Aplicación de la ingeniería del software sobre la herramienta MATE: *Application Controller*

Memoria del proyecto
de Ingeniería Técnica en
Informática de Sistemas
realizado por
Antonio Pimenta Soto
y dirigido por
Anna Sikora

Escola d'Enginyeria
Sabadell, Septiembre de 2011

Agradecimientos

Primero de todo agradecer a mis dos compañeros Noel y Rodrigo, con los cuales no solo he realizado este proyecto sino que he compartido estos últimos años de estudiante.

Gracias a Anna y Eduardo por contar con nosotros para participar de un proyecto tan interesante como MATE.

Y también a Joan Piedrafita que ha estado con nosotros a lo largo de todo el proceso, ayudándonos con nuestras dudas y ofreciéndonos un punto de vista diferente.

FULL DE RESUM
PROJECTE FI DE CARRERA DE L'ESCOLA D'ENGINYERIA

Títol del projecte: Aplicación de la ingenieria del software sobre la herramienta MATE: Application Controller	
Autor[a]: Antonio Pimenta Soto	**Data:** Septembre de 2011
Tutor[a]/s[es]: Anna Sikora	
Titulació: Enginyeria Técnica Informatica de sistemes	
Paraules clau (mínim 3) • Català: Enginyeria del software, programació paral·lela i qualitat de software. • Castellà: Ingenieria del software, programación paralela y calidad del software. • Anglès: Software engineering, parallel programming and software quality.	

Resum del projecte (extensió màxima 100 paraules)

- Català:

Aquest projecte té com a objectiu crear i aplicar una metodologia a una aplicació anomenada MATE, que va ser creada al any 2003 per Anna Sikora per a la seva tesis doctoral. Es tracta de dotar el projecte MATE de les eines necessàries per garantir la seva evolució. La metodologia creada consta de l'especificació d'un entorn de treball y una serie de documents que detallen els processos relatius al desenvolupament de MATE. A més s'han creat algunes noves característiques que fan de MATE una eina més completa i còmoda.

- Castellà:

Este proyecto tiene como objetivo crear y aplicar una metodología a una aplicación llamada MATE que fue creada en en el año 2003 para su tesis doctoral. Se trata de dotar el proyecto MATE de las herramientas necesarias para garantizar su evolución. La metodología creada consta de la especificación de un entorno de trabajo y una serie de documentos que detallan los procesos relativos al desarrollo de MATE. Además se han creado algunas nuevas características que hacen de MATE una herramienta mas completa y cómoda.

- Anglès:

This project has the goal of creating and applying a methodology to an application called MATE, which was created by Anna Sikora in the year 2003 for her Thesis. The main task consists in providing MATE with the necessary tools to be able to evolve. The methodology we've created specifies an work environment and a series of documents detailing the processes used in the developments of MATE. Furthermore we've added new characteristics to MATE such as a shut down mechanism and a configuration system that make MATE more complete and enhance its usability.

Índice

1 Introducción

En esta sección se exponen los aspectos referentes al proyecto proporcionando una visión general de lo que se trata en las secciones posteriores y ofreciendo al lector información útil para comprender el marco en el que se desarrolla el proyecto.

1.1 Perspectiva general

1.1.1 Computación de altas prestaciones

En la actualidad el uso de computadoras se ha extendido a todas las ramas de la ciencia, proporcionando una capacidad de calculo hasta ahora inimaginable permiten realizar cálculos y simulaciones extremadamente complejas que seria imposible llevar a cabo de cualquier otra forma.
Esté afán de conseguir máquinas mas y mas potentes cada día motiva la investigación en HPC (High Performance Computing), en castellano computación de altas pretaciones que reúne diferentes aspectos del hardware y software.

1.1.2 Computación paralela

La evolución de las computadoras hace que cada vez sean mas potentes y por tanto capaces de realizar cálculos mas complejos en menos tiempo. Esta evolución ha consistido durante mucho tiempo en aumentar la frecuencia a la que se realizan estos cálculos, pero en los últimos tiempos han surgido otras ideas como el computo paralelo que presenta una alternativa viable para dar continuidad a esta evolución. Debido a que existe una obvia limitación física a la hora de aumentar la frecuencia de los procesadores y el alto consumo que esto supone la computación paralela ha ganado protagonismo en el campo de la supercomputación.
Podemos discernir dos claras variantes de la computación paralela, sistemas multinúcleo en los que una sola maquina posee varios núcleos de procesamiento y *clusters* de maquinas donde varias maquinas colaboran para realizar tareas de forma mas rápida. Pero a pesar de las posibilidades que este tipo de arquitecturas presenta también existen desventajas. Principalmente a la hora de desarrollar software para este tipo de arquitecturas se exige un alto conocimiento debido a que la complejidad del código aumenta y aparecen muchos problemas nuevos derivados del tipo de ejecución concurrente. Problemas como la sincronización o la comunicación entre cada uno de los nodos son comunes obstáculos que aparecen y dificultan la creación de aplicaciones paralelas eficientes.
Además la programación paralela solo puede reducir el tiempo de ejecución hasta cierto punto ya que parte de las tareas que se han de ejecutar no pueden hacerlo de forma paralela. Esta limitación viene descrita por la ley de Amdahl y nos permite calcular la máxima optimización que podemos obtener en función del porcentaje de la tarea que sea paralelizable.

