, planteándole una solución total o parcial al conjunto de situaciones problemáticas que atentan al buen empleo de Ksike, este constituya un marco de trabajo a tener en cuenta para los proyectos futuros. Orientando el presente trabajo en aras de cumplir el verdadero propósito con el cuál fue concebido: ser una base tecnológica para el desarrollo de soluciones informáticas.

CAPÍTULO 2: TENDENCIAS Y TECNOLOGÍAS

En este capítulo se realizará el análisis de las principales tendencias o soluciones desarrolladas, haciendo una descripción exhaustiva de las características principales de los Entornos Integrados de Desarrollo actualmente más populares y con mayores prestaciones. Además se estarán analizando algunos análisis preliminares a modo de identificar las principales herramientas y recursos a utilizar durante todo el proceso de desarrollo de la solución planteada.

2.1. Análisis de soluciones existentes

De acuerdo con mencionado en el Capítulo 1 acerca de herramienta para 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*, *desktop* o *mobile*. 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 *desktop* o de escritorio, aunque existen tendencias a desarrollar herramientas web debido a las grandes ventajas que posee este tipo de entorno. Estas herramientas son denominadas Web-IDE de acuerdo al medio donde se ejecutan: la web.

El presente epígrafe está enfocado a mostrar los resultados de una investigación realizada a los IDE actualmente mejor elaborados, en aras de localizar un conjunto de potencialidades, patrones, tecnologías y facilidades de los mismos que puedan ser empleados en el desarrollo de software sobre la plataforma Ksike. A continuación se exponen algunos de los más conocidos, teniendo en cuenta las ventajas que presentan y su nivel de aceptación por las comunidades de desarrollo en las que son empleados. Se ha decidido agrupar bajo dos denominaciones (IDE y WIDE) al conjunto de aplicaciones analizadas para una mejor comprensión y organización de contenidos.

2.1.1. Entornos Integrados de Desarrollo (IDE)

NetBeans IDE: Entorno integrado de desarrollo libre, gratuito y sin restricciones de uso. 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 móv*iles*). Entre sus 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 multiplataforma y su última versión estable 7.1 es del 20 de noviembre del 2011. Es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento y con cerca de 100 socios en todo el mundo. Permite la creación de extensiones personalizadas (31).

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. Esta plataforma típicamente ha sido usada para desarrollar IDE como *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse. Sin embargo, también se puede usar para otros tipos de aplicaciones cliente como BitTorrent o Azureus. Adicionalmente Eclipse puede 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. Se provee soporte para Java y control de versiones (CVS) en el SDK[34] de Eclipse (32).

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). Es un software privativo aunque a partir de la versión de 2005 Microsoft ofrece

[28] **J2SE –** *Java 2 Platform Standard Edition*.
[29] **EJB -** *Enterprise JavaBeans,* APIs que forman parte del estándar de desarrollo de aplicaciones.
[30] **SOA** - Acrónimo de *Service Oriented Application* (Aplicaciones Orientadas a Servicios).
[31] **XML** - *Extensible Markup Language*.
[32] **WSDL** - *Web Service Description Language* (Lenguaje de Descripción de Servicios Web).
[33] **BPEL -** *Business Process Execution Language* (Lenguaje de Ejecución de Procesos de Negocio).
[34] **SDK –** Por sus siglas en inglés *Software Development Kit* (Paquete de Desarrollo de Software).

gratuitamente las *Express Editions*, que son varias ediciones básicas separadas por lenguajes de programación o plataforma enfocadas para novatos y entusiastas. Estas ediciones son iguales al entorno de desarrollo comercial pero sin características avanzadas (16).

Qt Creator: Es un IDE multiplataforma para C++ que forma parte de Qt SDK. Incluye un *debugger* visual y un editor de Interfaces Gráficas de Usuario y de formularios. Contiene varias facilidades entre las que incluyen el sobresaltado de sintaxis y el auto completamiento. Qt Creator usa el compilador de C++ de la colección de compiladores de GNU en 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) y para dispositivos móviles (Symbian, Maemo y Meego). Las opciones de ensamblado permiten cambiar los objetivos del mismo. Cuando se desarrolla una aplicación para un dispositivo móvil conectado a una computadora, Qt Creator genera un paquete de instalación, lo instala en el dispositivo y lo ejecuta. Los paquetes de instalación pueden ser publicados en *Ovi Store*. Para los dispositivos con Symbian los paquetes deben ser firmados (33).

