Universidad de las Ciencias Informáticas

Sistema de Planificación Inteligente.

Módulo de Planificación

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

Autores:

René Villalta Soto

José Enrique Benitez Jimenez

Tutores:

Lic. José Albert Cruz Almaguer

Ing. Norges Sánchez Tumbarell

Junio 2009

"La mejor forma de predecir el futuro es implementarlo"

David Heinemeier Hansson
(Creador Del framework web Ruby on Rails).

Agradezco a mis tutores por toda la ayuda que me han brindado, a la Comunidad Erlang y Mozart, a mis compañeros de tesis, a mis abuelos y madre que sin ellos esto no hubiese sido posible por todo el apoyo moral que me han brindado.

René Villalta Soto

Agradezco a mi familia por todo el apoyo moral que me han brindado siempre, mis tutores Lic. José Albert Cruz Almaguer y al Ing. Norges Sánchez Tumbarell por su ayuda y apoyo en estos meses de trabajo, a la Comunidad de Erlang y Mozart por sus contribuciones, a mis compañeros de tesis, mis compañeros de grupo y proyecto, a mis amigos sin su ayuda este trabajo de diploma no se hubiera realizado.

José Enrique Benitez Jimenez

Dedico este trabajo de Diploma, a todas las personas que han tenido una gran confianza en mi, a mis amigos: Elito, Ledif, Rey, Pedri, Rapha, mis familiares que son muy especiales y sin ellos hubiese, a mi madre por estar siempre ahí y ser madre y padre que nada fácil es, a mi abuelo macho por el ejemplo que me ha dado, y a mi abuela mene por nunca dejarme solo y siempre creer en mi, y por ultimo a mis hermanos que como mayor me corresponde darles el ejemplo y espero nunca decepcionarlos, a todos dedico el presente.

René Villalta Soto

A mi madre Miriam por el cariño incondicional que siempre me ha brindado, a mi padre Enrique por ser mi ejemplo y apoyarme en todo momento, a mis hermanas Marilyn y Zulema que siempre se preocupan por mí, las quiero mucho, a mis sobrinos Adrian, Hermis y Zuliet, ustedes son la razón de mi existencia…

José Enrique Benitez Jimenez

DECLARACIÓN DE AUTORÍA

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

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

Autores:

_____

René Villalta Soto.

_____

José Enrique Benítez Jiménez.

Tutores:

_____

José Albert Cruz Almaguer.

_____

Norges Sánchez Tumbarell.

RESUMEN

Módulo de planificación, es un módulo de gestión de horarios a gran escala implementado utilizando varios conceptos y paradigmas de programación entre los cuales se encuentran el concurrente, distribuido, y declarativo, el mismo fue creado con el objetivo de resolver lo que hoy mundialmente es un problema NP completo, el problema de la planificación de actividades.

El módulo está conformado de la forma más genérica posible, en la cual establecer un cambio es fácil. Permite la interoperabilidad con Erlang estableciendo la comunicación mediante socket y permitiendo integrarse así de una forma sencilla con el sistema, el mismo es un motor implementado en Mozart/Oz mediante el paradigma programación con restricciones.

El motor de planificación soporta un gran número de actividades y recursos. La técnica utilizada permite converger de manera rápida a una solución, además mientras más afectaciones se le proporcionen a la planificación mayor será dicha convergencia.

El módulo de planificación puede ser utilizado desde varios sistemas operativos y es adaptable a cualquier interfaz que cumpla los requisitos exigidos por el mismo. Además es posible disminuir el tiempo de respuesta del algoritmo mediante la utilización de motores de búsquedas paralelo y distribuido.

En el documento se brindan detalles específicos de las funcionalidades que debe ofrecer el módulo de planificación así como el diseño del mismo que facilite su implementación.

PALABRAS CLAVE

Inteligencia Artificial, Programación con restricciones, Planificación, Generador de Horarios, Erlang, Mozart/Oz.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El desarrollo de la informática ha permitido encontrarle solución a una gran variedad de problemas que en el pasado fueron calificados de imposibles. Planificar bien el tiempo es sin duda alguna un problema difícil, más cuando se trata de una institución como la Universidad de las Ciencias Informáticas en la cual se realizan un sin números de actividades donde su planificación es compleja, dinámica y altamente propensa a irregularidades y descoordinaciones que tienden a afectar el Proceso Docente Educativo, ya que las mayorías de estas actividades son de carácter docente.

Debido al desarrollo que ha alcanzado en la actualidad el software y el hardware, se le puede dar una solución factible a este tipo de problema aunque aun sigue siendo un problema NP Completo, lo cual significa que puede tomar un tiempo computacional considerable ya que son complejos y de naturaleza combinatoria.

En la UCI ocurren muchos eventos a demás de las actividades docentes los cuales se nombraran eventos generales, por ejemplo los Festivales de Artistas Aficionados, la Jornada Científica, Uciencia, las Copas de Programación e Ingeniería de Software, los Juego Deportivos, así como talleres y conferencias de invitados, aunque todos estos eventos tienen una planificación inicial, están propensos a cambios que ocurren en su mayoría de forma espontánea lo cual es un verdadero problema para los planificadores de la universidad y esto conlleva a tener que posponer o suspender otras actividades, por tales razones se plantea la necesidad de implementar un módulo de planificación que junto a otros conformen el Sistema de Planificación Inteligente que brinde una planificación tanto de actividades docentes como de eventos generales, y permita realizar cambios por parte de los planificadores y consulta por parte de los estudiantes, profesores y personas implicadas en general.

A partir de la situación problémica definida anteriormente el problema científico se centra en la necesidad de implementar una aplicación que mejore el proceso de planificación de actividades docentes y eventos generales en la UCI.

El **objeto de estudio** se centra en los procesos de planificación de actividades cuyo **campo de acción** es la planificación de actividades docentes y eventos generales en la UCI.

Como **objetivo general** se propone implementar una aplicación que le permita al planificador organizar las actividades docentes y de eventos generales en la UCI. Para complementar el objetivo de la investigación se han formulado los siguientes objetivos específicos que posibilitan el desarrollo del mismo:

- Definir el paradigma o enfoque a utilizar para dar solución a los problemas de planificación.
- Modelar el problema de planificación mediante el paradigma seleccionado.
- Implementar el módulo con una tecnología que soporte el paradigma seleccionado.

Para cumplir los objetivos y resolver la situación problémica planteada se debe seguir las siguientes **tareas investigativas**:

- ✓ Estudiar del estado del arte sobre la optimización de problemas con restricciones.
- ✓ Estudiar los principales enfoques a la solución de los problemas de planificación: Investigación de Operaciones (Programación Lineal), técnicas de diseño de algoritmos (backtracking, branch & bound) y programación con restricciones (lógica y concurrente).
- ✓ Estudiar las teorías matemáticas sobre la satisfacción de restricciones.
- ✓ Analizar las principales tecnologías de desarrollo para la programación con restricciones.
- ✓ Obtener habilidades para la Programación Concurrente con Restricciones (Mozart/Oz) y Programación Funcional (Erlang).
- ✓ Crear una Interfaz de aplicación en Erlang que permita la comunicación con el módulo de Planificación.
- ✓ Modelar el dominio de aplicación mediante programación con restricciones.
- ✓ Implementar el modelo realizado en el lenguaje Oz.

Se plantea la siguiente **hipótesis**: Con la implementación del Módulo de planificación, del Sistema de Planificación Inteligente (SPI) se informatizará el proceso de planificación de actividades docentes y eventos generales en la UCI.

Variable independiente: implementación del Módulo de planificación del SPI.

Variable dependiente: perfeccionar el proceso de planificación de actividades docentes y eventos generales en la UCI.

Variables	Dimensiones	Indicadores	Índice de los indicadores
Implementación del Módulo de planificación del SPI.	Factibilidad	Tiempo de Desarrollo	Extenso
		Costo	Moderado
		Esfuerzo	Breve
	Rendimiento	Tiempo de Respuesta	Mínimo
			Medio
			Alto
Perfeccionar el proceso de planificación de actividades docentes y eventos generales en la UCI.	Mejor Rendimiento	Complejidad	Alta
			Media
			Baja
		Control	Bueno
			Malo
		Importancia	Alta
			Media
			Baja
		Tiempo de ejecución	Rápido
			Medio

Tabla 1. Operacionalización de las variables.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción.

En este capítulo se realiza una descripción de los principales conceptos que intervienen en el tema de la planificación de actividades, se ofrece una breve panorámica actual de diferentes sistemas existentes que planifican. Además de brindar las ventajas de la propuesta del módulo de planificación para el Sistema de Planificación, se describen los lenguajes de programación más adecuados, así como la metodología de desarrollo de software propuesta. Se pretende sentar las bases teóricas para un correcto análisis del módulo de planificación.

1.2 Proceso actual de planificación.

El proceso de planificación es variado en cada facultad, pero de modo general se cumple que, el planificador recibe el listado de locales, p1 de cada asignatura y los profesores con sus respectivas afectaciones, departamento al que pertenece, asignatura que imparte, y si es profesor guía o no, después realiza la distribución de los mismos según la planificación de los turnos de clase para cada grupo, asignándole un local para realizar dicha actividad docente, la información por lo general se almacena en un documento digital y es enviado a los estudiantes y profesores por correo electrónico. De existir cambios debido a algún acontecimiento, los profesores de acuerdo con los estudiantes se comunican con el planificador y este reajusta el horario de forma tal que puedan recuperar el turno perdido.

1.3 Estado del Arte.

La planificación es un proceso muy estudiado en el mundo de la informática, existen disímiles de investigaciones que proponen soluciones a problemas de planificación en general, a continuación se describirán algunos de los métodos y tecnologías que constituyen la tendencia actual a la hora de crear programas o algoritmos que permitan planificar actividades:

- **EAH** (Elaborador automático de horarios) es un sistema experto que reproduce el conocimiento adquirido tras años de elaborar los horarios de la Escuela Superior de Informática de la Universidad Europea de Madrid (UEM). El conjunto de restricciones que impone la filosofía de la Universidad introduce complejas limitaciones diseñadas para minimizar el tiempo que pasan los estudiantes en el Campus y los conflictos entre asignaturas. El sistema está implementado en una herramienta basada en reglas que razona hacia delante, cuya ejecución es controlada oportunistamente por las prioridades asignadas a las reglas. Mejora los mecanismos genéricos de elaboración de horarios por cuanto que permite la introducción de restricciones concretas que determinan la forma del horario resultante.

- **SIGEH** (Sistema Generador de Horarios), sistema web que brinda servicio en la universidad autónoma de baja California, para la Facultad de Ciencias permite la creación de horarios para diferentes carreras [22].

- **GENHOR** (Generador de Horarios para Centros Docentes) software que permite la planificación de horarios docentes [12].

- **GHC** (Generador de Horarios para Centros de Enseñanza) permite la confección, edición y transferencia de horarios escolares semanales. Contiene un potente motor que genera resultados óptimos. Esta limitado a solo 10 profesores [11].

- **KRONOWIN-M-6**, interesante software generador de horario y con algunas otras funcionalidades docentes de interés, creado por la empresa (ADOSSIS, S.A.) Sistemas Informáticos [19].

- **Cronos**, sistema que permite generar de horarios de clase para las Unidades Educativas de Educación Básica y Diversificado [5].
- **GP-Untis**, es un software generador de horarios muy usado internacionalmente, su algoritmo de planificación se basa en algoritmos heurísticos genéticos y lineales, el manejo de los datos es regido por un número limitado de restricciones que el usuario determina [13].
- **ASC** Horarios, software generador de horarios, Para asignaturas, clases, aulas y profesores individuales puedes fijar el tiempo para la enseñanza o cuando el aula o el profesor estén libres [1].
- **Timetab**, potente software generador de horarios escolares sencillos de manejar y presenta un asistente para la introducir los datos y permite generar rápidamente un horario para un centro educativo Presenta como limitación que solo es aplicable a un máximo de 4 grupos y 6 profesores. Compatible para sistema operativo: Windows en sus versiones 95/98/Me/NT/2000/XP además de ser un software propietario [24].
- **Lantiv TimeTabler**, generador de horarios creado en Israel por Lantiv Internacional, presenta licencia de software libre pero está creado para planificar en el orden de los cientos de actividades no de los miles. Solo se encuentra disponible para el sistema operativo Windows y no permite importar de una base de datos previamente creada [25].
- **Mimosa software**, aplicación planificadora de cursos, utilizada en más de 60 países, permite la creación de calendarios de forma automática o interactiva. Funciona sobre el sistema operativo Windows solamente [26].
- **Event Management System (EMS)**, creado por la compañía Dean Evans & Associates, permite la gestión de calendarios utilizando una interfaz web. Presenta módulos específicos para cada tipo de institución, especialmente para universidades, colegios y empresas [27].

En varias facultades de la Universidad de las Ciencias Informáticas se utiliza un software denominado “Planificador de Horarios Docentes”, dicho software permite gestionar la planificación de 20 semanas, auxiliando al planificador para evitar irregularidades en el horario docente, aunque no permite la planificación de forma automática.

Los sistemas de planificación estudiados no son adecuados para resolver la problemática planteada, debido a que la mayoría de ellos no permiten la planificación de grandes cantidades de recursos, la realización de cambios imprevistos y la introducción de restricciones se torna muy complejo. La UCI tiene características muy específicas, debido a la gran cantidad de estudiantes, profesores y trabajadores que posee la misma, así como también la inmensa cantidad de recursos que esta involucra, por tal motivo el proceso de planificación es dinámico y propenso a cambios de última hora. Los sistemas mencionados anteriormente se basan en una planificación estática y aunque la mayoría presentan de alguna manera, un control de cambios, el mismo no es lo suficientemente dinámico como para adaptarse adecuadamente a las condiciones de la UCI.

Después de que se realizara una investigación de los sistemas de planificación que existen en la actualidad, y de concluirse que no son totalmente adecuados para ser utilizados en la UCI, se inicio una investigación de los principales algoritmos y técnicas que permiten darle solución a este tipo de problema.

Los problemas de planificación de recursos pueden ser modelados como problemas de optimización. Existen tres ramas fundamentales de la ciencia que estudian los problemas de optimización estas son: Inteligencia Artificial, Investigación de Operaciones y Programación con Restricciones. Dentro de la rama de la Inteligencia Artificial dichos problemas son atacados utilizando diversas técnicas como: los sistemas expertos, algoritmos genéticos y búsquedas heurísticas. Todos los años se proponen nuevas variantes en conferencias mundiales sobre estos temas. La Investigación de Operaciones permite modelar estos problemas con programación entera y utiliza varios algoritmos para su resolución, ejemplo de estos algoritmos es el Simplex muy conocido a nivel mundial. Desde el punto de vista práctico, la Programación con Restricciones es una herramienta muy útil ya que permite modelar y resolver estos problemas fácilmente, utilizando técnicas eficientes y robustas además de aprovechar la potencia del hardware.