Cuando programamos una aplicación para que esta se ejecute de forma paralela debemos dividirla en segmentos que serán procesados por las diferentes máquinas. En este aspecto podemos dividir el paralelismo en dos tipos:

- **Paralelismo de datos:**
  En este tipo de paralelismo se divide la estructura de datos con la que se trabaja en diferentes segmentos y se aplica la misma tarea a cada una de ellos. Este tipo de paralelismo es aplicable a programas en forma de bucle donde en cada iteración se trabaja con una parte de la estructura de datos.
- **Paralelismo de tareas:**
  En cambio en este tipo de paralelismo se ejecuta una tarea distinta en cada uno de los nodos del *cluster*.

1.1.3 Aplicaciones paralelas

En las arquitecturas multinúcleo todos los núcleos de procesamiento trabajan sobre la misma memoria y por tanto no requieren de un sistema de paso de mensaje especializado para comunicarse entre ellos.

En cambio en las arquitecturas distribuidas como *clusters* o *grids* es necesario una comunicación y sincronización entre las diferentes máquinas que la ejecutan para controlar el flujo de las instrucciones que se procesan y poder aprovechar los resultados obtenidos. Este proceso de paso de mensajes es necesario para distribuir las tareas a cada uno de los procesadores y posteriormente para obtener los resultados.
En el caso de MATE las aplicaciones con las que este trabaja usarán MPI (Message Passing Interface) para realizar estas comunicaciones. Este protocolo que se usa como estándar en aplicaciones paralelas, dota a las aplicaciones de herramientas para comunicarse una vez han sido ejecutadas en una computadora paralela distribuida.

1.1.4 Optimización de aplicaciones paralelas

Como se expone anteriormente para que cálculos complejos sean posibles se ha de disponer de computadoras muy potentes, pero estas tiene un coste alto y por tanto el tiempo que se emplea en realizar las tareas es un factor que influye directamente en el coste. Esto junto con la necesidad de realizar estas tareas en tiempos razonables motiva la creación de herramientas que aceleran la ejecución de este tipo de aplicaciones.

Como consecuencia existen programas cuya finalidad es la de analizar el comportamiento de aplicaciones paralelas en ejecución para identificar problemas, como cuellos de botella, y poder solucionarlos para agilizar la progresión de la aplicación.
El proceso de optimizar las aplicaciones paralelas tiene tres fases: monitorización, análisis y

sintonización. En la fase de monitorización se extrae información referente a la ejecución de la aplicación. Esta información servirá para analizar el comportamiento de la aplicación y encontrar posibles problemas, esto se hace en la fase de análisis. Por último en la fase de sintonización se hacen cambios en la aplicación para solucionar, en la medida de lo posible, los problemas encontrados,

Análisis manual

Normalmente este proceso se realiza manualmente, el programador observa los datos obtenidos durante la monitorización del programa una vez este ha terminado su ejecución. Con los datos recolectados el usuario puede identificar problemas y modificar el código de la aplicación en consecuencia, recompilar y volver a ejecutar el programa.
El problema de esta aproximación es que se requiere un alto conocimiento sobre programación paralela para identificar y solucionar este tipo de problemas y por tanto, además de ser costoso y consumir tiempo, está al alcance de pocos expertos.

Análisis automático

Existen aplicaciones que analizan automáticamente los datos recolectados y asisten al programador en encontrar los cuellos de botella. Pero en estos casos el programador necesita realizar el mismo los cambios y el análisis sigue siendo postmortem.

Análisis dinámico

En este caso el análisis se efectúa durante la ejecución de la aplicación esto permite eliminar la necesidad de un archivo donde almacenar los datos recolectados y el análisis se va realizando en paralelo a la ejecución de la aplicación. En este caso se obtienen mejores resultados y por tanto las mejoras sugeridas serán mas efectivas pero aun se requiere parar la aplicación para poderla aplicar.