Aptana Studio: IDE gratuito basado en Eclipse y desarrollado por Aptana, Inc., que puede funcionar bajo Windows, Mac y Linux. Provee soporte para lenguajes como: PHP, Python, Ruby, CSS, Ajax, HTML y Adobe AIR. Tiene la posibilidad de incluir complementos para nuevos lenguajes y funcionalidades. Presenta un asistente para HTML y JavaScript, librerías de Ajax, conexión vía ftp, sftp, ftps, y Aptana Cloud. Contiene además herramientas para el trabajo con bases de datos y es compatible con extensiones para Eclipse (34).

Zend Studio o Zend Development Environment: Es un completo entorno integrado de desarrollo para el lenguaje de programación PHP. Está escrito en Java, y está disponible para las plataformas Microsoft Windows, Mac OS X y GNU/Linux. Junto con su contraparte Zend Server (antes llamado Zend Platform), son la propuesta de *Zend* Technologies para el desarrollo de aplicaciones Web utilizando PHP, actuando Zend Studio como la parte cliente y Zend Server como la parte servidora. Se trata en ambos casos de software comercial, lo cual contrasta con el hecho de que PHP es software libre. A partir de la versión 6, Zend Studio fue hecho tomando como base el entorno de desarrollo Eclipse. No requiere instalación previa del entorno de ejecución de Java y soporta PHP4 y PHP5 (35).

2.1.2. Entornos Integrados de Desarrollo Web (WIDE)

WebDevStudio: Es un entorno de desarrollo basado en tecnologías AJAX/PHP accesible a través de un navegador web. Como cualquier otro entorno permite mantener proyectos informáticos implementados en diferentes lenguajes de programación, en este caso en C/C++ y Java, así como realizar una serie de operaciones básicas sobre ellos. WebDevStudio permite a los usuarios conectados tener un control absoluto sobre los proyectos que previamente hayan creado. Entre dicho control se destacan los procesos de compilación e implementación del proyecto puesto que la herramienta permite abstraerse totalmente tanto de dichos procesos como del lenguaje de programación (36).

Cloud9 IDE: Es un proyecto de software libre iniciado por Ajax.org, desarrollado sobre tecnología NodeJS y consecuentemente escrito en JavaScript. Propone brindar las mejores características de los IDE y editores de código fuente como: NetBeans, Eclipse, Textmate, definidos en *plugins.* Su centro de atención se basa en potenciar el desarrollo de JavaScript y 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 y soporte para JavaScript, HTML, CSS y modos mezclados de código. Integra un depurador para aplicaciones sobre NodeJS para el navegador Google Chrome y además es altamente extensible de acuerdo al sistema modular que define (37).

eXo IDE Cloud: Aplicación web que provee un entorno rico para el desarrollo de diversos contenidos, códigos fuente y servicios. No requiere instalación adicional y se ejecuta sobre los navegadores, posibilitando el acceso y manejo de ficheros desde cualquier lugar (38). 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 y editor WYSIWYG para HTML y Google Gadget.
- ✓ Soporte para varios lenguajes descriptivos y 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[35].
- ✓ Sistema de Control de Versiones GIT con soporte incluido para la mayoría de las operaciones con servidores locales y remotos.

J2EE[36] 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 (38).

eXo IDE Cloud brinda soporte para los navegadores web:

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

IDE Caxtor: Entorno de desarrollo conceptualizado en la Universidad de las Ciencias Informáticas (UCI) dirigido al desarrollo de aplicaciones web. Se compone de dos productos, una plataforma y un IDE. En la plataforma están definidos la mayoría de los componentes, acciones y patrones que usualmente se utilizan en la capa de presentación de una aplicación Web. El IDE Caxtor está concebido sobre la propia plataforma y cuenta con un diseñador de interfaces gráficas que permite la creación de GUI utilizando las ventajas de la librería ExtJS pero permitiendo una abstracción al código necesario para generarlas. Provee además un editor de código JavaScript con completamiento de código, chequeo de sintaxis, un editor de eventos y propiedades (39).