A continuación se destacan algunos de los algoritmos más utilizados para la resolución de este tipo de problemas.

Búsqueda TABU (*TABU Search*), es un método de optimización matemática, perteneciente a la clase de técnicas de búsqueda local. La búsqueda tabú aumenta el rendimiento del método de búsqueda local mediante el uso de estructuras de memoria, una vez que una potencial solución es determinada, se la marca como "tabú" de modo que el algoritmo no vuelva a visitar esa posible solución. La búsqueda tabú

es atribuida a Fred Glover, es un algoritmo meta heurístico que puede utilizarse para resolver problemas de optimización combinatoria, tales como el problema del viajante de comercio (*Travelling Salesman Problem*). La búsqueda tabú utiliza un procedimiento de búsqueda local o por vecindades para moverse iterativamente desde una solución x hacia una solución x' en la vecindad de x, hasta satisfacer algún criterio de parada. Para poder explorar regiones del espacio de búsqueda que serían dejadas de lado por el procedimiento de búsqueda local.

Algoritmo Vértice- Coloración, consiste en crear un grafo (ver Figura 1) en el que los vértices sean las actividades y las aristas supondrán que las dos actividades están unidas lo que significa que no pueden ir a la misma hora, ya que uno o más recursos son utilizados por estas actividades a la misma vez. Cada recurso tiene una lista de actividades a las que esta vinculado, y se hace un recorrido de todos los recursos y las actividades en las que este vinculado un mismo recurso se le pone una arista entre ese par de actividades, indicando que esas dos actividades no pueden ocurrir a la misma hora. De esta manera se obtiene otro grafo, pero con tan solo las relaciones existentes entre todas las actividades.

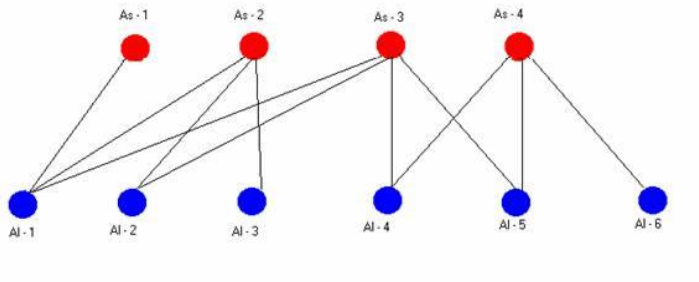

Figura 1. Grafo del algoritmo Vértice-Coloración.

El Sistema de Planificación Inteligente fue creado tomando en cuenta las limitaciones y ventajas de sistemas y algoritmos existentes, el primer objetivo que se persiguió fue crear un sistema que fuera capaz de planificar una gran cantidad de actividades y recursos y que además fuera capaz de ser adaptado a cualquier entorno; un objetivo secundario fue lograr varias interfaces amigables de escritorio con las cuales las personas encargadas de utilizar el sistema pueden planificar, y una aplicación web para las personas que van a consultar su calendario de actividades. Se aprovecho las ventajas de las aplicaciones de escritorio, ya que la persona encargada de planificar necesita manejar varias entidades de forma

cómoda, usando arrastrar y soltar (drag and drop) por ejemplo. Debido a que deben ser mucho menos las personas que planifican a las que consultan su calendario, se escogió utilizar la web para este fin ya que la ventaja más notable de la web es la difusión, o sea es más practico utilizar un navegador que normalmente ya viene con el sistema operativo a utilizar una aplicación extra. Debido a que uno de los objetivos principales del sistema era, como ya se mencionó anteriormente, lograr adaptarlo a cualquier entorno se decidió utilizar términos genéricos, el significado específico de una entidad esta determinada por la lógica del negocio y no acoplada por el sistema, si en una institución determinada se habla en términos de conferencias, seminarios etc. y en otra en términos de reuniones, exposiciones las dos pueden usar el sistema sin ningún problema. Desde el punto de vista técnico el querer utilizar el sistema en cualquier entorno tiene un significado diferente, por tal motivo se veló por que todas las tecnologías estuvieran bajo licencia de software libre y además para una segunda fase el sistema, fuera orientado a componentes de forma tal que sus parte puedan ser acopladas con otras partes de sistemas que ya estén instalados en una determinada institución.

El mayor reto fue lograr que el sistema fuera capaz de manipular y planificar una gran cantidad de información, según la investigación previa realizada a sistemas parecidos que ya existían esto era casi imposible, por eso se decidió pensar fuera de la caja (*Thinking outside the box*) y buscar la solución en lugares donde nadie o casi nadie había buscado. Los sistemas de generación de horarios que se investigaron estaban hechos con “tecnología convencional” o sea con los lenguajes de programación más populares por lo tanto se buscó en los menos populares. Se investigaron lenguajes de programación que eran conocidos por la comunidad de programadores como “lenguajes oscuros”, obviamente por el hecho de que no son populares, pero el resultado de dicha investigación señaló a dos tecnologías que a pesar de no ser populares son líderes en programación concurrente, distribuida, tolerante a fallas, crear sistemas con respuesta en tiempo real y además una de ellas tiene todo un paradigma de programación donde crear un algoritmo de planificación es algo ordinario, poniendo a disposición del software todo la potencia del hardware, ya que permite utilizar todos los procesadores del que disponga la computadora y utilizar varias computadoras a las vez, o sea que el límite del algoritmo estaba en cuantas computadoras usaras.

1.4 Metodologías de desarrollo de software.

Con el objetivo de crear y mantener las aplicaciones de software, aplicando las tecnologías y prácticas computacionales, surge la Ingeniería de Software. El desarrollo y evolución constante experimentada por los procesos de Ingeniería de Software, ha traído consigo la realización de varias tareas en este campo, como son: análisis de requisitos, especificación, diseño y arquitectura, programación, prueba, documentación y mantenimiento.

El proceso de desarrollo del software, define el conjunto de actividades precisas para convertir los requisitos de los usuarios en el conjunto seguro y resistente de artefactos que componen un producto de software. Las tendencias presentes, luego del perfeccionamiento de los procesos del software durante años, han llevado a cabo dos corrientes significativas: los llamados métodos ligeros y métodos pesados. Aunque ambos están enfocados a beneficiar la labor de aquellas personas que intervienen en el proceso de desarrollo.

Los métodos ligeros o ágiles, proponen mejorar la calidad del software teniendo como premisa la comunicación inmediata y directa, mientras que los métodos pesados obtienen sus resultados a través de orden y documentación.

Se hace necesario definir metodologías para guiar el proceso de desarrollo de un producto de software. Las metodologías se definen por pasos a seguir para el cumplimiento de un objetivo. El objetivo dentro del desarrollo del software es producir un producto de alta calidad que cumpla con los requerimientos del cliente.

Las metodologías de desarrollo de software que se encontraron:

- XP (eXtreme Programing)
- FDD (Feature Driven Development)
- MSF (Microsoft Solution Framework)
- RUP (Rational Unified Process)
- METRICA3

- SCRUM
- Crystal
- DSDM(Dynamic Systems Development Method)

Entre las características generales que presentan dichas metodologías se encuentran las siguientes:

- No pueden aplicarse a todo tipo de proyectos.
- Están orientadas en función de los nuevos principios de desarrollo del software.
- Pueden ser ajustables de acuerdo a las características del proyecto.

RUP

RUP es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del Lenguaje Unificado de Modelado (UML), y trabajo de muchas metodologías utilizadas por los clientes. Como proceso define como sus principales elementos:

- **Trabajadores ("quién"):** Define el rol de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- **Actividades ("cómo"):** Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos
- **Artefactos ("qué"):** Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.
- **Flujo de actividades ("Cuándo"):** Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo, los 6 primeros son flujos de ingeniería y los tres últimos de apoyo.

Cada flujo de trabajo cumple con algunas actividades específicas. En el funcionan trabajadores específicos y producen y consumen artefactos también definidos.

Cada fase representa un estado del proyecto, y produce un hito que sirve de entrada a la próxima fase. Todos los flujos se aplican en todas las fases, si bien algunos tienen más carga de trabajo que otros en algunas fases específicas.

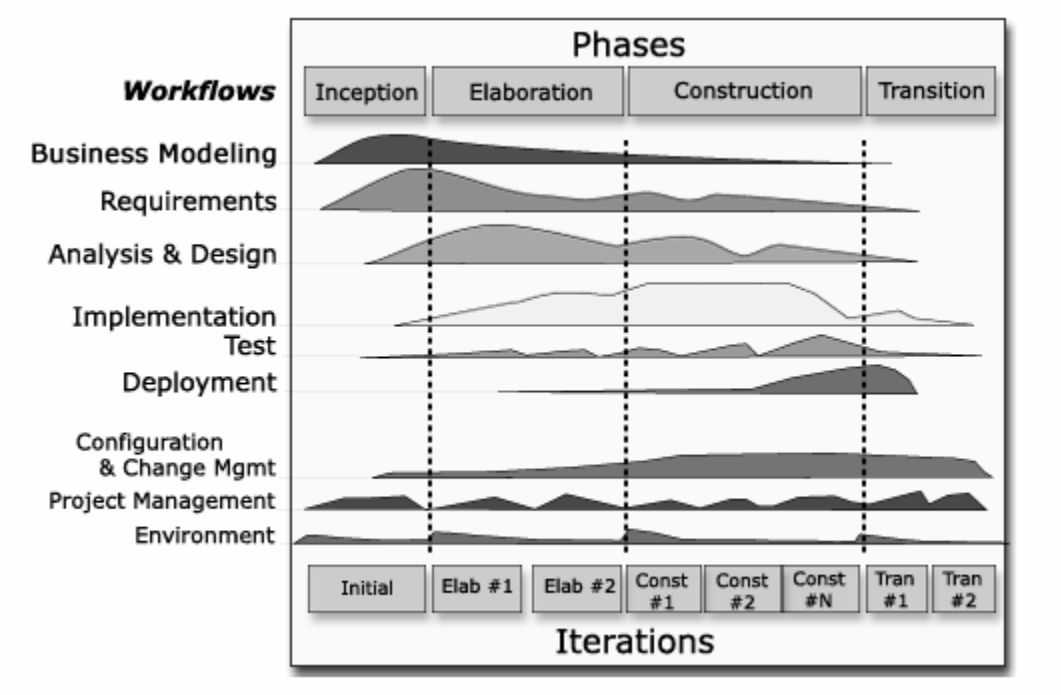

Figura 2. Flujos de trabajo y faces del RUP [15].

El ciclo de vida de RUP se caracteriza por ser dirigido por casos de uso, que reflejan lo que los usuarios futuros necesitan y desean, guiando el proceso de desarrollo ya que los demás artefactos representan la realización de los casos de uso; centrado en la arquitectura que muestra la visión común del sistema completo y describe los elementos del modelo que son mas importantes para su construcción; iterativo e incremental, lo que significa que cada fase se desarrolla en iteraciones que involucran actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros, obteniendo un producto con un determinado nivel que irá creciendo incrementalmente en cada iteración.

SCRUM

SCRUM define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprint, con una duración de 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a

lo largo proyecto, entre ellas destaca la reunión diaria de 15 minutos del equipo de desarrollo para coordinación e integración. Aunque presenta otra serie de características como:

Características:

- Equipos auto dirigidos.
- Utiliza reglas para crear un entorno ágil de administración de proyectos.
- No prescribe prácticas específicas de ingeniería.
- Los requerimientos se capturan como ítems de la lista reserva del producto.
- El producto se construye en una serie de sprint de un mes de duración.
- Usado para proyectos complejos con requerimientos cambiantes.
- Basado en un control de proceso empírico.

Prácticas:

- Iteración (Sprint).
- Reunión de planificación de las iteraciones.
- Reuniones diarias.
- Reuniones de revisión de las iteraciones.
- Reuniones de revisión del diseño.
- Estabilización de las iteraciones.

META SCRUM

- Los proyectos que utilizan Scrum, generalmente poseen las siguientes características:
- Rápido cambio de requisitos.
- No mayor de 10 ingenieros (ideal, con más se forman equipos).

SXP

Esta metodología surgió en Cuba, en la Universidad de las Ciencias informáticas (UCI) y no es más que la unión de XP y Scrum, para el logro de un buen desarrollo de software, propuesta en el 2007 por la ingeniera Malay Rodríguez Villar, y probada en los proyectos que trabajan con Software Libre, obteniendo buenos resultados. Esta metodología surgió con el nombre metodología ágil Gladys Marsi Peñalver Romero UNICORNIOS revisión 2 (MA-GMPR-UR2) la cual fue renombrada poco tiempo después.

La metodología está dividida en cuatro fases, que son precisamente la base de la estructura del nuevo expediente de proyecto, estas son:

- Planificación-Definición.
- Desarrollo.
- Entrega.
- Mantenimiento.

Cada una de estas fases está compuesta por una serie de actividades tales como escribir la visión y escribir la reserva del producto que son las que generan los artefactos como la plantilla de concepción del sistema y la lista de reserva del producto que quedan incluidos en el nuevo expediente de proyecto. Estas actividades están recogidas en el guión de la metodología.

Para la definición de los artefactos que se generan en cada una de las fases se tiene en cuenta como elemento fundamental, las características de las metodologías ágiles, las cuales tienen como premisa la

no duplicación de esfuerzos, así como la integración del cliente en el equipo de desarrollo, esto garantiza que no haya necesidad de documentaciones extensas, sólo se documenta lo necesario para una futura reutilización.

XP

Basándose en la simplicidad, la comunicación y la reutilización del código, surge la Programación Extrema (XP), siendo así una metodología ligera de desarrollo de software. XP consiste en una programación extrema (rápida). Como requisito para alcanzar el éxito del proyecto se tiene al usuario final como parte del equipo. Es una metodología con reconocido éxito y se utiliza en proyectos con entregas a cortos plazos.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

Lo esencial en este proceso de desarrollo es lograr la comunicación entre desarrolladores y usuarios, la retroalimentación entre ellos y con los usuarios finales y la simplicidad en el código.

¿Por qué se utiliza XP?

La metodología XP encaja perfectamente con el tipo de proyecto, las condiciones y con la idea de desarrollo que se tiene del sistema. A continuación, las razones fundamentales que se tuvieron en cuenta al escoger esta metodología.

- **El proyecto es pequeño.** XP está concebida para ser utilizada dentro de proyectos pequeños y de desarrollo rápido se adapta perfectamente a este caso.
- **Empieza en pequeño y añade funcionalidad con retroalimentación continua.** El desarrollo del sistema comienza a partir de los requerimientos básicos y a partir de ahí se van añadiendo funcionalidades que tanto el desarrollador como el cliente entiendan necesarias.