Sintonización dinámica

Este modelo permite aplicar los cambios sugeridos por el análisis de los datos recolectados en tiempo de ejecución y por tanto eliminamos la necesidad de parar la ejecución y que el programador deba modificar el código y recompilar el programa. Esto además nos permite aplicar cambios adaptativos ya que cada ciclo de monitorización recolectará datos referentes a las modificaciones del ciclo anterior. Este proceso adaptativo hace que las mejoras aplicadas sea mas eficientes, ya que en caso de aplicaciones no deterministas las soluciones aplicadas en referencia a análisis anteriores podrían ser solo efectivas en casos concretos.

Modificación dinámica

Para que la sintonización dinámica sea posible ha de existir una forma de modificar las instrucciones de la aplicación mientras esta se ejecuta. MATE utiliza una librería llamada Dyninst para eso. Dyninst proporciona una API que permite generar pequeños bloques de instrumentación llamados *snippets* y introducirlos en un punto de la aplicación en cuestión. Esto permite que un programa *mutator*, en nuestro caso MATE, modifique un *mutatee*, la aplicación sintonizada. El *mutator* utiliza una representación de la imagen en memoria del *mutatee* llamada *image* para controlar el flujo del proceso y de esta manera poder pararlo e insertar *snippets* en un punto predeterminado (*point*).

MATE usa estas características en dos puntos transcendentales del proceso, la monitorización y la sintonización. Cuando MATE inicia una aplicación MPI inserta en ella una serie de *snippets* que envían información al Analyzer y cuando este decide que cambios realizar en el programa, es mediante las funciones de esta API que MATE aplica los cambios en el programa.

1.2 MATE

Es en este tipo de entornos, *clusters* de maquinas en paralelo, que MATE aplica una serie de funciones para optimizar el funcionamiento de estas.

Existen muchas aplicaciones que intentan optimizar el uso de estas supercomputadores para incrementar su rendimiento. MATE es una propuesta innovadora que va un paso mas allá en este proceso de optimización y lo automatiza.

MATE tiene como objetivo encontrar cuellos de botella en la ejecución de la aplicación y modificarla mientras esta se ejecuta, es decir mientras se encuentra en memoria, para resolverlos y de esta forma aumentar el rendimiento general de la computadora.

1.3 Aportación

Dado que se trata de software ya creado, nuestra aportación al proyecto consiste en dotarlo de las herramientas necesarias para seguir evolucionando, y de esta forma hacer posible que se convierta en un producto que puedan usar terceras partes.

Para que esto sea posible MATE debe disponer de unos mínimos requisitos de calidad que permitan a otros usuarios ejecutar y usar la aplicación pero que también ellos puedan comprender el código y participar de la evolución de este.

Además de el trabajo realizado sobre software ya existente también añadiremos a MATE nuevas características que lo harán mas cómodo de usar y permitirán que llegue a más usuarios.

1.4 Alcance

Nuestro proyecto, pese a que trabajemos con MATE, no modificará el funcionamiento básico de este. MATE trabaja con funciones complejas de análisis y sintonización de aplicaciones, que se escapan de nuestro campo de conocimiento y por tanto no modificaremos. Estas funciones han sido desarrolladas por nuestros antecesores en este proyecto y cumplen con su función.

1.5 Objetivos

Debido a su envergadura, los objetivos de este proyecto se dividirán entre los diferentes miembros del equipo de desarrollo. A continuación se listarán los objetivos generales y se especificará la división de los mismos.

	Objetivo	Prioridad	Miembro Asignado
1	Crear especificaciones del entorno de desarrollo	Prioritario	Grupo
2	Implantar entorno de desarrollo	Crítico	
2.1	Herramienta de colaboración	Crítico	Rodrigo Echeverría, Antonio Pimenta
2.2	Herramienta de control de versiones	Crítico	Antonio Pimenta
2.3	Herramienta de construcción	Crítico	Noel De Martin
3	Construir la metodología de desarrollo.	Crítico	Grupo
3.1	Guía de estilo de documentación.	Crítico	Rodrigo Echeverría
3.2	Guía de estilo de codificación.	Crítico	Noel De Martin
3.3	Guía de estilo de documentación de código.	Prioritario	Rodrigo Echeverría
3.4	Guía de estilo de desplegamiento.	Prioritario	Antonio Pimenta
4	Aplicar la metodología y especificaciones a MATE.	Crítico	
4.1	Aplicación sobre las clases comunes (Common)	Crítico	Noel De Martin
4.2	Aplicación sobre el módulo DMLib.	Crítico	Noel De Martin
4.3	Aplicación sobre el módulo AC.	Crítico	Antonio Pimenta
4.4	Aplicación sobre el módulo Analyzer.	Crítico	Rodrigo Echeverría

5	Desarrollo de nuevas características	Secundario	
5.1	Crear un instalador para cualquier versión de Linux.	Secundario	Rodrigo Echeverría, Antonio Pimenta
5.2	Crear un lector de configuraciones flexible.	Secundario	Noel De Martin
5.3	Crear un sistema de cerrado de MATE.	Secundario	Rodrigo Echeverría, Antonio Pimenta
6	Crear documentación de MATE para futuros colaboradores y usuarios.	Prioritario	Grupo

1.6 Estructura del documento

Este documento esta dividido en 8 secciones siendo esta la primera donde se introduce el tema del proyecto y se da información necesaria para la comprensión del conjunto.