Lycan GENESIS - Component Builder: Es un IDE para el desarrollo de aplicaciones enriquecidas para la web basado en el IDE Caxtor. Facilita el desarrollo de componentes de una manera intuitiva, rápida y cómoda. Se reutiliza y configura para desarrollos más especializados en otros dominios (40).
Las principales características se describen a continuación:

[35] **REST** – Acrónimo de *Representational State Transfer.*

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

- ✓ Componentes que asistan el trabajo de diseño (Espacio de Diseño, Paleta de Herramientas, Editor de Propiedades, Navegador del Diseño, Editor de Código con resaltado de sintaxis).
- ✓ Diseño vía Sujetar-Arrastrar-Soltar.
- ✓ Diseño vía Seleccionar-Mover / Dimensionar uno o más componentes.
- ✓ Asistentes para facilitar la alineación de componentes (asistente de acercamiento superior, inferior, derecho e izquierdo; asistentes de alineación vertical derecho, centro e izquierdo; asistentes de alineación horizontal superior, centro, inferior).
- ✓ Facilidad Deshacer / Rehacer.
- ✓ Exportación / Importación de componentes.
- ✓ Facilidad de especificación e integración de nuevos componentes para el diseñador.

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.

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, ya que generan documentación suficiente para una mejor comprensión, en este caso se encuentra el *Rational Unified Process* (RUP).

Por otra parte están las metodologías ágiles, las cuales son más propensas al cambio, a la falta de información, generan poca documentación y se conciben para proyectos más sencillos, ajustándose a equipos de desarrollo pequeños con poco tiempo de entrega del producto. Un ejemplo clásico de estas es *Extreme Programming* (XP), conocido en español como Programación Extrema (1).

De acuerdo a lo antes descrito y 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 y 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.

RUP

Proceso Unificado de Desarrollo (*Rational Unified Process*, RUP) es una de las metodologías pesadas más conocidas y utilizadas. Esta es bastante robusta y precisa, permite controlar y documentar muy bien el desarrollo del software, eliminando los riesgos que puedan existir en el proceso, propiciando en todo momento un enfoque de trabajo bien estructurado, así como la asignación correcta y óptima de tareas y responsabilidades.

Para guiar el proceso de desarrollo del software emplea nueve flujos de trabajo organizados en cuatro fases de desarrollo como muestra la Figura 25. Las cuatro fases de desarrollo que definen el ciclo de vida son: (41)

- 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.

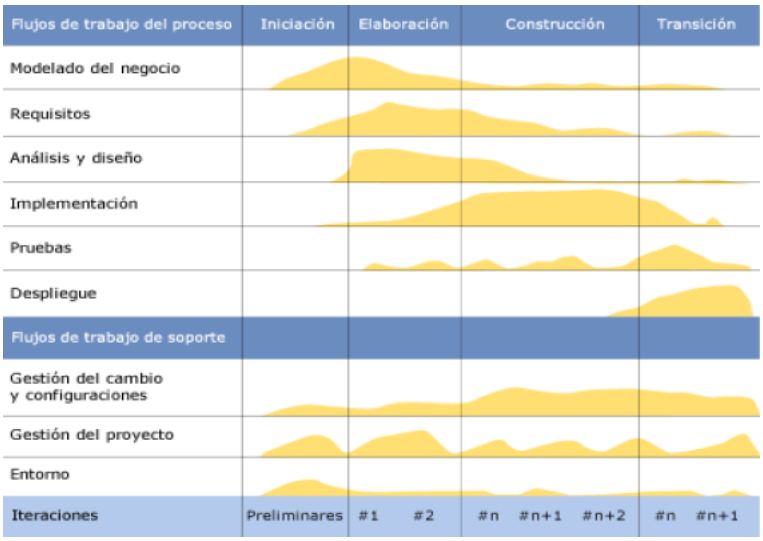

Figura 25 - Diagrama del esfuerzo de actividades según la etapa del proyecto.

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 esperadas. 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, ya 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 una 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 y se vuelven 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 que no se cometan fallos en el desarrollo de la aplicación y que se adquiera calidad y experiencia (42).

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, ya que permite visualizar, especificar, desarrollar y documentar los artefactos de la aplicación de forma eficiente y entendible (43). Básicamente UML es una herramienta gráfica para permitir 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 (44).

2.3. Framework de desarrollo

Con el objetivo de desarrollar una herramienta que responda, en calidad de IDE, al total de las necesidades de los desarrolladores de software que emplean Ksike y que además, sea desarrollada sobre tecnología libre, sea de fácil acceso e instalación, multiplataforma y extensible al mayor número de usuarios posibles. Además constituya un ejemplo demostrativo de las capacidades del propio Ksike, se define el desarrollo de una solución web, ya que 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.

Ksike *Framework*:

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.

Ext JS:

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 (45). 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.