- **Pocos roles.** Esta metodología está dirigida a grupos de desarrollo pequeños y con pocos roles como este caso.
- **El manejo del cambio se convierte en parte sustantiva del proceso**. A medida que el proyecto avanza pueden surgir nuevas expectativas o ideas que pueden ser incorporadas fácilmente permitiéndole mayor adaptabilidad al producto, con la metodología XP esto es completamente factible pues esta se adapta perfectamente a los proyectos cuyos requerimientos cambian a menudo.
- **El cliente o el usuario se convierte en miembro del equipo.** Con el uso de esta metodología y la importancia que esta le concede a la retroalimentación, el cliente es parte del equipo de desarrollo y en este caso que se desarrolla un proyecto para desarrolladores la relación es aún más fuerte.
- **Propiedad colectiva del código.** XP plantea que todos los programadores pueden realizar cambios en cualquier parte del código en cualquier momento. En el proceso de desarrollo con que cuenta la empresa esta es una práctica común.
- **Comunicación de los programadores a través del código.** XP enfatiza el uso de líneas directivas para la codificación que están bien establecidas. Desde sus comienzos la empresa cuenta con una línea directiva para la codificación.

1.5 Programación con restricciones.

La Programación con restricciones es un paradigma de la programación, donde las relaciones entre las variables son expresadas en términos de restricciones (ecuaciones). Actualmente es usada como una tecnología de software para la descripción y resolución de problemas combinatorios particularmente difíciles, especialmente en las áreas de planificación y programación de tareas (calendarización) por lo que es realmente útil para modelar y resolver el problema de planificación de actividades.

Este paradigma representa uno de los desarrollos más fascinantes en los lenguajes de programación desde 1990 y no es sorprendente que haya sido identificada por la ACM (Asociación de Maquinaria Computacional) como una dirección estratégica en la investigación en computación.

Se trata de un paradigma de programación basado en la especificación de un conjunto de restricciones, las cuales deben ser satisfechas por cualquier solución del problema planteado, en lugar de especificar los pasos para obtener dicha solución.

La programación con restricciones se relaciona mucho con la programación lógica y con la investigación de operaciones. De hecho cualquier programa lógico puede ser traducido en un programa con restricciones y viceversa. Muchas veces los programas lógicos son traducidos a programas con restricciones debido a que la solución es más eficiente que su contraparte.

La diferencia entre ambos radica principalmente en sus estilos y enfoques en el modelado del mundo. Para ciertos problemas es más natural (y por ende más simple) escribirlos como programas lógicos, mientras que en otros es más natural escribirlos como programas con restricciones [20].

El enfoque de la programación con restricciones se basa principalmente en buscar un estado en el cual una gran cantidad de restricciones sean satisfechas simultáneamente. Un problema se define típicamente como un estado de la realidad en el cual existe un número de variables con valor desconocido. Un programa basado en restricciones busca dichos valores para todas las variables.

Un programa escrito siguiendo el paradigma de programación con restricciones, representa un problema como este en términos de variables de decisiones y restricciones, y encuentra una asignación para cada variable que satisfaga las restricciones. Programas como estos buscan en el espacio de soluciones sistemáticamente ya sea con algoritmos Backtraking o algoritmos de Ramificación y Poda (Branch and Bound), o utilizan forma de búsquedas locales que pueden ser incompletas. Otros métodos combinan búsqueda con inferencia, donde inferencia consiste en propagar la información contenida en una restricción en las restantes restricciones. Tal inferencia (denominada propagación de restricción) es útil ya que puede reducir el número de búsquedas en el espacio de solución.

Mientras que definir un conjunto de restricciones parece ser una forma simple de definir problemas del mundo real, encontrar un modelo que trabaje bien con un determinado algoritmo no siempre es fácil, un modelo mal elaborado puede ser muy difícil de resolver, se debe tener cuidado a la hora de seleccionar el modelo y el algoritmo adecuado. Programación con restricciones es “programación” en el sentido de “programación matemática” ya que se definen las restricciones como relaciones matemáticas entre las variables de decisión. Sin embargo, programación con restricciones es también “programación” en el

sentido de “programación computacional”, ya que se necesita programar una estrategia de búsqueda que encuentre la solución.

La programación con restricciones es tan general que se puede encontrar en diversas áreas como Álgebra Lineal, Optimización Global, Programación Entera y Programación Lineal, en cada una de estas áreas hay métodos de solución para estos tipos de problemas bien definidos. Es recomendable utilizar un método de solución específico para un dominio determinado de problemas que usar un método general ya que estos últimos por lo general suelen ser menos eficientes.

La definición clásica de un problema de satisfacción de restricciones (PSR) es la siguiente. Un PSR P es una 3-tupla $P = (X, D, C)$ donde X es una n-tupla de variables $X = (X_1, X_2, \dots, X_n)$, D es la n-tupla correspondiente a los dominios $D = (D_1, D_2, \dots, D_n)$, tal que xi € Di, C es la t-tupla de restricciones $C = (C_1, C_2, \dots, C_n)$. Una restricción Cj es un par (Rsj, Sj) donde Rsj es la relación entre las variables en Si, Si es el conjunto de variables involucradas en la restricción Ci. En otras palabras Ri es un subconjunto del producto cartesiano de los dominios de las variables en Si. Una solución al PSR P es una n-tupla $A = (a_1, a_2, \dots, a_n)$ donde ai € Di y cada Cj es satisfecha.

Algunos Dominios de Variables que nos brinda este paradigma son:

- Dominios ***booleanos***, donde solo existen restricciones del tipo verdadero/falso.
- Dominios en variables ***enteras*** y ***racionales***.
- Dominios ***lineales***, donde sólo se describen y analizan funciones lineales.
- Dominios ***finitos***, donde las restricciones son definidas en conjuntos finitos.
- Dominios ***mixtos***, los cuales involucran dos o más de los anteriores.

Los lenguajes de programación con restricciones son típicamente ampliaciones de otro lenguaje. El primer lenguaje utilizado a tal efecto fue *Prolog*. Por esta razón es que este campo fue llamado inicialmente Programación Lógica con Restricciones. Ambos paradigmas comparten características muy similares,

tales como las variables lógicas (una vez que una variable es asignada a un valor, no puede ser cambiado), o el *backtracking*.

La programación con restricciones puede ser implementada como un lenguaje propio o como bibliotecas para ser usadas en algún lenguaje de programación imperativo. Algunos lenguajes populares de programación con restricciones son:

B-Prolog (Basado en Prolog, propietario [2])

CHIP V5 (Basado en Prolog, también existen bibliotecas en C y C++, propietario [6])

Ciao Prolog (Basado en Prolog, software libre: GPL/LGPL [4])

ECLiPSe (Basado en Prolog, software Libre [8])

Mozart (Basado en Oz, software libre: X11 [17])

SICStus (Basado en Prolog, propietario [21])

GNU Prolog (Basado en Prolog, software libre [14])

SWI Prolog Un entorno Prolog que contiene varias librerías para soluciones con restricciones (LGPL) [23].

1.5.1 Aplicaciones de la Programación con restricciones.

Las aplicaciones de respuesta en tiempo real que se han beneficiado de las ventajas técnicas de la programación con restricciones han ido aumentando a pasos agigantados cada año por más de una década. Una gran cantidad de áreas tales como fabricación, servicios financieros, telecomunicaciones y defensa han empleando restricciones y programación lógica.

Las aplicaciones de la programación con restricciones se dividen en 3 grandes ramas, las aplicaciones científicas, las comerciales y las industriales:

Aplicaciones científicas:

- **Restricciones y Biología Molecular:** Las técnicas de la programación con restricciones pueden utilizarse para predecir la estructura de una proteína, este se considera uno de los problemas más importante en la Biología Computacional. El problema de predicción de la estructura de la proteína ha sido transformado a un problema con la reducción del mínimo con dominio finito y variables booleanas. El lenguaje Oz fue utilizado para implementar este problema. Algunas variables se han definido para todo el problema de restricciones de la predicción de la estructura de la proteína. Después se utilizan restricciones de optimización para reducir al mínimo la variable superficie.

- **Restricciones y problemas combinatorios:** La programación con restricciones puede ser una buena herramienta para problemas combinatorios tales como planificación, calendarización y problemas de asignación de recursos. La principal ventaja es que los problemas de toda de decisiones pueden ser expresados como problemas de satisfacción de restricciones.

Aplicaciones Comerciales:

- **XLufthansa:** Un proyecto llamado PARROT fue diseñado e implementado con el objetivo de proveer una manera eficiente de manejar la planificación aérea, combinando técnicas de Investigación de Operaciones y Programación con restricciones. Fue desarrollada una librería para las restricciones de enrutamiento así como un solucionador paralelo diseñado para correr en varias plataformas.

- **Aeropuerto Internacional de Hong Kong:** El problema de la asignación del contador es usualmente encontrado en grandes aeropuertos, se trata de ubicar suficientes contadores dado cualquier aeropuerto (por supuesto en dependencia del tamaño del mismo). Sin embargo la cuestión está en que los contadores de cada vuelo forman una isla y todos los contadores de un vuelo deben ser agrupados en la misma isla. Varios modelos han sido propuestos para atacar este problema.

- **El Consejero de Renta de Múnich (Múnich Rent Advisor):** MRA ganó el premio por mejor aplicación en JFPLC, Clermont Ferrand, Francia, Junio del 1996. MRA es un pequeño sistema experto que permite calcular el costo de la renta de un viaje a Múnich a través de un cuestionario. Este es un buen ejemplo de la combinación de los conceptos de Computación por intervalos y Programación con restricciones. Esta aplicación se caracteriza por un simple servidor web,

restricciones de intervalos sobre ecuaciones lineales, una base de datos de restricciones y algunas reglas if-then-else.

- **Siemens, un proyecto llamado POPULAR:** (Planning of Picocellular Radio Cordless Communication) fue desarrollado en el Departamento de Investigación y Desarrollo para Verificación de Circuitos de Siemens. Esta herramienta optimiza la colocación de los trasmisores de radios para la comunicación inalámbrica local. Dado la blue-print de la instalación y la información sobre los materiales utilizados en paredes y techos, POPULARES calcula el número mínimo de los transmisores de su ubicación y por la simulación y la optimización posterior.

Aplicación Industrial:

- **Proyecto CHIC (Constaint Handling in Industry and Commerce):** Este proyecto tiene como objetivo utilizar los problemas de satisfacción de restricciones para aplicaciones industriales. Se ha desarrollado un sistema que dado los saldos líquidos de los próximos 10 días y las tazas de interés del mercado monetario es capaz de generar las operaciones que cubren las deficiencias y utiliza los excedentes para optimizar los beneficios.

1.5.2 Beneficios de la Programación con restricciones

El algoritmo utilizado por el Sistema de Planificación Inteligente esta basado en la Programación con Restricciones la misma está encargada de resolver problemas de optimización, al igual que la Investigación de Operaciones, la Programación Lineal, la Programación Entera y la Optimización Matemática. A continuación se presenta una tabla obtenida de [3] con casos reales de aplicaciones que utilizan estos algoritmos y los ahorros que los mismos aportaron:

Organización	Aplicación	Año	Ahorros anuales(USD)
The Netherlands Rijkswaterstaat	Desarrollo de la política nacional de administración del agua, incluyendo mezcla de nuevas instalaciones, procedimientos de operaciones y costeo	1985	$15 millones
Monsanto Corp.	Optimización de las operaciones de producción para cumplir metas con un costo mínimo	1985	$2 millones
Weyerhauser Co.	Optimización del corte de árboles en productos de madera para maximizar su producción	1986	$15 millones
Electrobas/CEPAL	Asignación óptima de recursos hidráulicos y térmicos en el	1986	$43 millones

Brasil	sistema nacional de generación de energía		
United Airlines	Programación de turnos de trabajo en oficinas de reservaciones y aeropuertos para cumplir con las necesidades del cliente a un costo mínimo	1986	$6 millones
Citgo Petroleum Corp.	Optimización de las operaciones de refinación y de la oferta, distribución y comercialización de productos	1987	$70 millones
SANTOS, Ltd., Australia	Optimización de inversiones de capital para producir gas natural durante 25 años	1987	$3 millones
Electric Power Research Institute	Administración de inventarios de petróleo y carbón para el servicio eléctrico con el fin de equilibrar los costos de inventario y los riesgos de faltantes.	1989	$59 millones
San Francisco Police Department	Optimización de la programación y asignación de oficiales de patrulla con un sistema computarizado	1989	$11 millones
Texaco Inc.	Optimización de la mezcla de ingredientes disponibles para que los productos de gasolina cumplieran con los requerimientos de ventas y calidad	1989	$30 millones
IBM	Integración de una red nacional de inventario de refacciones para mejorar el apoyo al servicio	1990	$20 millones + $250 millones en menor inventario
American Airlines	Diseño de un sistema de estructura de precios, sobreventas y coordinación de vuelos para mejorar las utilidades	1992	$500 millones más de ingresos
Yellow Freight System, Inc.	Optimización del diseño de una red nacional de transporte y la programación de rutas de envío	1992	$17.3 millones
New Haven Health Dept.	Diseño de un programa efectivo de cambio de agujas para combatir el contagio del SIDA	1993	33% menos contagios
AT&T	Desarrollo de un sistema basado en PC para guiar a los clientes del negocio en el diseño del centro de llamadas	1993	$750 millones
Delta Airlines	Maximización de ganancias a partir de la asignación de los tipos de aviones en 2.500 vuelos nacionales	1994	$100 millones
Digital Equipment Corp.	Reestructuración de toda la cadena de proveedores entre proveedores, plantas, centros de distribución, sitios potenciales y áreas de mercado	1995	$800 millones
China	Selección y programación óptima de proyectos masivos para cumplir con las necesidades futuras de energía del país	1995	$425 millones
Cuerpo de defensa de Sudáfrica	Rediseño óptimo del tamaño y forma del cuerpo de defensa y su sistema de armas	1997	$1.100 millones
Procter and Gamble	Rediseño del sistema de producción y distribución norteamericano para reducir costos y mejorar la rapidez de llegada al mercado	1997	$200 millones
Taco Bell	Programación óptima de empleados para proporcionar el servicio a clientes deseado con un costo mínimo	1998	$13 millones
Hewlett-Packard	Rediseño de tamaño y localización de inventarios de seguridad en la línea de producción de impresoras para cumplir metas de producción	1998	$280 millones de ingreso adicional

Tabla 1.1. Casos reales de aplicación de la Programación con restricciones.

1.6 Lenguajes de Programación

Mozart/Oz

El algoritmo propuesto para realizar la planificación de las actividades del Sistema de Planificación Inteligente se basa en la programación con restricciones, de varios lenguajes que soportan dicho paradigma se escogió el lenguaje “Oz” cuya primera implementación fue liberada en 1995, este provee a los programadores y desarrolladores de sistemas de una gran variedad de abstracciones que les permite crear aplicaciones complejas y robustas. El mismo combina diseños de varios lenguajes de programación en un diseño simple y coherente. Cada paradigma de programación tiene sus beneficios ya sea Orientado a Objetos, Funcional o Programación lógica con restricciones. Al escribir programas en un cierto lenguaje de programación, existen limitaciones por los conceptos de un determinado paradigma, Oz resuelve este problema combinando abstracciones de varios paradigmas de forma sencilla.