La segunda sección contiene la información relativa a la gestión del tiempo usado para realizar el proyecto. En ella se expresan los requerimientos que debe satisfacer el producto final y se comprueba la viabilidad de estas metas. Además se hace una descripción a fondo del funcionamiento de MATE y de sus módulos.

En la sección tercera se hace hincapié en la importancia de la calidad del software, que es uno de los fundamentos de la realización de este proyecto. En esta sección también se hjabla de los diferentes modelos que existen para asegurar la calidad en un producto software y especificamos cual de ellos es mas adecuado a MATE.

La cuarta sección habla del diseño y desarrollo de la metodología, y se puede dividir en dos bloques: el primero que habla del entorno de desarrollo que se propone para MATE y el segundo que incluye las especificaciones según las cuales se regirá este y futuros proyectos sobre MATE.

La quinta sección hace referencia a las actividades que se han llevado a cabo sobre el código de MATE referentes a la metodología descrita en la sección 4. Esta sección incluye detalles de como se han aplicado las diferentes especificaciones en MATE y como se ha usado el entorno de desarrollo.

La sección sexta habla de las nuevas características con las que se ha dotado a MATE. Se especifican las diferentes fases del desarrollo de estos componentes nuevos, cual fue el motivo de su creación y los resultados de esta.

En la séptima sección se explican los resultados del proyecto incluyendo posibles mejoras aplicables a este y ideas para futuros proyectos MATE. En esta sección se valora el trabajo realizado y se analiza el resultado obtenido.

Por último en la sección octava se incluyen las referencias a los documentos y paginas web usadas para docuentarse.

2. Plan de proyecto y estudio de viabilidad

En esta sección se detalla en que marco se sitúa este proyecto y se plantean los requisitos específicos que deberán ser satisfechos por el producto final, además, se propone una planificación para realizar el proyecto en el tiempo estimado. Por ultimo se realiza un estudio para comprobar que se trata de un proyecto viable económica y técnicamente.

2.1 Estado actual

2.1.1 MATE: el proyecto

MATE es una aplicación desarrollada por Anna Sikora en el año 2003 como parte de su tesis doctoral. Esta aplicación tiene como objetivo mejorar el rendimiento de sistemas distribuidos y pretende llevar un paso mas allá esta tarea por medio de automatizar el proceso de optimización y realizarlo de forma dinámica, sin parar la ejecución de la aplicación.

El hecho de analizar el comportamiento de la aplicación objetivo y aplicar los cambios adecuado produce un *overhead* en el tiempo de ejecución y por tanto el reto es realizar este proceso sin influir negativamente en el tiempo total de ejecución para que el trabajo de optimización sea efectivo. El análisis que se realiza es mas superficial y ligero para poder obtener soluciones en tiempo real.

En su inicio MATE se basaba en los siguientes puntos:

- La monitorización, análisis y modificaciones se realizan en tiempo de ejecución, por contra a otras aplicaciones similares que realizan análisis postmortem o que los cambios deben ser realizados a mano.
- MATE no debe influenciar negativamente en el tiempo de ejecución de la aplicación, la intrusión tanto en cada una de las maquinas como en la red debe ser mínima para hacer eficaz la optimización.
- El sistema debe ser aplicable a diferentes aplicaciones pudiendo adoptar diferentes modelos de análisis.
- El sistema debe estar adaptado a personal no experto, parte de los beneficios de MATE es que automatiza los aspectos mas complejos de la optimización de aplicaciones paralelas y por tanto hace la participación de expertos innecesaria.

Posteriormente a su creación MATE ha derivado en varios trabajos haciéndolo parte de diferentes proyectos, cada uno de estos proporcionándole nuevas funcionalidades y de esta forma perpetuando su evolución. Entre otros Paola Caymes y Andrea Martínez han trabajado con MATE para dotarlo de nuevas características, la creación automática de técnicas de sintonización y nuevos modelos de

rendimiento respectivamente.

El proyecto MATE sigue activo y existen lineas de investigación que en el futuro harán que MATE sea aplicable en sistemas de gran escala. Esto se conseguirá jerarquizando el sistema de análisis de forma que no se lleve a cabo en un único nodo sino que se distribuya en niveles para hacer posible el análisis de miles de nodos.

2.2 Requisitos funcionales y no funcionales

Al tratarse de un proyecto que se basa en una aplicación ya desarrollada los objetivos naturales de un proyecto de desarrollo se ven llevados a un segundo plano.