2.4. 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 tanto en Alto o Bajo nivel, como en Estructurado u Orientado a Objetos, o con fines Web o Desktop. Algunos de estos lenguajes son PHP, C, Java y ASP.NET (46).

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.

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, además de ser de software libre. PHP es uno de los lenguajes de programación que nos 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 y un rendimiento del nivel de la plataforma .NET, ya que es la versión libre del sistema equivalente de Microsoft ASP para la creación de aplicaciones complejas (47).

En resumen, con PHP se pueden hacer grandes cosas con pocas líneas de código. La práctica ha demostrado que PHP5 es mejor que Java para la creación de aplicaciones web dinámicas específicamente, ya que el desarrollo en PHP es mucho más rápido, y el rendimiento y la escalabilidad es más fácil de lograr que en Java. Además, Java depende de su lenta máquina virtual y la separación entre presentación y controlador o lógica de negocio que en JSP es bastante mala. Tanto Java como PHP5 son ampliamente aceptados, sin embargo muchas organizaciones utilizan PHP para sus aplicaciones web porque en Java es mucho más difícil hacer cosas simples (48).

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 (24).

JavaScript es un poderoso 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:

- ✓ Crear poderosas aplicaciones web (ejecutables sobre navegadores web).
- ✓ Codificar aplicaciones del lado servidor, tales como ASP, o por ejemplo, código ejecutable usando Rhino[37].
- ✓ Programar *scripts* para automatizar tareas en los escritorios de Windows, a través del *Windows Scripting Host*.
- ✓ Crear extensiones/*plugin* para aplicaciones de escritorio como son: Firefox, Chrome, Dreamweaver.
- ✓ Crear aplicaciones sobre Adobe Air, las cuales son ejecutables en ambientes de escritorio (24).

2.5. Servidor Web

Un servidor web es un programa que, empleando el modelo cliente/servidor y el protocolo de transferencia de hipertexto (HTTP), brinda una salida al contenido desde las páginas web a sus usuarios, los que deben poseer equipos de cómputo con aplicaciones clientes de HTTP 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 *Microsoft's Internet Information Server* (IIS) y el más usado Apache, del cual a continuación se brindarán algunas características (4).

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

[37] **Rhino** – Motor de JavaScript escrito en Java.

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. Es muy sencillo ampliar las capacidades del servidor web Apache, cualquiera que posea una experiencia decente en la programación de C o Perl puede escribir un módulo para realizar una función determinada. Tiene una alta configurabilidad en la creación y gestión de *logs*. Apache permite la creación de ficheros de log a medida del administrador, de este modo puedes tener un mayor control sobre lo que sucede en el servidor (49).

2.6. Entorno de Desarrollo Integrado (IDE)

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.

Netbeans v7.0.1

Se define a Netbeans como IDE debido a que responde a software libre, soporta lenguajes de alto nivel orientado a objetos como Java y PHP, así como *frameworks* de desarrollo tales como ExtJS y Symfony. Es bastante robusto, hace el trabajo más cómodo y fácil, siendo agradable para el usuario y sencillo de usar. Netbeans es un producto gratuito y no tiene restricciones de uso, ya que es de código abierto, tiene una comunidad que le ofrece soporte a nivel mundial (31).

Este IDE permite que las aplicaciones sean desarrolladas a partir de módulos, lo cual posibilita que se pueda extender la aplicación, agregándole módulos nuevos. Netbeans incluye herramientas de esquemas XML, orientación a servicios web y modelado UML, así como soporte para PHP5. También permite a los desarrolladores crear web dinámica y posee bastantes fuentes de documentación. Unido a esto, soporta formato JSON y HTML5, genera documentación PHP y puede ser integrado con varios sistemas (31).

2.7. Herramienta CASE

CASE, *Computer Aided Software Engineering*, es un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el 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 y 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 (44).

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. Ésta última clasificación es la más recomendada, ya que asegura guiar y entender el proceso en todo momento, en este caso entran Visual Paradigm y Rational Rose (44).

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 con esto, tales como buena documentación y soporte online. Además es una herramienta fácil de utilizar, con una interfaz agradable, presenta soporte para la notación y 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 muy eficiente (1).