Oz presenta características de la Programación Orientada a Objetos, permite trabajar con tipos de datos abstractos, clases, objetos y herencia. También provee variables lógicas y estrategias de búsquedas programables, propias de los paradigmas de Programación Lógica y Programación con Restricciones respectivamente. Además es un lenguaje concurrente donde los usuarios pueden crear dinámicamente un gran número de hilos de ejecución que pueden interactuar unos con otros, sin embargo, a diferencia de los lenguajes de programación convencionales actuales, los hilos de ejecución solo ejecutan una instrucción cuando todas las variables involucradas tienen un dato asignado, esta característica se denomina flujo de datos (*dataflow*).

Una buena implementación de este lenguaje es Mozart/Oz una tecnología que es tolerante a fallos, soporta programación multi-nucleo, programación en tiempo real y programación distribuida, varias computadoras con Mozart/Oz pueden ser conectadas y trabajar como si fueran una sola, compartiendo variables, objetos, clases y procedimientos. Mozart/Oz está disponible para Windows y Linux entre otros sistemas operativos y tiene licencia de software libre, es soportado actualmente por el Consorcio Mozart, organización integrada por desarrolladores e investigadores de DFKI (Centro Alemán de Investigación de Inteligencia Artificial), SICS (Instituto Sueco de Ciencias de la Computación), la Universidad de Saarland,

UCL (Universidad católica de Lovaina) y es frecuentemente mejorado por las contribuciones de miles de usuarios a través de internet.

Erlang

Erlang es un lenguaje de programación concurrente y un sistema de ejecución que incluye una máquina virtual y librerías. El subconjunto de programación secuencial de Erlang es un lenguaje funcional, con evaluación estricta, asignación única, y tipado dinámico. Fue diseñado en la compañía Ericsson para realizar aplicaciones distribuidas, tolerantes a fallos, con respuestas en tiempo real (*soft-real-time)* y de funcionamiento ininterrumpido. Proporciona el cambio en caliente de código de forma que éste se puede cambiar sin parar el sistema. Originalmente, Erlang, era un lenguaje propietario de Ericsson, pero fue cedido como fuente abierta (*open source)* en 1998.

La creación y gestión de procesos es trivial en Erlang, mientras que, en muchos lenguajes, los hilos se consideran un apartado complicado y propenso a errores. En Erlang toda concurrencia es explícita.

La mayor fortaleza de Erlang es el soporte para concurrencia. Tiene un pequeño pero potente conjunto de primitivas para crear procesos y comunicar entre los mismos. Dichos procesos son la forma principal de estructurar una aplicación, y se puede crear un gran número de ellos sin que se degrade el rendimiento (se ha llegado a hacer una prueba con 20 millones de procesos).

El soporte para procesos distribuidos es también parte de Erlang. Los procesos se pueden crear en nodos remotos, y la comunicación con ellos es transparente. Es decir, la comunicación con procesos remotos se hace exactamente de la misma manera que la comunicación con procesos locales.

En este contexto de Programación Funcional donde se encuentran lenguajes como Prolog, Lisp, Erlang y Haskell, a diferencia de los de Programación Imperativa C, Pascal, Fortran y Basic, el programador se olvida de cómo se hacen las tareas, y concentra sus esfuerzos en qué hacer. Los lenguajes de programación funcionales se fundamentan en el concepto de función y razonamiento matemático.

Erlang surgió en la década de los 80 en los laboratorios de la empresa sueca Ericsson, como un intento de desarrollar un lenguaje de alto nivel, estructurado y con capacidad para afrontar el tipo de proyectos, especialmente de Telecomunicaciones, que la empresa estaba desarrollando, y que optimizara el uso de la tecnología emergente en Microelectrónica, especialmente, en Microinformática.

Erlang surge como una solución a los problemas encontrados en cualquier proyecto de Telecomunicación, que también son similares a los encontrados en los sistemas en Tiempo Real ampliamente utilizados en la industria. Necesidades de estructuración, ausencia de errores y funcionamiento continuo, que se resuelven en Erlang mediante la agrupación de código fuente en módulos, un complejo sistema detector de errores (interno al lenguaje) y la posibilidad de carga en caliente, hacen de Erlang un potente y eficaz lenguaje para modelar y resolver problemas de Telecomunicaciones y Tiempo Real, donde la distribución de procesos, robustez y tolerancia a fallos son prioritarios.

Las características más relevantes del lenguaje, son las siguientes:

- **Procesos concurrentes en tiempo real**: Erlang se adapta perfectamente a Sistemas Soft Real-Time, con tiempos de proceso comprendidos en un intervalo de tiempo probabilístico (milisegundos).
- **Sistemas distribuidos**: Diseñado para resolver problemas distribuidos por naturaleza, donde el problema tiene origen distribuido, y no es necesario incorporar la distribución de procesos de forma artificial. Asimismo, está especialmente diseñado para el uso con sistemas compuestos por un elevado número de procesos, donde la creación, ejecución y eliminación de procesos se hace de una forma sencilla y eficaz. El mecanismo de comunicación inter-procesos utilizado es el paso de mensajes asíncrono.
- **Tolerancia a fallos**: La propia naturaleza del sistema, formado por un elevado número de procesos concurrentes y procesados de forma distribuida, así como la capacidad de ejecución de procesos de forma local (nodo local), o distribuir los procesos en máquinas remotas (nodos remotos), en una red de computadores, hacen de Erlang un entorno de programación distribuida y con una alta tolerancia a fallos, al permitir la asignación de procesos a determinados nodos de la red, así como establecer alternativas de ejecución ante problemas o caídas del nodo local de ejecución del proceso.
- Otras características y funcionalidades específicas de Erlang son:
- Lenguaje de alto nivel basado en procesos.
- Las variables no se pueden sobrescribir, para evitar efectos colaterales.

- El código de programa está formado por funciones, agrupadas en módulos.
- Acoplamiento de patrones (*Pattern-Matching*).
- Uso de librerías de programa (OTP, "*Open Telecomunication Platform*").
- Organización de memoria mediante Colector de Basura automática (*garbage collector*).

En general, Erlang representa una elevada potencia de computación y optimización de código basado en un lenguaje declarativo, que al estar más próximo al razonamiento humano, se obtienen códigos de programa más compactos, eficientes y de más fácil comprensión para el ser humano.

1.7 Programación con restricciones en Mozart/Oz.

En Mozart/Oz el modelo de los PSR define que los dominios de las variables son finitos y los valores de los mismos son números enteros no-negativos. Las dos técnicas básicas usadas para resolver estos problemas son la propagación de las restricciones y la distribución de las restricciones.

Existen dos tipos de restricciones, las básicas y las no básicas, las primeras son del tipo X=N, X=Y, X € D, donde D es un dominio finito. Las restricciones no básicas son mas complejas, pueden ser ecuaciones, inecuaciones y funciones que contienen determinados módulos, ejemplos de restricción no básicas serian X<Y, X+20>Y·30.

En Mozart/Oz la arquitectura para resolver los problemas de restricción se denomina **Espacio** el cual tiene una serie de **propagadores** que cada uno se encarga de hacer cumplir una restricción no básica sobre un **almacén de restricciones** que contiene una conjunción de todas las restricciones básicas, ver Fig.2

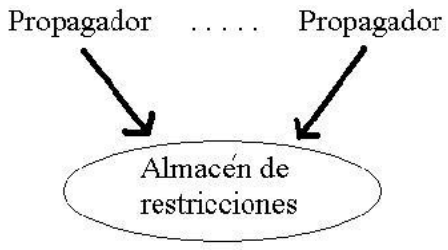

Figura 3. Diagrama que representa un espacio en Mozart/Oz.

$$X \in 0\#5 \ \wedge Y = 8 \ \wedge Z \in 13\#23.$$

Figura 4. Restricciones básicas dentro de un almacén de Restricciones.

Propagación y Distribución

La propagación de las restricciones es un mecanismo de inferencia para problemas de dominios finitos donde se reduce el dominio de las variables, por ejemplo:

Ejemplo 1:

Restricción no básica: **X**<**Y**

Restricción básica 1: **X** € 23#100

Restricción básica 2: **Y** € 1#33

Mediante la propagación con restricciones se deduce que:

X € 23#32 y **Y** € 24#33.

Cada propagador tiene una restricción no básica y todas las restricciones básicas están en forma de conjunción en el almacén de restricciones, pueden existir miles de propagadores influyendo sobre el almacén de restricciones para hacer cumplir su restricción, en el ejemplo anterior solo hay un propagador.

Un propagador es inconsistente si no hay ninguna asignación de variable que satisfaga tanto a las restricciones en el almacén de restricciones como la restricción del propagador ejemplo: **X**+**Y**=6, **X** € 3#9, **Y** € 4#9, en este caso el propagador falla.

Es implícito si cada asignación de variable que satisface las restricciones del almacén de restricciones también satisface la restricción del propagador, ejemplo: **X**<**Y**=6, **X** € 3#5, **Y** € 6#9 en dicho caso el propagador deja de existir.

Es estable cuando no puede aportar información nueva al almacén de restricciones. Un espacio falla si al menos uno de sus propagadores falla, el espacio es estable si todos sus propagadores son estables. Un

espacio es resuelto cuando no fallo y no le quedan propagadores y la solución del espacio se encuentra en el almacén de restricciones.

El ejemplo 1 muestra que hay casos en que la propagación de la restricción no soluciona el problema, en el ejemplo, después de aplicar la propagación las variables redujeron sus dominios pero no hasta un elemento por lo tanto no es solución. La distribución de la restricción conjuntamente con la propagación permite hallar una solución en todos los casos o determinar que el problema no tiene solución.

La distribución de la restricción consiste en asignarle un valor del dominio a una variable de las que están en el almacén de restricciones (por ejemplo **X**), entonces el problema se divide en dos espacios uno donde se añade el propagador con la restricción (**X** =4 por ejemplo) y otro donde se añade un propagador con la restricción **X≠**4. Utilizando la propagación y la distribución se genera todo un árbol de espacios, ver:

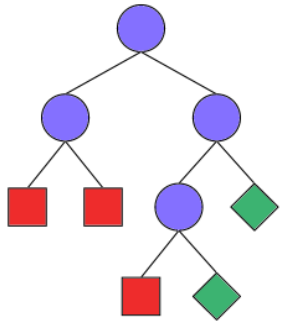

Figura 5. Los círculos representan los espacios estables, los cuadrados los espacios fallidos y los rombos los espacios resueltos dibujados por la herramienta Explorer de Mozart/Oz.

En el siguiente ejemplo intervienen juntas la propagación y la distribución.

Ejemplo 2:

Se posee un espacio que tiene en su almacén de restricciones las variables **X**, **Y**, **Z** con sus dominios, y tres propagadores con las restricciones **X≠**7, **Z≠**2 y **X**-**Z=**3·**Y**.

$$X \neq 7 \qquad Z \neq 2 \qquad X - Z = 3 \cdot Y$$
$$X \in 1\#8 \qquad Y \in 1\#10 \qquad Z \in 1\#10$$

Los propagadores de las desigualdades escriben la información en el almacén de restricciones y enseguida desaparecen, el almacén de restricciones queda de la siguiente forma:

$X \in [1\#6\ \ 8] \qquad Y \in 1\#10 \qquad Z \in [1\ \ 3\#10]$

Donde [1 3#10] denota el dominio {1} U {3,…,10}, el propagador de la restricción **X**-**Z**=3·**Y** ahora puede reducir el dominio de las variables.

$X \in [4\#6\ \ 8] \qquad Y \in 1\#2 \qquad Z \in [1\ \ 3\#5].$

Ahora el espacio es estable por lo tanto aquí es cuando entra la distribución

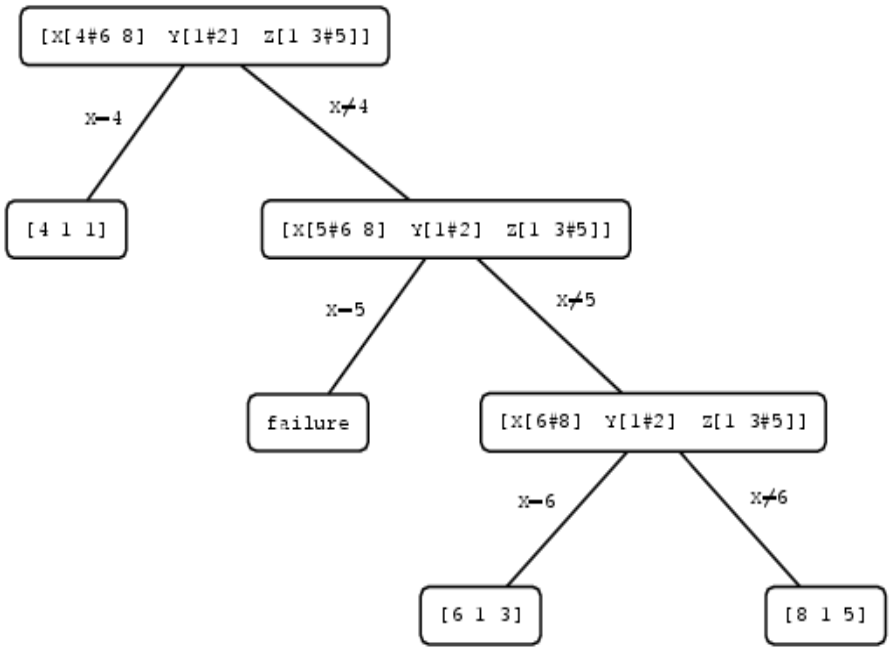

Figura 6. Árbol de propagación y distribución.

Estrategias de distribución

Además de los propagadores existen otras entidades que influyen sobre el almacén de restricciones, estas entidades se denominan distribuidores, estos son los encargados de distribuir el espacio cuando este es estable , si se definen varios distribuidores, en el momento que haga falta distribuir solo se activa uno de forma indeterminada. Para distribuir es necesario seleccionar una variable, de las que no han sido aun resueltas y asignarle un valor de su dominio, para hacer esto se requiere de un criterio que determine cual variable y cual valor de su dominio seleccionar, dicho criterio se denomina estrategia de distribución.

Dos de las estrategias mas conocidas son: estrategia de distribución inocente (*naive*) y estrategia de distribución primero en fallar (*first fail*).

- **Estrategia de distribución inocente**: consiste en seleccionar la variable no determinada más a la izquierda.
- **Estrategia de distribución primero en fallar**: consiste en seleccionar la variable no determinada mas a la izquierda en la secuencia ordenadas por el tamaño del dominio de menor a mayor, o sea que selecciona la variable no determinada con menor dominio para llegar rápido a un estado de fallo.