Por tanto ,aunque lo primero que puede surgir al pensar en requisitos sobre este proyecto sean requisitos de MATE (mejorar rendimiento de la aplicación, no sobrecargar demasiado la ejecución), sería un error enfocarlo de esa manera, ya que este proyecto no se trata de crear MATE, sino de crear una versión como producto software del mismo.

Esto es, desarrollar y aplicar una serie de procedimientos para producir, a partir de la versión existente de MATE, una aplicación adecuada a un uso general y proporcionar las herramientas necesarias para la continuidad de su desarrollo.

2.2.1 Requisitos funcionales

Debido al hecho de que MATE es un programa plenamente operativo los requisitos funcionales se reducen a que la funcionalidad actual se mantenga y a añadir algunos pequeños módulos que hacen mas cómodo el uso de MATE aunque no modifican su funcionamiento principal, como por ejemplo un sistema de Shut Down (Apagado). El resto de requisitos aparecerán al realizar los test de prueba sobre los cuales trabajaremos para modificar el programa.

1.- Mantener funcionalidad actual de MATE.

2.- Adaptar a nuevos entornos.

3.- La aplicación debe cerrarse de forma controlada.

4.- Permitir lectura de archivos de configuración.

2.2.2 Requisitos no funcionales

En nuestro caso los requisitos no funcionales se basan en homogeneizar la codificación y documentación. Por lo tanto harán referencia a las guías de especificación. También serán encontrar los posibles errores del sistema mediante un sistema cíclico de testeo.

Dado que, como hemos explicado anteriormente, los requisitos funcionales básicos de MATE ya están satisfechos, la carga de nuestro proyecto se encuentra en los no funcionales. Estos, pese a no añadir funcionalidad a la aplicación incrementan su calidad.

1.- Homogeneizar codificación y documentación.

2.- Realizar proceso de testeo.

3.- El programa debe funcionar en todas las distribuciones de linux.

2.2.3 Catalogación y priorización de requisitos

Entre los requisitos de este proyecto el más importante es el de mantener la funcionalidad actual. Los siguientes serían referentes a tener un buen soporte y documentación para que futuros colaboradores o desarrolladores de MATE puedan solventar errores residuales. Y para acabar tener el mayor rango de sistemas compatibles posible.

Priorización de los requisitos:

Requisitos	Prioridad
Mantener funcionalidad actual de MATE.	Esencial
Adaptar a nuevos entornos.	Condicional
La aplicación debe cerrarse de forma controlada.	Opcional
Permitir lectura de archivos de configuración.	Opcional
Homogeneizar codificación y documentación.	Esencial
Realizar proceso de testeo.	Esencial
El programa debe funcionar en todas las distribuciones de linux.	Opcional

Relación de requisitos con objetivos del proyecto:

Requisitos	Objetivos
Mantener funcionalidad actual de MATE.	5
Adaptar a nuevos entornos.	6
La aplicación debe cerrarse de forma controlada.	5
Permitir lectura de archivos de configuración.	5
Homogeneizar codificación y documentación.	1, 4
Realizar proceso de testeo.	1,2,3
El programa debe funcionar en todas las distribuciones de linux.	6

2.3 Descripción del sistema MATE

2.3.1 Lógica del sistema

En nuestro entorno es necesario realizar sintonización dinámica. Desde el punto de vista funcional podemos distinguir tres fases básicas:

- **Monitorización**

  Esta fase es la encargada de obtener información sobre la ejecución de la aplicación. Esto no es un proceso trivial ya para obtener medidas de rendimiento de la aplicación esta debe estar en marcha, por lo tanto esta tiene que incluir fragmentos de código que se encargan de captar eventos y notificar de estos al AC. Estos fragmentos pueden haber sido introducidos en el código original por el programador o se pueden introducir de forma directa en el programa compilado y en ejecución. En nuestro caso optamos por la segunda opción de nos da un grado de versatilidad mayor.

  Para que esto sea posible la aplicación que queremos monitorizar debe ejecutarse bajo la tutela de un proceso de control y recolección [AC]. Este proceso se encarga, en una primera instancia, de introducir en la aplicación objetivo una serie de funciones baliza que monitorizan una parte especifica de la aplicación, y posteriormente de almacenar y tratar los resultados. Es necesario también cargar en la aplicación una librería que contiene las herramientas necesaria para la monitorización de la aplicación [DMLib].

  Estas medidas de rendimiento pueden ser de varios tipos, por ejemplo tiempos de ejecución de funciones clave o repeticiones de llamadas a una misma función. También se pueden medir las veces que ocurre un evento complejo.