También cuenta para la parte de base de datos, con un generador de mapeo de objetos-relacionales para los lenguajes de programación Java, .NET y PHP. Unido a ella, permite su integración con otras herramientas como son los IDE Netbeans y Eclipse. Esta herramienta es reconocida porque realiza de forma clara y organizada la diagramación visual y el diseño de proyectos, permitiendo así integrar y desplegar las aplicaciones (44). Visual Paradigm es apoyado por un conjunto de idiomas tanto en la generación del código, como en la ingeniería inversa, ya que proporciona compatibilidad con 10 lenguajes, entre ellos C#, Delphi, C++, Java y PHP (50). Agregando a todo lo anterior que es una herramienta con alta interoperabilidad y que tiene también conexión con Rational Rose en sus archivos de proyecto, así como importación y exportación a formato XML (51).

Resumiendo, se selecciona esta herramienta para el diseño y modelado de la solución del problema de la investigación, ya que presenta grandes ventajas por encima de otras similares, esto se demuestra con sus características, como son la navegación

entre código y modelo, su sincronización de código fuente, su poderoso generador de documentación y reportes UML PDF/HTML/MS Word, su alto entorno de modelado visual, su soporte completo de notaciones UML y aplicaciones Web, sus diagramas de diseño automáticos sofisticados y su fácil instalación y actualización.

2.8. Conclusiones parciales

Luego de haber realizado la investigación sobre el estado de posibles soluciones al problema presentado, fueron encontradas innumerables herramientas que no automatizan el trabajo con Ksike porque el mismo es muy reciente y además no cuenta con la debida difusión en las comunidades de desarrollo. También se ha concluido que aunque dichas herramientas, en muchos casos permiten la personalización de extensiones que sí pudieran estar orientadas a agilizar los procesos de Ksike, estas no garantizan integración alguna con el mismo, por lo que no es posible desarrollar extensiones sobre esta tecnología.

A modo de brindar una solución al problema descrito y también al empleo de Ksike como base tecnológica de la misma, se llegó a la conclusión de que es necesario desarrollar una aplicación en su ciclo completo sobre el *framework*, de forma tal que se garantice el uso de la mayor cantidad de los recursos que brinda el mismo y la mayor independencia posible sobre el uso de otras tecnologías. La misma debe estar concebida sobre el uso de tecnologías libres, debe garantizar a los usuarios facilidades de acceso e instalación, debe ser multiplataforma y extensible al mayor número de usuarios posibles. Además se deben absolver el conjunto de buenas prácticas que rigen el funcionamiento de los IDE existentes, de forma tal que la herramienta que se desarrolle sea lo más fácil, intuitiva y funcionalmente posible a la mano de los usuarios.

CAPÍTULO 3: ANÁLISIS Y DISEÑO

En este capítulo se desarrolla el análisis y diseño de la aplicación definiendo además los Modelos de Análisis y Diseño respectivamente, de acuerdo con las fases de Análisis y Diseño especificadas por la metodología RUP para el desarrollo de software. Además se definen los estándares y elementos principales que permitan confeccionar la arquitectura de la herramienta de desarrollo propuesta como solución. Además se estarán analizando las principales características y componentes de los Entornos Integrados de Desarrollo.

3.1. Nivel 2

Contenido.

3.2. Descripción general de los IDE

Los Entornos Integrados de Desarrollo (IDE), definidos con anterioridad, proveen un marco de trabajo positivo para la concepción y desarrollo de software de alta calidad con elevada eficiencia. Están diseñados para maximizar la productividad de los programadores mediante el uso de una interfaz amigable e intuitiva que permite el empleo de funcionalidades, *frameworks* y otros recursos, que 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 tanto que poseen actualmente mucho más que herramientas convencionales de desarrollo, sino que posibilitan un marco más amplio de recursos para darle seguimiento a funciones básicas de los sistemas operativos. En este acápite se pretende abordar algunos de los elementos fundamentales de los IDE.

3.2.1. 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 explicará las fundamentales. Para ello se empleará el esquema definido por el autor Carlos Blanco en su artículo “Entornos de Desarrollo

Integrado (IDE´s) – Introducción" quien propone como partes fundamentales: editor de código fuente, compilador y/o intérprete, depurador, automatización de procesos.

3.2.1.1. 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 en un entorno de desarrollo integrado (IDE)"* (52). Los editores de código fuente están diseñados con características específicas que permitan simplificar y acelerar la entrada del código fuente (*source code*), 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 poseen también brindan accesos fáciles, desde ellos mismos como componentes particulares del IDE, que permiten ejecutar a un compilador, intérprete, depurador u otra herramienta que sea útil en el proceso de obtención de software (52).

3.2.1.2. 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[38]. A este proceso, ilustrado en la Figura 8, se le denomina **compilación** (53). 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 (54).