1.8 Conclusiones.

En este capítulo se han planteado una panorámica de los sistemas y algoritmos que constituyen actualmente la tendencia en el tema de la planificación de actividades, muchos de ellos presentan limitaciones, como es el número de actividades que son capaces de planificar. También se ha descrito brevemente Scrum una metodología de desarrollo de software que entra en la clasificación de las metodologías agiles. Se explica el paradigma de programación con restricciones, el cual permite resolver problemas de satisfacción de restricciones (PSR), dentro de los cuales se encuentra el problema de la planificación de actividades. Se representaron las principales características de los lenguajes propuestos para la realización del módulo de planificación, además se mostro como el lenguaje Oz implementa el paradigma de programación funcional, estas son las bases teóricas necesarias en la solución propuesta para el módulo de planificación del Sistema de Planificación Inteligente.

CAPÍTULO 2. CARACTERÍSTICAS DEL SISTEMA.

2.1 Introducción.

En el capítulo siguiente se establece como objetivo una valoración de las características principales del sistema a desarrollar, siempre prestando gran atención a la situación problémica que dio origen al mismo. Se determinan claramente las necesidades de los usuarios, describiéndose las funcionalidades que serán objeto de automatización y las mismas que permitirán que el sistema una vez logrado presente una buena calidad y un buen funcionamiento. Por último se presentará una propuesta del software a implementar que establezca sobre todo su dominio, un resultado factible referente a las valoraciones iniciales que se tomaron en cuenta, especificando detalladamente los requerimientos funcionales y no funcionales.

2.2 Propuesta del Módulo de Planificación.

El presente trabajo propone la implementación de un sistema que provea nuevas funcionalidades con respecto al flujo de trabajo actual. Dicho módulo debe permitir iniciar una sesión de trabajo para cada cliente, una vez iniciada se podrá configurar de una manera simple el motor de planificación, permitiendo establecer el dominio de tiempo sobre el cual se desea planificar y la estrategia de solución que se desee, los mismos una vez iniciada la sesión también podrán hacer de forma alterna la gestión de actividades, recursos y restricciones, ver el estado de su sesión de trabajo (activo ó inactivo) y según el estado de la misma el sistema brinda la posibilidad de planificar y replanificar, también podrán finalizar la sesión y de esta forma librar los recursos que permitirán una planificación o replanificación por parte de otra sesión activa. Los clientes de planificación podrán planificar solo cuando su estado sea activo, determinando que no hay ningún recurso en planificación por parte de otro cliente que el pueda solicitar previamente al planificar. Cada una de las funcionalidades será implementada en Erlang y Oz.

El sistema contará con los siguientes roles:

- Administrador del Sistema.
- Planificador.

2.3 Personas relacionadas con el módulo.

Se define como persona relacionada al módulo toda aquella que obtiene un resultado del valor de uno o varios procesos que se ejecutan en el mismo. Además de aquellas que se encuentran involucradas en dichos procesos, pues participan en ellos pero no obtienen ningún resultado de valor.

2.4 Lista de reserva del producto.

Una vez conocidos todos los conceptos que rodean al objeto de estudio, se puede analizar qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio de este trabajo. Para ello se enumeran a través de requerimientos funcionales las prestaciones que el sistema será capaz de brindar. Dentro de ellas se incluyen las acciones que podrán ser ejecutadas por el cliente, las acciones ocultas que debe realizar el sistema y las condiciones extremas a determinar por el sistema. De acuerdo a los objetivos planteados, el sistema debe ser capaz de:

1) Iniciar sesión de trabajo.
2) Consultar el estado de la sesión de trabajo.
3) Insertar recurso.
4) Insertar restricción.
5) Insertar actividad.
6) Planificar.
7) Replanificar.
8) Cerrar sección de trabajo.
9) Configurar motor de planificación.

2.5 Requerimientos no funcionales del sistema.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

Usabilidad:

Para utilizar el módulo de planificación es necesario poseer conocimientos básicos de programación funcional en Erlang.

Soporte:

- Sistema multiplataforma.
- Gestor de base de datos funcional, Mnesia, permite alta velocidad de procesamiento de datos, y un tiempo de respuesta rápida en acciones concurrentes.

Software:

- Estaciones de trabajo (PC Cliente)
    - Sistema operativo: Multiplataforma.
- Servidor de Aplicaciones
    - Servidor Erlang.
    - Servidor Mozart.
    - Versión de Erlang 5.7.1.
    - Versión de Mozart/Oz 1.4.
- Servidor de Base de Datos.
    - Sistema Gestor de Base de Datos Mnesia 4.4.9.

Hardware:

- Los servidores Proxy, Web y de Bases de Datos deben poseer 512 MB de memoria RAM como mínimo.

Seguridad:

- Chequear que el motor se encuentre listo para planificar.
- Chequear que se encuentre algún juego de datos sobre los cual establecer la planificación.
- Garantizar que las funcionalidades del sistema se muestren de acuerdo al tipo de usuario que esté activo.
- La información manejada por el sistema estará protegida de acceso no autorizado y divulgación.
- Realizar salvas periódicas de la información en otros dispositivos.
- Llevar un registro de sucesos donde se archiven los eventos del sistema incluyendo los eventos de error, inicio de sesión, cierre de sesión y modificación de la información.

Funcionalidad

- Capacidad de planificación en un tiempo factible.
- Un módulo que brinda funcionalidades para la planificación.

Rendimiento

- La eficiencia de esta aplicación debe ser óptima en cuanto a la velocidad de procesamiento, disponibilidad, tiempo de respuesta y aprovechamiento de los recursos, entre otros.

Disponibilidad:

- El sistema deberá tener un 100% de disponibilidad por lo que podrá ser usado las 24 horas del día por todos sus clientes.

2.6 Factibilidad

Para valorar la utilidad de un sistema de planificación, se realizó una investigación en Internet utilizando como herramienta el famoso buscador Google. La investigación concluyó que existe una gran variedad de sistemas parecidos que planifican desde escuelas primarias hasta universidades. Algunos de los principales, actualmente en el mundo son los siguientes: Lantiv TimeTabler [25], ASC TimeTables [1], Mimosa Software [26], Event Management System-EMS [27] y GP-Untis [13], según la información publicada en dichos sitios se puede concluir que:

- Todos son productos comerciales.
- La mayoría siguen esquemas de subscripción anual para el uso de sus productos, mientras que otros cobran únicamente por sus actualizaciones.
- Los precios varían, pero uno de los mas completos (mimosa) cobra €6,000 (aproximadamente $8,850 USD) para uso de instituciones de mas de 5 mil estudiantes y actualizaciones de por vida, mientras que EMS y GP-Untis no ofrecen información alguna con respecto al precio, este último es utilizado en 80 países.
- La mayoría son productos de escritorios para Windows, implicando el pago de la licencia de dicho sistema operativo.
- Algunos ofrecen maneras complejas de agregar restricciones por el usuario.
- Ninguno ofrece maneras transparentes de acoplarse con sistemas ya existentes.
- La mayoría ofrece la posibilidad de generar asignaciones manualmente y automáticamente; sin embargo la mayoría de las soluciones limitan el número de salones y profesores para poder usar su sistema automático.
- La mayoría de los programas están diseñados para escuelas secundarias o preuniversitarias, donde todos los estudiantes siguen el mismo plan de estudio.

Personal necesario

La creación de un software de este tipo requiere de amplios conocimientos de programación en varios lenguajes. Los programadores y analistas tienen un salario entre €11,000 y €34,000. Según los sistemas estudiados el número mínimo de especialistas necesarios es de 20. Estos datos pueden variar según la tecnología que se use, la cual es generalmente la convencional, con las tecnologías que se usan para la creación del Sistema de Planificación Inteligente el tiempo de desarrollo se acorta considerablemente, y debido a la naturaleza de las mismas, el número de personas también es reducido, ejemplo de esto es que el núcleo del Sistema de Planificación Inteligente fue realizado por 3 personas en 2 meses. La curva de aprendizaje de estas tecnologías se considera de baja a media.

Valor del producto por venta

Como se mencionó al principio de la sección, los precios de estos productos son muy variados oscilando alrededor de los $ 8,500 USD, aunque las principales empresas mantienen sus precios en secretos para negociarlos con los clientes. Existe una gran diversidad de sistemas generadores de horarios y planificadores de recursos, demostrando así la salud del mercado de este tipo de sistemas y la demanda mundial que presentan. Debido a que todas las herramientas utilizadas para la elaboración de este producto son libres y multiplataforma, el mismo presenta una importante ventaja sobre la mayoría de sus homólogos, ya que el cliente no estaría obligado a utilizar sistemas operativos comerciales y pagar sus caras licencias.

2.7 Conclusiones

En este capítulo se inicia el desarrollo de la propuesta de solución que se desea implementar, tras el análisis de los flujos de trabajos actuales descritos por el cliente. Se obtuvo un listado de funcionalidades que debe tener el sistema, expresados en los requerimientos funcionales. Partiendo de este punto, base de todo proceso de desarrollo, se puede comenzar con la construcción de la propuesta, velando por el cumplimiento de todos los requerimientos y funcionalidades consideradas. Para la obtención de un producto bien logrado, el proceso de desarrollo se debe guiar continuamente por el flujo de procesos establecidos, y fomentarse sobre las base de sus requerimientos no funcionales para valorarse la calidad del mismo en cualquier plano que se desarrolle. Se muestra un breve estudio de la factibilidad del sistema, el cual muestra cuan factible es el desarrollo del mismo con respecto a otros sistemas similares a nivel mundial, también muestra los beneficios que otorga una vez terminado.

CAPÍTULO 3. EXPLORACIÓN Y PLANIFICACIÓN DEL SISTEMA

3.1 Introducción

En el presente capítulo se hace alusión a las fases de exploraciones y planificaciones propias de la metodología de desarrollo utilizada para la implementación del sistema que se propone. Se exponen además los artefactos generados durante el transcurso de las mismas.

3.2 Fase de exploración. Definición

El ciclo de vida de XP enfatiza en el carácter interactivo e incremental del desarrollo. Una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas, que en el caso de XP corresponden a un conjunto de historias de usuarios (Beck, 2000).

La metodología de desarrollo XP comienza con la fase de exploración, en fase los clientes plantean a grandes rasgos las historias de usuario mediante un proceso de identificación que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto.

Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (XP, 2006).

3.3 Historias de usuarios

Las historias de usuarios (HU) representan una breve descripción del comportamiento del sistema empleando una terminología del cliente sin lenguaje técnico. Éstas se realizan una por cada característica principal del sistema y se emplean para hacer estimaciones de tiempo y para el plan de lanzamientos. Una de las ventajas es, que reemplazan un gran documento de requisitos y preceden la creación de las pruebas de aceptación. Durante el análisis en la fase de exploración se identificaron seis HU que a continuación se describen:

Historia de Usuario	
Número: 1	**Nombre:** Insertar recurso
Usuario: Planificador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración asignada:** 1
Descripción: Este proceso define la inserción de recursos al sistema, mediante una sesión de trabajo activa.	
Observaciones: Un recurso puede ser compartido por varias secciones de trabajo.	

Tabla 3.1. Historia de usuario "Insertar recurso".

Historia de Usuario	
Número: 2	**Nombre:** Insertar actividad
Usuario: Planificador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración asignada:** 1
Descripción: Este proceso define la inserción de actividades al sistema, mediante una sesión de trabajo activa.	

Observaciones: Una actividad es única en el sistema.

Tabla 3.2. Historia de usuario "Insertar actividad".

Historia de Usuario	
Número: 3	**Nombre:** Insertar restricción
Usuario: Planificador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración asignada:** 1
Descripción: Este proceso define la inserción de restricciones al sistema, mediante una sesión de trabajo activa.	
Observaciones:	

Tabla 3.3. Historia de usuario "Insertar restricción".

Historia de Usuario	
Número: 4	**Nombre:** Planificar.
Usuario: Planificador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 3	**Iteración asignada:** 1
Descripción: Este proceso define la planificación de actividades.	
Observaciones: Previamente se debe haber provisto al módulo con los datos necesarios para realizar una planificación, estos datos son actividades, recursos y restricciones.	

Tabla 3.4. Historia de usuario "Planificar".

Historia de Usuario	
Número: 5	**Nombre:** Replanificar.
Usuario: Planificador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 2	**Iteración asignada:** 1
Descripción: Este proceso define la re planificación de actividades.	
Observaciones: Previamente se debe haber provisto al módulo con los datos necesarios para realizar una re planificación, estos datos son actividades nuevas de un espacio de trabajo en conjunto con las restantes que pertenecen al mismo, pero de estas ultimas solo la restricción de mantenerse fija.	

Tabla 3.5. Historia de usuario "Replanificar".

Historia de Usuario	
Número: 6	**Nombre:** Iniciar Sesión de trabajo.
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración asignada:** 2
Descripción: Este proceso define como se inicia una sesión de trabajo en el módulo de planificación de actividades. La sesión permite al usuario acceder a las otras funcionalidades que brinda el módulo	
Observaciones: La sesión de trabajo es necesaria para diferenciar las peticiones de los diferentes usuarios, ya que cuando un usuario manda a planificar las demás secciones deben bloquearse mientras dure el proceso.	

Tabla 3.6. Historia de usuario "Iniciar sesión de trabajo".

Historia de Usuario	
Número: 7	**Nombre:** Consultar el estado de la sesión de trabajo.
Usuario: Administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración asignada:** 2
Descripción: Este proceso define como se consulta el estado de la sesión de trabajo.	
Observaciones: La sesión de trabajo es necesaria para diferenciar las peticiones de los diferentes usuarios, ya que cuando un usuario manda a planificar las demás secciones deben bloquearse mientras dure el proceso. Es necesario consultar el estado de la sesión de trabajo antes de realizar alguna operación ya que en caso de que otro usuario halla ordenado planificar, los demás usuarios no pueden utilizar ninguna funcionalidad excepto la de consultar el estado de su sesión, los posibles estados son activado y desactivado.	

Tabla 3.7. Historia de usuario "Consultar el estado de la sesión de trabajo".

Historia de Usuario	
Número: 8	**Nombre:** Cerrar sesión de trabajo.
Usuario: Planificador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos Estimados: 1	**Iteración asignada:** 1
Descripción: Este proceso define como se cierra una sesión de trabajo en el módulo de planificación de actividades. Esto permite liberar los recursos utilizados por la sesión.	
Observaciones: Una restricción es una relación que afecta a uno o varios recursos.	

Tabla 3.8. Historia de usuario "Cerrar sesión de trabajo".

Historia de Usuario	
Número: 9	**Nombre:** Configurar Motor de Búsqueda.
Usuario: Administrador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Medio
Puntos Estimados: 2	**Iteración asignada:** 3
Descripción: Este proceso define la configuración en el cual se debe establecer los algoritmos de búsqueda, el dominio máximo, las estrategias de distribución.	
Observaciones: El rendimiento del motor está estrechamente vinculado con la configuración del motor.	