  No obstante, para conseguir obtener medidas de rendimiento, se debe conocer profundamente la aplicación que se desea optimizar. MATE utiliza unos modelos de rendimiento adaptados estrechamente a la aplicación objetivo. Estos proporcionan información sobre como obtener datos útiles para la medición del rendimiento y medidas a tomar para mejorarlo.

  Por ultimo debe existir un medio adecuado para transmitir los datos recolectados a una proceso de análisis para obtener resultados y, si es posible, sintonizar la aplicación para mejorar su rendimiento.

  Algunos aspectos que se deben tener en cuenta al trabajar en diferentes máquinas son las posibles diferencias en los relojes de sus procesadores y el tiempo de transmisión de los datos. Es importante que una serie de eventos que ocurren en la aplicación lleguen en ese orden al analizador.

- **Análisis**

  Una vez obtenidas las mediciones adecuadas un programador experto y conocedor de la aplicación a optimizar, sabría encontrar los cuellos de botella en su ejecución y podría proponer soluciones a estos. No obstante esta es una tarea pesada y duradera y como hemos dicho requiere un nivel de conocimiento muy alto.

  Como alternativa existen métodos de análisis automático. Estas herramientas identifican problemas en la ejecución de la aplicación y incluso proporcionan soluciones a estas. Para que esto sea posible se le debe proporcionar a la herramienta de análisis una base de conocimiento sobre la aplicación así como posibles zonas criticas donde buscar problemas. El proceso de producir unos modelos que permitan al analizador automático identificar exitosamente estos problemas no es fácil. Además a pesar de que se obtenga un modelo valido las soluciones proporcionadas serán estrictamente útiles para el comportamiento que tuvo la aplicación durante esa concreta ejecución.

  Con MATE se intenta atajar este problema no solo automatizando el proceso de análisis si no además realizarlo de forma dinámica. Esto implica que el análisis de puede realizar mientras la aplicación se ejecuta eliminando la necesidad de un archivo donde almacenar los datos de medición. Este método, pese a que conserva muchas de las desventajas del análisis manual, permite un análisis adaptativo en aplicaciones iterativas ya que durante la ejecución se puede modificar dinámicamente la monitorización y instrumentación de la aplicación.

  En el caso de MATE el análisis de produce dinámica y automáticamente, sin necesidad de un archivo con los datos de monitorización, ya que el modulo Analyzer recibe los eventos recolectados directamente.

- **Optimización**

  La fase de optimización (Tuning) es en la cual aplicamos los cambios adecuados para mitigar los problemas detectados. Estos cambios se deben hacer en el código de la aplicación ya que forman parte de esta, y por tanto es normalmente necesario cerrar la aplicación, modificarla recompilarla y volverla a ejecutar. Si los cambios aplicados son adecuados se debería observar una mejora en el rendimiento.

  No obstante este método de optimización requiere la atención directa del programador y además es necesario recompilar, y por tanto cerrar, la aplicación. Los cambios realizados podrían ser inútiles si en la siguiente ejecución el programa se comporta de forma distinta, debido, por ejemplo, a diferentes valores de entrada.

  MATE proporciona un modelo de optimización autónomo que no requiere de la intervención del programador y que se realizar dinámicamente, es decir, sin cerrar la aplicación. Además el hecho

de que se sintonice la aplicación de forma dinámica permite que se realice de forma adaptativa, así aunque cambien las condiciones de la ejecución el programa sigue operando de forma eficiente.

En mate la aplicación de estas variaciones es llevada a cabo por el AC (Application Controller), este dispone de un sintonizador (Tuner) que contiene las herramientas necesarias para instrumentar la aplicación dinámicamente. Usando la información proporcionada por el Analyzer el AC introduce los cambios necesarios en la aplicación mientras esta se ejecuta.

Estas tres fases tienen que estar realizándose continuamente, dinámicamente y automáticamente mientras el programa está en ejecución. Para que esto método sea efectivo la aplicación objetivo debe ser iterativa (ejecución de un bucle que realiza de forma repetida una serie de instrucciones), y se obtiene la eficacia en procesos largos y que usen muchos recursos.

Básicamente, MATE está formado por los siguientes módulos que operan conjuntamente, controlando y intentando mejorar el rendimiento de la aplicación objetivo.

- **AC (Application controller):** Se trata de un proceso *daemon* que controla la ejecución de la aplicación. Este proceso se inicia de forma manual en cada uno de los nodos y es el que se encarga de inicializar las tareas (aplicación) que se va a monitorizar.

  Lo primero que hace el modulo es cargar en la imagen del proceso que representa la aplicación una librería dinámica llamada DMLib, que contiene las herramientas necesarias para la monitorización dinámica de la aplicación. Esta librería debe inicializarse con los datos del analizador (*host* y puerto) para que sea posible la comunicación entre los nodos y el modulo Analyzer.