Figura 26 - Proceso de compilación de un programa de computo.
(Tomado del artículo de Carlos Blanco)

Los compiladores generalmente realizan todas o casi todas las siguientes operaciones: análisis léxico, pre-procesamiento, el análisis semántico (dirigida por la sintaxis de traducción), la generación de código y la optimización del código (52).

[38] **CPU** – Acrónimo del inglés *Central Process Unity* (Unidad Central de Procesos).

Un intérprete es un programa informático capaz de analizar y ejecutar otros programas (55). 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 y los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción y normalmente no guardan el resultado de dicha traducción (52).

"Usando un intérprete, un solo archivo fuente se puede ejecutar incluso en sistemas sumamente diferentes (plataformas Linux, Mac o Windows; incluso con distinta arquitectura hardware). Usando un compilador, un solo archivo fuente es específico a cada sistema, en este caso sobre la arquitectura desde la cual se ha compilado.

Los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa mientras se ejecuta, pero a cambio son más flexibles como entornos de programación y depuración (lo que se traduce, por ejemplo, en una mayor facilidad para reemplazar partes enteras del programa o añadir módulos completamente nuevos), y permiten ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete, sino del propio intérprete (lo que se conoce comúnmente como máquina virtual)" (52).

3.2.1.3. Depurador

Un depurador (*debugger*), es un programa usado para probar y depurar (eliminar los errores) de otros programas (el programa "objetivo"). El código al ser examinado puede alternativamente estar corriendo en un Simulador de Conjunto de Instrucciones (ISS[39]), 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 o completa; para limitar este impacto.

Si es un depurador de nivel de fuente o depurador simbólico, estos normalmente, forman parte de entornos de desarrollo integrados (IDE), cuando el programa "peta" o alcanza una condición predefinida, en la depuración típicamente se muestra la posición en el código fuente donde se halla el error (nº de línea de instrucción) (52). En la Figura 9 se ejemplifica un depurador a nivel de fuente. En este caso, nos informa de dos errores en el código: 1- La primera instrucción no está cerrada con su ";", 2- La variable página no está declarada en el código.

[39] **ISS –** Siglas de *Instructions Simulation System* (Sistema de Simulación de Instrucciones).

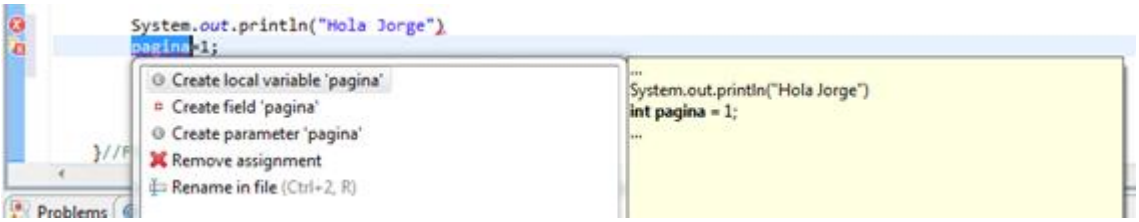

Figura 27 - Depurador a "nivel de fuente".

Si es un depurador de bajo nivel o un depurador de lenguaje de máquina, muestra la línea en el código fuente desensamblado (a menos que también tenga acceso en línea al código fuente original y pueda exhibir la sección apropiada del código del ensamblador o del compilador). En este caso "peta" cuando el programa no puede continuar normalmente debido a un error de programación. Por ejemplo, el programa pudo haber intentado usar una instrucción no disponible en la versión actual de la CPU o haber intentado tener acceso a memoria protegida o no disponible.

Típicamente, los depuradores también ofrecen funciones más sofisticadas tales como ejecutar un programa paso a paso, parar el programa (*breacking*), es decir, pausar el 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. También es posible continuar la ejecución en una posición diferente en el programa pasando a alguna instrucción que "peta" o a un error lógico.

"*La importancia de un buen depurador no puede ser exagerada. De hecho, la existencia y la calidad de tal herramienta para un lenguaje y una plataforma dada a menudo puede ser el factor de decisión en su uso, incluso si otro lenguaje/plataforma es más adecuado para la tarea*" (52). La ausencia de un depurador, una vez estamos acostumbrados a usar uno, se ha dicho que dificulta a sobremanera la búsqueda de los errores en el código fuente. La mayoría de los motores de depuración actuales, tal como GDB[40] proporcionan interfaces basadas en líneas de comandos (52).