Tabla 3.9. Historia de usuario "Configurar Motor de Búsqueda".

3.4 Fase de Planificación. Definición.

La metodología XP plantea la planificación como un diálogo continuo entre las partes involucradas en el proyecto, incluyendo al cliente, a los programadores y a los coordinadores o gerentes. Es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación. El resultado de esta fase es un Plan de Entregas.

La planificación no debe ser estricta puesto que hay muchas variables en juego, debe ser flexible para poder adaptarse a los cambios que puedan surgir. Una buena estrategia es hacer planificaciones detalladas para unas pocas semanas y planificaciones mucho más abiertas para unos pocos meses.

3.4.1 Estimación del esfuerzo por historia de usuario.

Durante la fase de planificación se realiza una estimación del esfuerzo que costará implementar cada historia de usuario. Este se expresa utilizando como medida el punto. Un punto se considera como una semana ideal de trabajo donde los miembros de los equipos de desarrollo trabajan el tiempo planeado sin ningún tipo de interrupción. (BECK, 2000) .

Los resultados obtenidos en esta estimación se exponen en la siguiente tabla:

Historias de Usuario	Puntos estimados
Iniciar Sesión de trabajo.	1
Consultar el estado de la sesión de trabajo.	1
Insertar recurso.	1
Insertar actividad.	1
Insertar restricción.	1
Planificar.	3
Replanificar.	2
Configurar Motor de Búsqueda.	2
Cerrar sesión de trabajo.	1
Total	13

Tabla 3.10. Plan de estimación de esfuerzo por historias de usuario.

3.4.2 Plan de iteraciones.

Las HU seleccionadas para cada entrega son desarrolladas y probadas en un ciclo de iteración, de acuerdo al orden preestablecido. Al comienzo de cada ciclo, se realiza una reunión de planificación de la iteración. Cada HU se traduce en tareas específicas de programación. Así mismo, para cada HU se establecen las pruebas de aceptación.

Estas pruebas se realizan al final del ciclo en el que se desarrollan, pero también al final de cada uno de los ciclos siguientes, para verificar que subsiguientes iteraciones no han afectado a las anteriores.

Las pruebas de aceptación que hayan fallado en el ciclo anterior son analizadas para evaluar su corrección, así como para prever que no vuelvan a ocurrir. Además de las entregas habituales al final de

cada iteración, todas las semanas se hace un resumen con la presencia del cliente y se muestran los adelantos del sistema, y con el mismo objetivo se realizan reuniones diarias entre los desarrolladores al final de cada sesión de trabajo. Las iteraciones se detallan a continuación:

Iteración 1

En la primera iteración se implementan las HU con mayor prioridad, obteniendo al final de la misma una primera versión de prueba y dando al sistema las primeras funcionalidades.

Iteración 2

En la segunda iteración se realizará la implementación de las HU con prioridad de negocio alta relacionada. Además, se corregirán errores o disconformidades del usuario con las HU implementadas en la iteración anterior. De esta forma se obtiene la segunda versión de pruebas del software. Esta segunda versión será mostrada a los clientes con el único objetivo de realizar cambios en base a la aceptación del mismo.

Iteración 3

En esta iteración se termina de implementar las restantes historias de usuarios y corregir errores de iteraciones anteriores, obteniéndose la versión 1.0 final del producto. Esta versión se pondrá en funcionamiento utilizándola para la evaluación de su comportamiento y rendimiento.

3.4.3 Duración de las Iteraciones.

Para una mayor organización del trabajo como lo plantea el ciclo de vida de XP se crea un plan de duración de las iteraciones, en este caso se realizaría un solo plan ya que existe un único equipo de desarrolladores.

Este plan se realiza con el objetivo de reflejar cuales serán las historias de usuario que serán implementadas en cada una de las iteraciones, así como el tiempo destinado a cada una de ellas y el orden en que se implementarán, lo que ayuda a obtener una idea general del tiempo que durará la confección total del sistema.

Iteración	Orden de las HU a implementar	Duración total de la iteración
Iteración #1	Insertar recurso. Insertar actividad. Insertar restricción. Planificar. Replanificar.	8 semanas
Iteración #2	Iniciar Sesión de trabajo. Consultar el estado de la sesión de trabajo. Cerrar sesión de trabajo.	3 semanas
Iteración #3	Configurar Motor de Búsqueda.	2 semanas

Tabla 3.11: Listado de iteraciones.

3.4.4 Plan de entregas

En el plan de entrega que se plantea a continuación se hace una propuesta de la fecha aproximada en que se harán versiones *(releases)* al sistema al finalizar cada iteración en la fase de implementación.

Entregable	Final 1ra Iteración 2da semana de marzo	Final 2da Iteración 3ra semana de abril	Final 3ra Iteración 3da semana de mayo
Planificador	0.1	0.2	1.0

Tabla 3.12. Plan de entregas.

3.5 Conclusiones

Como parte del presente capítulo se abordó todo lo referente a las fases de exploración y planificación del proyecto, haciendo una descripción de cada uno de los artefactos generados durante el transcurso de las mismas.

CAPÍTULO 4. DISEÑO, IMPLEMENTACIÓN Y PRUEBA.

4.1 Introducción.

La metodología XP hace especial énfasis en los diseños simples y claros ya que un diseño simple se implementa más rápidamente que uno complejo. Por ello XP propone implementar el diseño más simple posible que funcione y sugiere nunca adelantar la implementación de funcionalidades que no correspondan a la iteración en la que se esté trabajando. Un desarrollo mediante programación extrema está compuesto por una serie de iteraciones cortas. Cada iteración concluye ejecutando un conjunto de pruebas de aceptación que permitan al cliente comprobar si está satisfecho con el resultado.

4.2 Diseño del Módulo.

Para el diseño del módulo de planificación se tuvieron en cuenta los principios de diseños de la Plataforma de Telecomunicación Abierta (OTP) de Ericson, una de las mayores empresas dedicadas a las telecomunicaciones a nivel mundial. El uso de esta tecnología permite crear software altamente tolerante a fallas y rápidos.

4.2.1 Principio de OTP. Árbol se supervisión.

Un concepto básico en la plataforma Erlang\OTP es el árbol de supervisión. Este es un modelo que estructura los procesos basado en la idea de **trabajadores** y **supervisores**. (Ver Figura 7)

- **Trabajadores:** Son procesos que realizan la computación.
- **Supervisores:** Son procesos que monitorizan el comportamiento de los trabajadores, un supervisor puede reiniciar un trabajador si este presenta algún problema.

- **Árbol de supervisión:** Es un árbol jerárquico de trabajadores y supervisores, permitiendo que los supervisores monitoricen a otros supervisores y a trabajadores implementando así la tolerancia a fallas.

4.2.2 Principio de OTP. Comportamientos (Behaviours).

En un árbol de supervisión muchos procesos tienen estructuras similares, siguen el mismo patrón. Por ejemplo, los supervisores son muy similares en estructuras, la única diferencia es a quien supervisan, de igual forma muchos trabajadores son servidores en una relación cliente-servidor, maquinas de estado finito, manejadores de eventos y registros de errores.

Los **Comportamientos** son la formalización de esos patrones comunes. La idea es dividir el código del proceso en una parte genérica y una parte especifica.

4.2.3 Principio de OTP. Aplicaciones.

Erlang/OTP viene con un gran número de componentes, cada uno implementa una funcionalidad específica. En la terminología de Erlang/OTP los componentes son denominados **aplicaciones**, ejemplo de aplicaciones Erlang/OTP son **Mnesia**, que presenta todo lo necesario para programar servicios de base de datos, **Debugger** que es utilizado para tracear programas escritos en Erlang. El sistema basado en Erlang/OTP más pequeño esta conformado por las aplicaciones **Kernel** y **STDLIB**.

4.2.4 Proceso Schedule_server.

El módulo tiene dos partes fundamentales, la primera es un proceso denominado Schedule_server que forma parte del árbol de supervisión de la aplicación servidora y la segunda es una aplicación escrita en Oz que realiza la planificación.

Según el usuario cree o modifique eventos, plantillas o actividades estos cambios son reflejados en sus respectivas tablas en la base de datos, cuando se ordena planificar el proceso “Schedule_server” (ver Figura 7), envía todas las actividades, mediante socket TCP al servidor de planificación, el mismo realiza la planificación y la tabla de actividades se actualiza. Los eventos y actividades tienen un campo en la base de datos donde se guardan las posibles opciones de los mismos, estas opciones son restricciones para el planificador, cuando las actividades son planificadas se le agrega una opción indicando la fecha a la cual fueron asignadas, de forma tal que para futuras planificaciones el planificador las ubique en esa

fecha y pueda concentrarse en las actividades que realmente necesitan planificarse. Al planificar es necesario pasar todas las actividades, incluso aquellas que ya están planificadas ya que el planificador requiere tener la información completa para no ubicar dos actividades en un mismo local o ubicar una persona en dos lugares distintos al mismo tiempo. Además de planificar, el sistema permite replanificar lo cual significa que las actividades que ya estaban ubicadas van a ser tratadas cono si no lo estuvieran, adquiriendo así, en la mayoría de los casos, una nueva ubicación.

4.2.5 Planificador

El planificador es una aplicación servidora escrita en el lenguaje Oz, el mismo espera por un puerto una cadena de caracteres que contiene la información y comandos, ejemplo de esta cadena es “e:1:45|e:2:45|r:1:1:2|c:2:1:2” la cual significa que hay dos actividades que duran 45 minutos las dos, usan un mismo recurso, y la primera actividad debe ir primero que la segunda actividad. El Oz soporta al paradigma con restricciones, permitiendo escribir un script que tiene una función por cada opción que tenga una actividad, por ejemplo la opción de que una determinada actividad debe planificarse primero que otra, se cumple gracias a una función que tiene el script, esta función es tratada como una restricción no básica. Además se utiliza la estrategia de distribución “primero en fallar” (aunque pueden utilizarse otras) la cual lleva a una solución para 172 actividades en 31 ms (ver Figura 8). Cuando la planificación esta completa se codifica en una cadena de caracteres y es enviada de vuelta para el proceso Schedule_server del servidor de Erlang donde es procesada y se actualiza la tabla de actividades (ver Figura 9).

Las restricciones que utiliza el script para planificar están todas almacenadas en un módulo denominado “Restricciones”, por lo que es posible aumentar las restricciones o modificarlos simplemente cambiando dicho fichero, incluso es posible hacerlo en tiempo de ejecución sin tener que detener el programa. Esto es posible ya que Mozart/Oz permite el enlace de módulos en caliente.

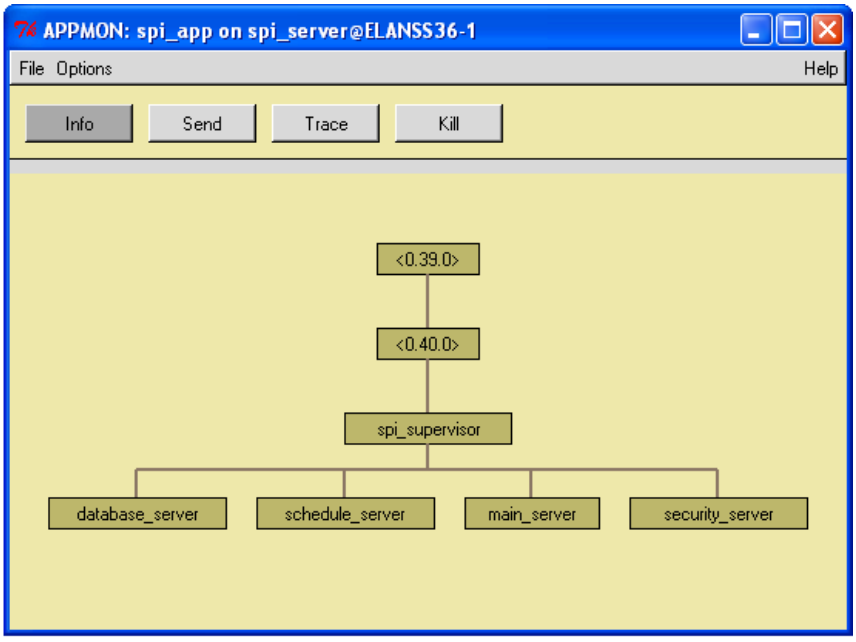

Figura 7. Jerarquía de procesos en el servidor

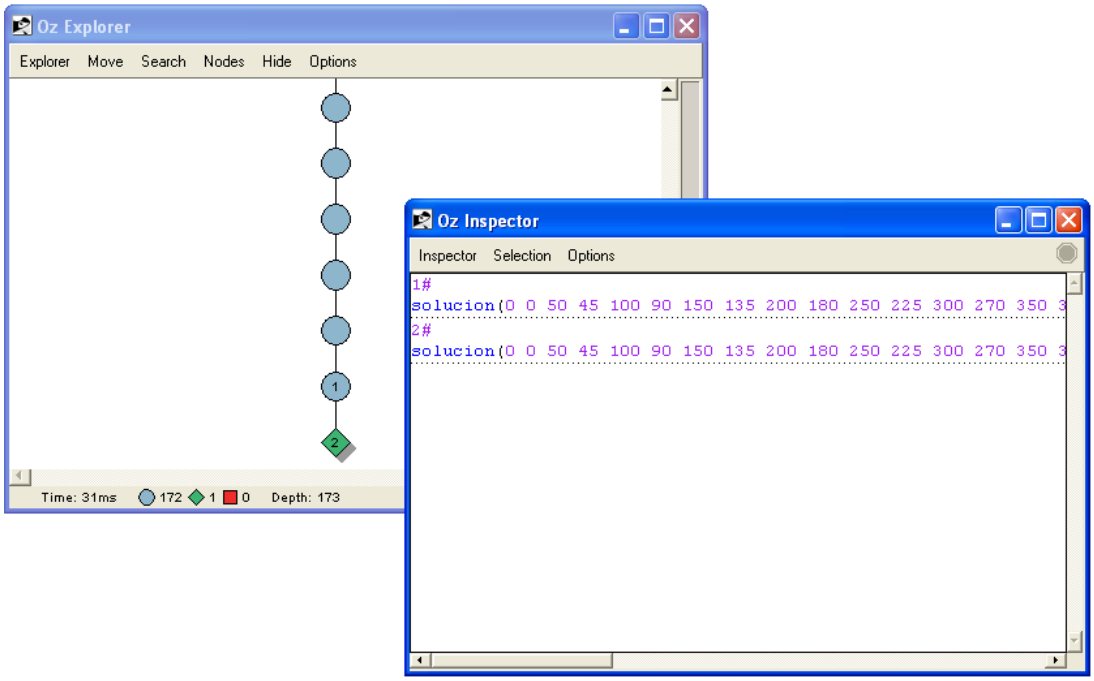

Figura 8. Árbol de solución de la planificación de 172 actividades.