  A continuación se buscan los puntos indicados por el Analyzer donde se deberán introducir los monitores que recopilan información de la ejecución. Esto es diferente para cada aplicación y viene indicado por unos módulos incluidos en el Analyzer.

  Una vez la aplicación esta lista y en linea se procede a iniciar la aplicación, y una vez esta está iniciada los datos recopilados se envían como eventos al Analyzer directamente desde la aplicación usando funciones de la DMLib.

  En este punto el AC ha de estar preparado para recibir instrucciones del Analyzer a la vez que controla la aplicación en ejecución ya que esta puede producir nuevos eventos que reportar. En cuanto se recibe una petición de sintonización (Tuning request) el AC informa a las tareas de que han de ser modificadas y estas a su vez pedirán al Tuner que las actualice, siempre esperando a que se de el momento adecuado. Esta sincronización de las modificaciones es posible gracias a

los *breakpoints* (Puntos de parada), que indican los lugares donde se debe insertar la instrumentación nueva y paran la a ejecución de la aplicación para que estas modificaciones sean posibles.

Este ciclo de comunicación-modificación se realiza de forma cíclica hasta que se han realizados todas las modificaciones o la aplicación se ha cerrado.

- **DMLib (Dynamic Monitoring Library):** Una librería compartida que se carga dinámicamente dentro de las tareas de la aplicación para facilitar la instrumentación y recolección de datos. La librería contiene funciones que son responsables de el registro de eventos con todos los atributos necesarios para entregarlos para análisis. Usamos la función `loadLibrary` de Dyninst para cargar la librería una vez la tarea ya ha sido iniciada.

  Esta librería debe ser inicializada con los datos del Analyzer para hacer posible la transmisión de los datos recolectados. Esta inicialización se inserta como un snippet en la aplicación.

  Una vez cargada e inicializada esta librería implanta las conexiones necesarias vía proxy para comunicarse con el Analyzer.

- **Analyzer:** Es el proceso que se encarga de analizar el rendimiento de la aplicación, detecta problemas de rendimiento a tiempo real y solicita los cambios idóneos para mejorarlo.

  Mediante un sistema de captura de eventos este módulo obtiene información sobre la ejecución de la aplicación y aplica unas funciones especificas a la aplicación para identificar problemas y proporcionar al AC posibles soluciones. Este procedimiento es también cíclico y en cada iteración ser manejan varios eventos que resultan en soluciones para el AC.

  Pese a que es el A quien se ejecuta junto a la aplicación el Analyzer dispone de una abstracción de esta que usa para identificar los eventos con de cada una de las tareas que se ejecutan en los diferentes nodos.

- **Common:** Por ultimo existe un modulo en MATE que contiene las clases compartidas que son usadas por los demás módulos. Este módulo cumple el objetivo de reutilización de código y encapsulación de los diferentes componentes de MATE.

2.3.2 Descripción física

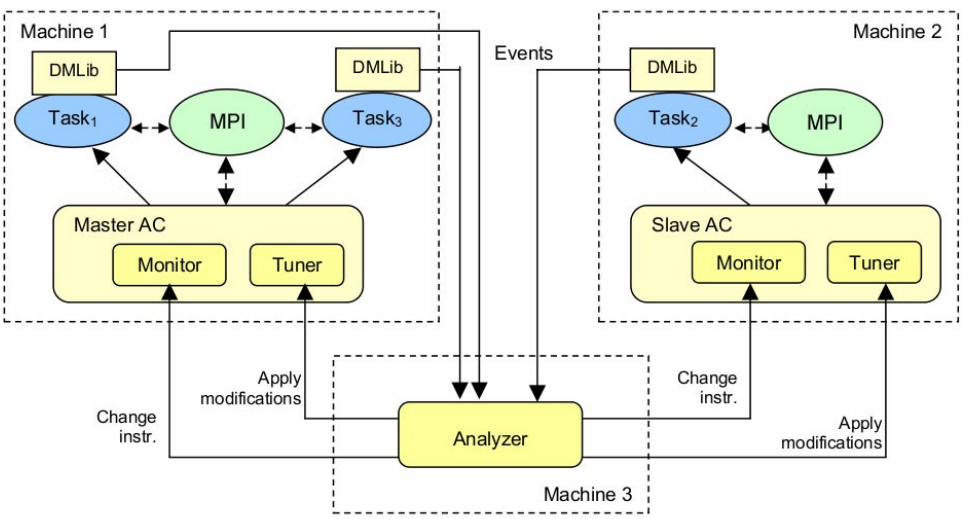


Figura 1: Esquema del sistema extraido de la tesis sobre MATE

En cuanto a aplicación, MATE se divide en dos partes diferenciables, el Analyzer y el Application Controller. Estos son los dos ejecutables que cumplen funciones distintas y se complementan para formar en entorno de sintonización que es MATE. Al ser una herramienta para programas distribuidos, MATE se ejecuta en diferentes máquinas, en concreto el modulo AC se ejecuta en cada uno de los nodos que ejecutan la aplicación objetivo, mientras que el Analyzer es un programa centralizado que se ejecuta en una sola máquina.