3.2.1.4. 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 (52):

[40] **GDB** o GNU *Debugger* es el depurador estándar para el sistema operativo GNU.

- 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, ya 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 muchos de los Entornos Integrados de Desarrollo como: NetBeans, Eclipse, Visual Studio, etc.

3.2.2. Principales funcionalidades de los IDE

3.2.2.1. Edición (creación y modificación) del código fuente

Como se ha analizado en el punto “Editor de código fuente” de este trabajo, estos tienen características específicamente diseñadas para simplificar y acelerar la entrada de código fuente, como el resaltado de sintaxis, autocompletado y soporte de la funcionalidad de juego. Todos estos atajos son configurables; pueden ser adaptados a nuestras preferencias personales (52).

Garantizar esas configuraciones es una de las principales características que debe poseer los Entornos Integrados de Desarrollo, además de brindarle la oportunidad a los usuarios de incluir sus propios atajos, permitiendo hacer de los IDE, unas herramientas más configurables y adaptables a las preferencias de los usuarios.

3.2.2.2. Asistente de código

Una de las funciones que más beneficia a los desarrolladores es el auto completamiento del código fuente. A medida que se va escribiendo el código, el IDE ofrece un listado con sugerencias, ayudando a escribir más rápido. “*En esas sugerencias pueden salir constantes o funciones nativas de los lenguajes de programación, o cualquier otra parte del código que ya se haya desarrollado en el proyecto en uso*” (52).

Cuando se emplea esta característica sobre proyectos de programación orientada a objetos (POO), ayuda a encontrar propiedades o funciones de clases desarrolladas. Sin dudas es una característica de vital importancia que deben poseer los IDE, en aras

de agilizar el proceso de desarrollo de software. A continuación se ejemplifica en la Figura 10.

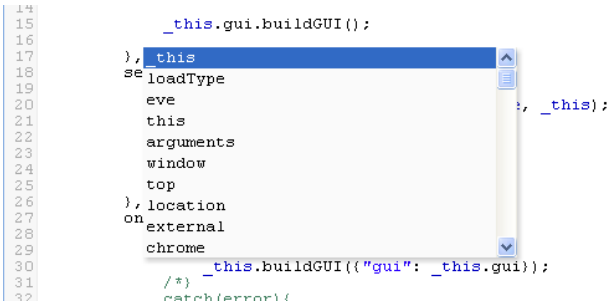

Figura 28 - Ejemplo de auto completamiento de código fuente.

3.2.2.3. Validación y modo depuración de errores

El depurador es sumamente importante en el IDE, como se hizo alusión anteriormente en el apartado correspondiente en este trabajo. El modo de funcionamiento de un depurador de manera genérica es el siguiente:

Primeramente el depurador ejecuta el programa a depurar, el mismo se ejecuta normalmente hasta que el depurador detiene su ejecución, permitiendo al usuario examinar la situación (52).

El depurador permite detener el programa en:

1. Un punto determinado mediante un punto de ruptura.
2. Un punto determinado bajo ciertas condiciones mediante un punto de ruptura condicional.
3. Un momento determinado cuando se cumplan ciertas condiciones.
4. Un momento determinado a petición del usuario.

Durante esa interrupción, el usuario puede:

1. Examinar y modificar la memoria y las variables del programa.
2. Examinar el contenido de los registros del procesador.
3. Examinar la pila de llamadas que han desembocado en la situación actual.
4. Cambiar el punto de ejecución, de manera que el programa continúe su ejecución en un punto diferente al punto en el que fue detenido.
5. Ejecutar instrucción a instrucción.
6. Ejecutar partes determinadas del código, como el interior de una función, o el resto de código antes de salir de una función.

El depurador depende de la arquitectura y sistema en el que se ejecute, por lo que sus funcionalidades cambian de un sistema a otro. En este punto, se han mostrado las más comunes (52).

3.3. Conclusiones parciales

[10:56:18] Antonio Membrides Espinosa: para que tengas en cuanta el patrón que utilizas en la presentación para construir las interfaces se conoce como ¨inyección de vista¨ asi que ya sabes debes ir anotando esas cositas para que no se te queden, la otra e s que lo que me pedias de linker para el cliente lo voy a implementar pero mas salvaje y basado mas bien en el patron arquitectonico MVVM propuesto por microsoft para los temas GUI

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

En este capítulo se efectua un análisis a la varias técninas, métodos, herramientas y funcionalidades que se emplean en la implementación del sistema de acuerdo a lo descrito en los modelos de diseño. También se analizarán algunas técnicas a emplear así como los resultados obtenidos al realizar la fase de prueba al software.