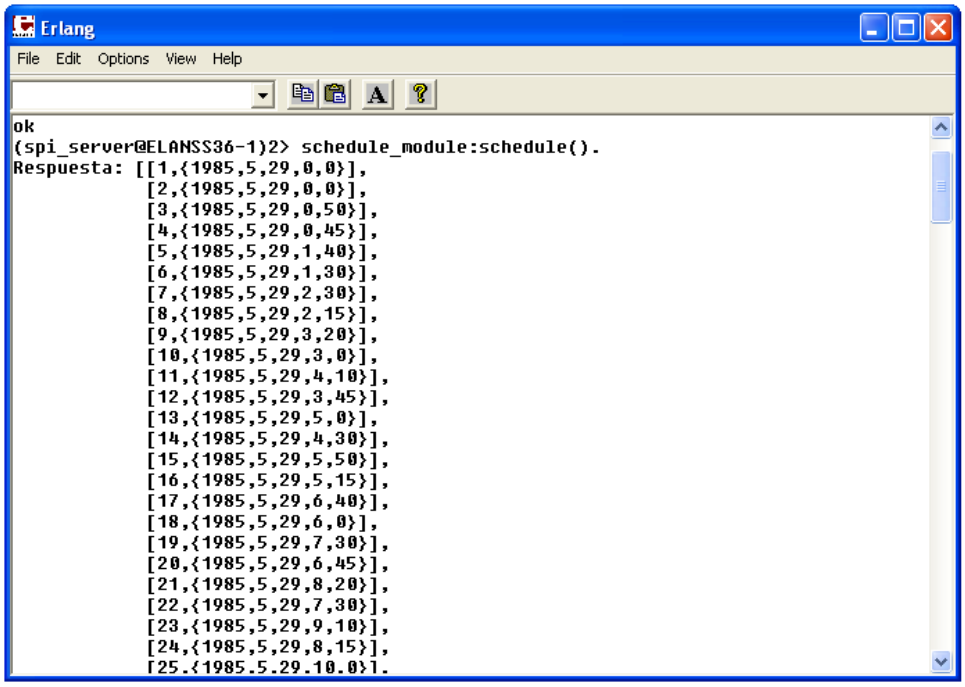

Figura 9. Resultado de la planificación de 172 actividades.

4.3 Desarrollo del Sistema

El desarrollo es la parte más importante en el proceso de la programación extrema. Todos los trabajos tienen como objetivo que se programen lo más rápidamente posible, sin interrupciones y en dirección correcta. También es muy importante el diseño, y se establecen los mecanismos, para que éste sea revisado y mejorado de manera continuada a lo largo del proyecto, según se van añadiendo funcionalidades al mismo. (BECK, 2000).

La metodología XP plantea una serie de prácticas básicas para una producción efectiva, estas deben seguirse al pie de la letra para que el proyecto tenga un desarrollo exitoso.

Estas se describen a continuación:

- **La planeación:** En la cual la opinión del cliente y del equipo de desarrollo deben fusionarse como un todo coherente. El cliente y los programadores negocian el alcance del proyecto para cada iteración, esta etapa es conocida como “Juego de Planificación”. El factor crítico es permitir al

cliente tomar las decisiones de negocio y al equipo de desarrollo tomar las decisiones técnicas. *Ver (Ilustración1) de los Anexos.*

- **Entregas en iteraciones pequeñas:** Las mini-versiones deben ser lo suficientemente pequeñas como para poder hacer una cada pocas semanas. Deben ser versiones que ofrezcan algo útil al usuario final y no trozos de código que no pueda ver funcionando.

- **Hacer historias y usar metáforas:** Guiar todo el desarrollo del sistema a través de una Historia Compartida por el Equipo (o Metáfora) acerca de cómo trabaja (o debería trabajar) el Sistema.

- **Diseñar simple:** El sistema debería diseñarse de la manera más simple posible en cualquier momento dado. La complejidad extra es removida, tan pronto como es descubierta.

- **Probar:** Los desarrolladores continuamente escriben pruebas unitarias, las cuales deben correr sin error para que el desarrollo pueda continuar. Cuando se detecta un error en una corrida, su reparación pasa a ser la máxima prioridad para el Programador y/o el Equipo. Los Clientes (ayudados por desarrolladores) participen en el diseño de pruebas funcionales para probar qué funcionalidades están terminadas de acuerdo a sus expectativas.

- **Refactorizar:** Los desarrolladores reestructuran el sistema sin cambiar su comportamiento para remover duplicación de código, mejorar la comunicación, simplificar el código, o agregar flexibilidad.

- **Programar por pares:** Todo el código desarrollado es escrito por dos desarrolladores sentados frente a una única estación de trabajo. El que utiliza el teclado piensa en la mejor manera de implementar alguna funcionalidad, el otro piensa estratégicamente, cuestionando si se puede simplificar, anticipando pruebas, o preguntándose si el enfoque es el adecuado o si debe descartarse código y replantear el problema. Se alienta la rotación continua de programadores en los pares, combinando diferentes niveles de experiencia y de conocimiento del código.

- **Propiedad colectiva:** Cualquier integrante del Equipo puede cambiar cualquier código de cualquier parte del sistema en cualquier momento. Se asume que al seguir las prácticas de XP, después de un tiempo razonable, cualquier programador conoce todo el código, principalmente por el hecho de la programación en pares.

- **Integrar continuamente:** El sistema se integra y se construye (por ejemplo, se compila), es decir, se unen sus partes, varias veces por día, hasta el extremo de integrar el sistema completo, cada vez que se termina una tarea.
- **Semanas de 40 horas:** Trabajar no más de cuarenta horas por semana como una regla estándar. Nunca trabajar sobre-tiempo dos semanas seguidas; si esto es necesario, hay problemas más grandes que hay que descubrir.
- **Cliente disponible localmente:** Es condición esencial la inclusión de al menos un Cliente real, vivo, como parte del Equipo. Debe estar disponible en turnos completos para responder preguntas e interactuar con el resto del Equipo.
- **Usar estándares de codificación:** Los desarrolladores escribirán todo el código de acuerdo a reglas predeterminadas que enfatizarán la comunicación a través del código. Estos estándares serán simples de seguir y se seguirán siempre.

La forma iterativa para la implementación de un software que plantea XP junto a estas prácticas da como resultado que al culminar cada iteración se obtenga una versión del producto funcional, éste debe ser probado y mostrado al cliente sirviendo de retroalimentación para el equipo de trabajo, se exponen detalladamente las tres iteraciones generadas por la planificación descrita anteriormente así como las tareas que se plantearon para la realización de cada una de las historias de usuarios y las pruebas que se efectuaron al sistema.

Iteración 1

En esta primera iteración se desarrollan las historias de usuarios de mayor prioridad en el sistema con el objetivo de obtener una primera versión del producto con las principales características o funcionalidades para ser mostrado al cliente.

No.	Historia de Usuario	Estimación	Real
1	Insertar recurso.	1	1
2	Insertar actividad.	1	1
3	Insertar restricción.	1	1
4	Planificar.	3	3
5	Replanificar.	2	2
Total		**8**	**8**

Tabla 4.1. Historias abordadas en la primera iteración.

Tareas generadas por cada historia de usuario.

Tarea	
Número de tarea: 1	**Número de historia:** 1
Nombre de la tarea: insertar recursos a utilizar	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 2009-02-17	**Fecha fin:** 2009-02-23
Programador responsable: José Enrique Benítez	
Descripción: Este proceso define la inserción de recursos al sistema, mediante una sesión de trabajo activa, el cual permite al motor estar previsto de las condiciones iniciales.	

Tabla 4.2. Tarea #1 HU "Insertar Recurso".

Tarea	
Número de tarea: 1	**Número de historia:** 2
Nombre de la tarea: insertar actividades a planificar	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 2009-02-24	**Fecha fin:** 2009-03-1
Programador responsable: José Enrique Benítez	
Descripción: Este proceso define la inserción de actividades al sistema, mediante una sesión de trabajo activa, el cual permite al motor estar previsto de las condiciones iniciales.	

Tabla 4.3. Tarea #2 HU "Insertar Actividad".

Tarea	
Número de tarea: 1	**Número de historia:** 3
Nombre de la tarea: insertar restricciones de actividades	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 2009-02-17	**Fecha fin:** 2009-02-23
Programador responsable: René Villalta Soto	
Descripción: Este proceso define la inserción de restricciones al sistema, mediante una sesión de trabajo activa, el cual permite al motor estar previsto de las condiciones iniciales.	

Tabla 4.4. Tarea #3 HU "Insertar Restricción".

Tarea	
Número de tarea: 1	**Número de historia:** 4
Nombre de la tarea: planificación de actividades en el tiempo	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha inicio: 2009-03-3	**Fecha fin:** 2009-03-25
Programador responsable: René Villalta Soto	
Descripción: Previamente se debe haber provisto al módulo con los datos necesarios para realizar una planificación, estos datos son actividades, recursos y restricciones, se activa el algoritmo de planificación.	

Tabla 4.5. Tarea #4 HU "Planificar".

Tarea	
Número de tarea: 1	**Número de historia:** 5
Nombre de la tarea: replanificar actividades en el tiempo	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha inicio: 2009-03-26	**Fecha fin:** 2009-04-3
Programador responsable: José Enrique Benítez	
Descripción: Previamente se debe haber provisto al módulo con los datos necesarios para realizar una re planificación, estos datos son actividades nuevas de un espacio de trabajo en conjunto con las restantes que pertenecen al mismo, pero de estas ultimas solo la restricción de mantenerse fija, se activa el algoritmo de planificación.	

Tabla 4.6. Tarea #5 HU "Replanificar".

Iteración 2

En el transcurso de esta iteración se implementan las historias de usuarios con nivel de prioridad alto para el sistema .Esto permite agregar nuevas funcionalidades al sistema por lo que se obtiene una versión más completa y el cliente podrá observar algunas de las facilidades en cuanto a las funcionalidades del sistema en su culminación.

No.	Historia de Usuario	Estimación	Real
1	Iniciar sesión de trabajo.	1	1
2	Consultar el estado de la sesión de trabajo.	1	1
3	Cerrar sesión de trabajo.	1	1
Total		**3**	**3**

Tabla 4.7. Historias abordadas en la segunda iteración.

Tareas generadas por cada historia de usuario

Tarea	
Número de tarea: 1	**Número de historia:** 1
Nombre de la tarea: Iniciar una sesión de trabajo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 2009-04-3	**Fecha fin:** 2009-04-10
Programador responsable: René Villalta Soto.	
Descripción: Este proceso define como se inicia una sesión de trabajo en el módulo de planificación de actividades. La sesión permite al usuario acceder a las otras funcionalidades que brinda el módulo.	

Tabla 4.8. Tarea #1 HU "Iniciar sesión de trabajo".

Tarea	
Número de tarea: 1	**Número de historia:** 2
Nombre de la tarea: Consultar si la sesión de trabajo está ocupada.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 2009-04-10	**Fecha fin:** 2009-04-16
Programador responsable: José Enrique Benítez.	
Descripción: La sesión de trabajo es necesaria para diferenciar las peticiones de los diferentes usuarios, ya que cuando un usuario manda a planificar las demás secciones deben bloquearse mientras dure el proceso. Es necesario consultar el estado de la sesión de trabajo antes de realizar alguna operación ya que en caso de que otro usuario halla ordenado planificar, los demás usuarios no pueden utilizar ninguna funcionalidad excepto la de consultar el estado de su sesión, los posibles estados son activado y desactivado.	

Tabla 4.9. Tarea #2 HU "Consultar el estado de la sesión de trabajo".

Tarea	
Número de tarea: 1	**Número de historia:** 3
Nombre de la tarea: Cerrar una sesión de trabajo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 2009-04-16	**Fecha fin:** 2009-04-20
Programador responsable: José Enrique Benítez.	
Descripción: Este proceso define como se cierra una sesión de trabajo en el módulo de planificación de actividades. Esto permite liberar los recursos utilizados por la sesión.	

Tabla 4.10. Tarea #3 HU "Cerrar sesión de trabajo".

Iteración 3

Al finalizar esta iteración se habrán implementado todos los elementos y funcionalidades del sistema brindando una versión final del producto listo para ponerlo en funcionamiento.

No.	Historia de Usuario	Estimación	Real
1	Configurar Motor de Búsqueda.	2	2
Total		**2**	**2**

Tabla 4.11. Historias abordadas en la tercera iteración.

Tareas generadas por cada historia de usuario

Tarea	
Número de tarea: 1	**Número de historia:** 1
Nombre de la tarea: Establecer el dominio máximo de tiempo.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 2009-04-20	**Fecha fin:** 2009-04-25
Programador responsable: René Villalta Soto.	
Descripción: Establece el dominio máximo en el cual se efectuara la planificación.	

Tabla 4.12. Tarea #1 HU "Configurar motor de búsqueda".

Tarea	
Número de tarea: 2	**Número de historia:** 1
Nombre de la tarea: Establecer estrategia de distribución.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Fecha inicio: 2009-04-26	**Fecha fin:** 2009-04-30
Programador responsable: René Villalta Soto.	
Descripción: Establece que estrategia de distribución debe seguir el motor para dar un tipo de solución de acuerdo a las necesidades del planificador.	

Tabla 4.13. Tarea #2 HU "Configurar motor de búsqueda".

4.4 Pruebas.

La metodología XP pone la comprobación como el fundamento del desarrollo, con cada programador escribiendo pruebas cuando escriben su código de producción. Las pruebas se integran en el proceso de integración continua y construcción lo que rinde una plataforma altamente estable para el desarrollo futuro. La programación extrema define dos tipos de pruebas las cuales son:

- Pruebas de aceptación.
- Pruebas unitarias

4.4.1 Pruebas unitarias

La producción de código está dirigida por las pruebas unitarias. Las pruebas unitarias son establecidas antes de escribir el código y son ejecutadas constantemente ante cada modificación del sistema. Los clientes escriben las pruebas funcionales para cada historia de usuario que deba validarse. En este contexto de desarrollo evolutivo y de énfasis en pruebas constantes, la automatización para apoyar esta actividad es crucial. (BECK, 2000).

4.4.2 Pruebas de aceptación

Las pruebas de aceptación permiten confirmar que la HU ha sido implementada correctamente al final de cada iteración. Este período de prueba se conoce también como período de caja negra donde se definirán las entradas al sistema y los resultados esperados de estas entradas. Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de éstas es garantizar que los requerimientos han sido cumplidos y que el sistema es aceptable.