3.4. Nivel 2

Contenido.

2.9. Conclusiones parciales

Contenido.

CONCLUSIONES GENERALES

Conclusiones.

RECOMENDACIONES

Recomendaciones.

REFERENCIAS BIBLIOGRÁFICAS

1. **Pressman, Roger.** *Ingeniería del Software: Un enfoque práctico.* España : McGraw-Hill Companies, 2002. 5ta Edición.

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

3. **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/.

4. **Rouse, Margaret.** SearchSoftwareQuality. [En línea] WhatIs.com. [Citado el: 06 de Diciembre de 2012.] http://searchsoftwarequality.techtarget.com/definition/.

5. **The McGraw-Hill Companies, Inc.** Chapter 3. *BASIC APPLICATION SOFTWARE.* 2004.

6. **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.

7. **TechTarget y Rouse, Margaret.** 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.

8. **ProgramacionDesarrollo.es y editorbfb.** 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/.

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

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

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

12. **Reynoso, Carlos y Kiccillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

13. **Szyperski, Clemens Alden, Gruntz, Dominik y Murer, Stephan.** *Component Software - Beyond Object-Oriented Programming – Second Edition.* s.l. : Addison-Wesley / ACM Press, 2002. 0-201-74572-0.

14. **Sileika, Rytis.** *Pro Python System Administration.* New York : Apress, 2010. 978-1-4302-2605-5.

15. **TechTerms.com.** techterms.com. *Template.* [En línea] TechTerms.com. [Citado el: 11 de Diciembre de 2012.] http://www.techterms.com/definition/template.

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

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

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

19. **Clifton, Marc.** 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.

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

21. **Jimenez, Monica Talledo.** *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.

22. **Microsoft.** 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.

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

24. **Stefanov, Stoyan.** *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.

25. **Qt Project.** 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.

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

27. **Potencier, Fabien.** *El Tutorial Jobeet.* 2009.

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

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

30. **Ziller, Eike.** 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/.

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

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

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

34. **Aptana.** Aptana Studio 3. [En línea] http://www.aptana.com/products/studio3.

35. **Zend.** Zend Studio - The Leading PHP IDE. [En línea] [Citado el: 18 de Diciembre de 2011.] http://www.zend.com/en/products/studio/.

36. **WebDevStudio Team.** Welcome to WebDevStudio. [En línea] [Citado el: 21 de Diciembre de 2011.] http://gayuba5.datsi.fi.upm.es/~iortiz/webdevstudio/modules/home/index.php.

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

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

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

40. **Proyecto Lycan.** Lycan GENESIS - Component Builder. [En línea] [Citado el: 24 de Septiembre de 2012.] http://comunidades.uci.cu/projects/lycan-ide.

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

42. *The Rational Unified Process: An Introduction.* **Kruchten, P.** s.l. : Addison Wesley, 2000.

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

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

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

46. **Arias Marin, Marvin David.** 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.

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

48. **Hardz Website.** 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/.

49. **Apache Software Foundation.** 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.

50. **Vizcaíno, Aurora, García, Félix Oscar y Caballero, Ismael.** *Una Herramienta CASE para ADOO: Visual Paradigm.* Ciudad Real : Universidad de Castilla-La Mancha.

51. **Virtual Paradigm International.** 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/.

52. **Blanco, Carlos.** 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/.

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

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

55. **Bolton, David.** 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.

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

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

58. **WikiMedia.** Wikipedia. [En línea] MediaWiki. [Citado el: 6 de Diciembre de 2012.] http://en.wikipedia.org/wiki/Application_software.

BIBLIOGRAFÍA CONSULTADA

1. **Bolton, David.** 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.

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

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

4. **WikiMedia.** Wikipedia. [En línea] MediaWiki. [Citado el: 6 de Diciembre de 2012.] http://en.wikipedia.org/wiki/Application_software.

ANEXOS

Anexo 1. Nombre del anexo

GLOSARIO DE TÉRMINOS

Término: significado (sin referencias).

***logs*:** registros de sucesos del sistema.

PAC: Los Puntos de Acceso Común son empleados por los proyectos de Ksike para cargar y ejecutar los componentes del cliente, así como direccionar las peticiones AJAX hasta el Controlador Frontal de Ksike.