La prueba de aceptación es realizada por un grupo de usuarios finales o los clientes del sistema, para asegurarse que el sistema desarrollado cumple sus requisitos. A continuación se representan las pruebas de aceptación realizadas para las HU del sistema:

Caso de Prueba de Aceptación	
Código: HU1_P1	**Historia de Usuario:** Insertar recurso.
Nombre: Insertar recurso.	
Descripción: Prueba de funcionalidad de la inserción de recursos.	
Condiciones de ejecución: Debe estar activado el motor de planificación.	
Entrada / Pasos de Ejecución: Se pasan los recursos de forma automática al motor en forma de paquetes cuando el planificador decida planificar.	
Resultado esperado: Los recursos son asignados correctamente.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.14. Caso de prueba de aceptación HU1_P1 “Insertar Recurso”.

Caso de Prueba de Aceptación	
Código: HU2_P1	**Historia de Usuario:** Insertar actividad.
Nombre: Insertar actividad.	
Descripción: Prueba de funcionalidad de la inserción de actividades.	
Condiciones de ejecución: Debe estar activado el motor de planificación.	
Entrada / Pasos de Ejecución: Se pasan las actividades de forma automática al motor en forma de paquetes cuando el planificador decida planificar.	
Resultado esperado: Las actividades son asignadas correctamente.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.15. Caso de prueba de aceptación HU2_P1 "Insertar Actividad".

Caso de Prueba de Aceptación	
Código: HU2_P1	**Historia de Usuario:** Insertar actividad.
Nombre: Insertar actividad.	
Descripción: Prueba de funcionalidad de la inserción de actividades.	
Condiciones de ejecución: Debe estar activado el motor de planificación.	
Entrada / Pasos de Ejecución: Se pasan las actividades de forma automática al motor en forma de paquetes cuando el planificador decida planificar.	
Resultado esperado: Las actividades son asignadas correctamente.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.16. Caso de prueba de aceptación HU3_P1 "Insertar Restricción".

Caso de Prueba de Aceptación	
Código: HU4_P1	**Historia de Usuario:** Planificar.
Nombre: Planificación de actividades.	
Descripción: Previamente se debe haber provisto al módulo con los datos necesarios para realizar una planificación, estos datos son actividades, recursos y restricciones, se activa el algoritmo de planificación.	
Condiciones de ejecución: Debe estar activado el motor de planificación, las actividades a planificar en conjunto con los recursos asociados a ellas y las restricciones deben haber sido insertadas previamente.	
Entrada / Pasos de Ejecución: Se le envía un mensaje al motor que ya puede planificar.	
Resultado esperado: Se obtiene una planificación en tiempo óptimo de acuerdo a las restricciones establecidas.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.17. Caso de prueba de aceptación HU4_P1 "Planificación de actividades".

Caso de Prueba de Aceptación	
Código: HU5_P1	**Historia de Usuario:** Replanificar.
Nombre: Re planificación de actividades.	
Descripción: Previamente se debe haber provisto al módulo con los datos necesarios para realizar una re planificación, estos datos son actividades, recursos y restricciones, se activa el algoritmo de planificación.	
Condiciones de ejecución: debe estar activado el motor de planificación, las actividades a planificar en conjunto con los recursos asociados a ellas y las restricciones deben haber sido insertadas previamente.	
Entrada / Pasos de Ejecución: se le envía un mensaje al motor que ya puede replanificar.	
Resultado esperado: las restricciones son asignadas correctamente.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.18. Caso de prueba de aceptación HU5_P1 "Replanificación de actividades".

Caso de Prueba de Aceptación	
Código: HU6_P1	**Historia de Usuario:** Iniciar sesión de trabajo.
Nombre: Iniciar sesión de trabajo.	
Descripción: Este proceso define como se inicia una sesión de trabajo en el módulo de planificación de actividades. La sesión permite al usuario acceder a las otras funcionalidades que brinda el módulo.	
Condiciones de ejecución: Debe haberse autenticado al Sistema y debe poseer permisos que le acrediten una sesión de trabajo.	
Entrada / Pasos de Ejecución: Al autentificarse un usuario la sesión de trabajo se crea de forma automática.	
Resultado esperado: Se le asigna una sesión de trabajo con permiso a funcionalidades según el rol.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.19. Caso de prueba de aceptación HU6_P1 "Iniciar sesión de trabajo".

Caso de Prueba de Aceptación	
Código: HU7_P1	**Historia de Usuario:** Consultar el estado de la sesión de trabajo.
Nombre: Consultar estado de la sesión de trabajo.	
Descripción: La sesión de trabajo es necesaria para diferenciar las peticiones de los diferentes usuarios, ya que cuando un usuario manda a planificar las demás secciones deben bloquearse mientras dure el proceso. Es necesario consultar el estado de la sesión de trabajo antes de realizar alguna operación ya que en caso de que otro usuario halla ordenado planificar, los demás usuarios no pueden utilizar ninguna funcionalidad excepto la de consultar el estado de su sesión, los posibles estados son activado y desactivado.	
Condiciones de ejecución: Debe haberse autenticado al Sistema y debe poseer permisos que le acrediten una sesión de trabajo.	
Entrada / Pasos de Ejecución: Se pide un reporte al sistema del estado de una sesión.	
Resultado esperado: Se comprueba si una sesión está planificando en este momento.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.20. Caso de prueba de aceptación HU7_P1 "Consultar estado de la sesión de trabajo".

Caso de Prueba de Aceptación	
Código: HU8_P1	**Historia de Usuario:** Cerrar sesión de trabajo.
Nombre: Cerrar sesión de trabajo.	
Descripción: Este proceso define como se cierra una sesión de trabajo en el módulo de planificación de actividades. Esto permite liberar los recursos utilizados por la sesión.	
Condiciones de ejecución: Debe estar iniciada la sesión de trabajo.	
Entrada / Pasos de Ejecución: Se le envía un mensaje al planificador que cierre la sesión de trabajo.	
Resultado esperado: Se cierra la sesión y se liberan los recursos usados por la misma.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.21. Caso de prueba de aceptación HU8_P1 "Cerrar sesión de trabajo".

Caso de Prueba de Aceptación	
Código: HU9_P1	**Historia de Usuario:** Configurar Motor de Búsqueda.
Nombre: Establecer el dominio máximo de tiempo.	
Descripción: Establece el dominio máximo en el cual se efectuara la planificación.	
Condiciones de ejecución: Debe estar iniciada la sesión de trabajo.	
Entrada / Pasos de Ejecución: Se le envía un mensaje al planificador con el intervalo de tiempo permitido para su planificación.	
Resultado esperado: La ultima actividad planificada, se encuentra antes o en el punto del ultimo valor del intervalo.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.22. Caso de prueba de aceptación HU9_P1 "Establecer dominio máximo de tiempo".

Caso de Prueba de Aceptación	
Código: HU9_P2	**Historia de Usuario:** Configurar Motor de Búsqueda.
Nombre: Establecer estrategia de distribución.	
Descripción: Establece que estrategia de distribución debe seguir el motor para dar un tipo de solución de acuerdo a las necesidades del planificador.	
Condiciones de ejecución: Debe estar iniciada la sesión de trabajo.	
Entrada / Pasos de Ejecución: Se le envía un mensaje al planificador con la estrategia a seguir para su planificación.	
Resultado esperado: Se obtienen resultados diferentes según la estrategia utilizada.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 4.23. Caso de prueba de aceptación HU9_P2 "Establecer estrategia de distribución".

4.5 Conclusiones

En el presente capítulo se hizo referencia a las etapas de desarrollo, implementación y pruebas del software en elaboración, presentando las principales funcionalidades del sistema junto a sus descripciones. Se hizo énfasis en la simplicidad y la reutilización de código, un factor saludable en la práctica de la programación, evitando exceso de código y una lógica deficiente, un diseño innecesariamente complejo, problemas para el mantenimiento del sistema, dificultad para la ampliación del mismo, y defectos que degraden la calidad del producto.

CONCLUSIONES

En el documento se ha planteado una panorámica de los sistemas y algoritmos que constituyen actualmente la tendencia en el tema de la planificación de actividades, muchos de ellos presentan limitaciones, por ejemplo, el número de actividades que son capaces de planificar. Se explicó el paradigma de programación con restricciones, el cual permite resolver problemas de satisfacción de restricciones (PSR), dentro de los cuales se encuentra el problema de la planificación de actividades.

En el trabajo se presenta las principales características de los lenguajes propuestos para la realización del módulo de planificación, además se muestra cómo el lenguaje Oz implementa el paradigma de programación funcional y con restricciones, estas son las bases teóricas necesarias en la solución propuesta para el Sistema de Planificación Inteligente.

Se cumplieron los objetivos planteados satisfactoriamente, arrojando a su vez una serie de resultados:

- Se definió el paradigma o enfoque a utilizar para dar solución a los problemas de planificación.
- Se modeló el problema de planificación mediante el paradigma seleccionado.
- Se implementó el módulo con una tecnología que soporte el paradigma seleccionado.

De esta manera se concluye que se cumplió el objetivo de la investigación, pues se realizó la implementación del módulo de planificación para su vinculación con el Sistema de Planificación Inteligente.

RECOMENDACIONES

- Extender las funcionalidades del módulo de planificación mediante la incorporación de nuevas funciones que implementen nuevas restricciones.
- Aplicar algún estándar que sirva como intermediario entre el módulo de planificación y cualquier aplicación que requiera su uso.
- Implementar la capacidad del motor de ejecutarse en varias computadoras y estudiar nuevas estrategias de distribución que disminuyan el tiempo de respuesta del módulo.

BIBLIOGRAFÍA

[0]. Aplicaciones de la programación con restricciones. Disponible en < http://www.constraintsolving.com/>

[1]. ASC Horarios. Disponible en < http://www.abcdatos.com/programas/programa/l8252.html >.

[2]. B-Prolog. Disponible en <http://www.probp.com>. [Fecha de consulta 2 febrero 2009].

[3]. Casos reales de Sistemas de optimización.

Disponible en < http://www.phpsimplex.com/casos_reales.htm > [Fecha de consulta 22 enero 2009].

[4]. Ciao Prolog. Disponible en <http://www.clip.dia.fi.upm.es/Software/Ciao/>.

[5]. Cronos

Disponible en: < http://cronos.sisvenca.com/Cronos.html> [Fecha de consulta 25 enero 2009].

[6]. CHIP V5.

Disponible en <http://www.cosytec.com/production_scheduling/chip/optimization_product_chip.htm>.

[7]. Christian Schulte, Gert Smolka. Mozart Documentation.2009.

Disponible en <http://www.mozart-oz.org/>

[8]. ECLiPSe. Disponible en <http://eclipse.crosscoreop.com/>.

[9]. Edward Tsang. Foundations of Constraint Satisfaction. London, Academic Press, 1993, 441.

[10]. Francesca Rossi, Peter Van Beek, Toby Walsh. Handbook of Constraint Programming. Amsterdam, The Netherlands, Elsevier, 2006, 1000.

[11]. Generador de Horarios para Centros de Enseñanza.

Disponible En: <http://www.penalara.com/> [Fecha de consulta 31 enero 2007].

[12]. Generador de Horarios para Centros Docentes 7.0.

Disponible en: <http://generador-de-horarios-para-centros-docentes.softonic.com/ie/22084>

[Fecha de consulta 31 enero 2009].

[13]. Gestión de Horarios.

Disponible en < http://www.algoritmica.com.mx/Docs/folletohorariov1.pdf >.

[14]. GNU Prolog. Disponible en <http://www.gprolog.org/>.

[15]. IVAR JACOBSON. El Proceso Unificado de Desarrollo de Software, Addison-Wesley, 1999.

[16]. Krzysztof R. Apt. Principles of Constraint Programming. Amsterdam, The Netherlands, Cambridge University Press, 2003, 420.

[17]. Mozart. Disponible en <http://www.mozart-oz.org/>.

[18]. Prueba realizada al Erlang. Disponible en

<http://groups.google.com/group/comp.lang.functional/msg/33b7a62afb727a4f?dmode=source>.

[19]. Re fusiones para el Generador de Horarios y Generador de Grupos Milenio-6.

Disponible en: < http://www.adossis.es/REFUGHM6.htm>

[Fecha de consulta 30 enero 2009].

[20]. ROMAN BARTÁK. Online Guide to Constraint Programming.

Disponible en < http://ktiml.mff.cuni.cz/~bartak/constraints/>

[Fecha de consulta 25 enero 2009].

[21]. SICStus. Disponible en <http://www.sics.se/sicstus/>.

[Fecha de consulta 24 enero 2009].

[22]. Sistema Generador de Horarios. Disponible En: http://lcc.ens.uabc.mx/sh2007_1/index.jsp

[Fecha de consulta 30 enero 2007].

[23]. SWI Prolog. Disponible en <http://www.swi-prolog.org/>. [Fecha de consulta 20 enero 2009].

[24]. Timetab. Disponible en <http://www.abcdatos.com/programas/programa/l1382.html>.

[Fecha de consulta 15 febrero 2009].

[25]. TimeTabler Lantiv. Disponible en <http://www.lantiv.com >.

[Fecha de consulta 20 febrero 2009].

[26]. TimeTabler Mimosa Software. Disponible en <http://www.mimosasoftware.com >.

[Fecha de consulta 15 febrero 2009].

[27]. TimerTabler Event Management System. Disponible en <http://www.dea.com >.

[Fecha de consulta 15 febrero 2009].

GLOSARIO DE TÉRMINOS

Administrador de Sistema: Persona encargada de instalar el sistema.

Administrador: Persona encargada de la configuración del sistema.

Almacén de Restricciones: Es la entidad que almacena las restricciones básicas en forma de conjunción, donde posteriormente los propagadores realizan la propagación de la restricción.

Distribuidores: Subprocesos que se encargan de aplicar las estrategias de distribución.

Dominio Finito: Intervalo de valores numéricos enteros no negativos.

Erlang: Lenguaje de programación funcional de alto nivel.

Espacio de Trabajo: Entidad que permite agrupar varias entidades a las que va a tener acceso el planificador.

Estrategia de Distribución: Criterio de distribución tomado al seleccionar las variables.

Evento: Es la entidad que vincula los **recursos** con actividades en un tiempo determinado.

JInterface: Librería usada para la conexión C#-Erlang

Mnesia: Gestor de Bases de Datos de Erlang.

Mono: Implementación del framework.Net para Linux.

Mozart/Oz: Implementación del lenguaje Oz

OTP: (Plataforma Abierta de Telecomunicaciones) Modelo de diseño para aplicaciones concurrentes en Erlang.

Oz: Lenguaje de programación multiparadigma.

Planificador: Persona encargada de crear eventos y planificar.

Propagación de Restricciones: Mecanismo de inferencia, el cual permite reducir el dominio de las variables según una restricción.

Propagadores: Entidades que hacen cumplir las restricciones en **Oz.**

Recurso: Se denomina recursos a las **personas** y **locales**.

Restricción: relación entre variables que debe cumplirse a la hora de planificar.