Al empezar la aplicación, MATE distribuye un proceso de AC en cada máquina para controlar el comienzo de las tareas.

Otro de los componentes de MATE de la librería DMLib (*dynamic monitoring library*), cuando comienza una nueva tarea MPI, el AC carga la librería compartida de monitorización en la memoria de la tarea para permitir la instrumentación de esta. Esto le permite al Analyzer añadir/eliminar eventos dinámicamente para recolectar información y realizar la sintonización.

2.4 Alternativas para el entorno

Considerando que el proyecto consiste en crear una metodología de desarrollo productiva que sirva como base a los futuros desarrolladores de MATE, para posteriormente aplicarla al software existente, las alternativas se definen en el marco de herramientas de desarrollo, colaboración y en la selección de la propia metodología.

Para proyectos de este tipo existen varias herramientas que nos asisten a la hora de trabajar en conjunto, controlar la evolución del proyecto y finalmente compilar y testear los resultados. A continuación se exponen las alternativas que hemos encontrado y la selección que usaremos, basándonos en cuales de ellas se adecuan mas a MATE.

Alternativa 1 (H. Colaboración – Redmine)

Redmine es una herramienta de colaboración que actúa como solución todo-en-uno ya que posee soporte multiproyecto, acceso basado en roles, sistema de seguimiento, gestor de calendarios y diagramas de gantt, soporte a wikis y foros y compatibilidad con diversos gestores de versiones concurrentes. En cuanto a coste, encontramos que se trata de una herramienta de código libre y gratuito.

Alternativa 2 (H. Colaboración – Trac)

Trac es una herramienta de gestión de proyectos que enlaza una base de datos de errores de software, un sistema de versiones y el contenido de una *wiki* de colaboración. En cuanto a coste, encontramos que se trata de una herramienta de código libre y gratuito.

Alternativa 3 (H. Control Versiones – CVS)

CVS (Concurrent Versions System) es una aplicación cliente-servidor donde el servidor se encarga de guardar un historial de las diferentes versiones de cada uno de los archivos que componen un proyecto; los clientes pueden acceder a estos archivos de forma directa o bien remotamente. En cuanto a coste, encontramos que se trata de una herramienta de código libre y gratuito.

Alternativa 4 (H. Control Versiones – SVN)

SVN (Subversion) es una herramienta de control de versiones que tiene la peculiaridad de que mantiene un único número de versión para un conjunto de archivos, de forma que lo que conserva es un estado determinado del proyecto en general. Además soporta el acceso desde redes, permitiendo a usuarios modificar los archivos desde distintas ubicaciones. En cuanto a coste, encontramos que se trata de una herramienta de código libre y gratuito.

Alternativa 5 (H. Desarrollo – Buildbot)

Buildbot es una herramienta de desarrollo software iterativa que automatiza los procesos de compilación y testeo. Posee soporte para control de versiones (CVS, SVN...). En cuanto a coste, encontramos que se trata de una herramienta de código libre y gratuito.

Alternativa 6 (H. Desarrollo – Tinderbox)

Tinderbox es una suite que proporciona capacidades de continua integración, básicamente permite manejar proyectos software y probar su funcionamiento en diversas plataformas. En cuanto a coste, encontramos que se trata de una herramienta de código libre y gratuito.

2.4.1 Solución propuesta

En cuanto a la herramienta de seguimiento utilizaremos Redmine por dos razones: primero, es más completa y nos ofrece en una misma aplicación todas las herramientas que necesitamos y, segundo, la experiencia del director de proyecto con esta herramienta nos servirá como guía.

Sobre la herramienta de control de versiones utilizaremos SVN, ya que nos interesa más guardar el proyecto por versiones en general, sin hacer hincapié en los archivos individuales. Además nos proporciona una fácil integración con apache para poder preparar un servidor a través del cual acceder al repositorio.

Y finalmente la herramienta de desarrollo que utilizaremos es Buildbot por su capacidad de integración con SVN y porque nuestros intereses no se dirigen especialmente al testeo multiplataforma, sino a un ciclo iterativo de compilación-testeo-recodificación.

2.5 Viabilidad técnica

Está sección está destinada a determinar que conocimientos serán necesarios para realizar el proyecto, y de esta forma poder prepararse adecuadamente para las tareas que realizaremos en adelante. El proyecto se realiza alrededor de una aplicación existente, MATE, que es compleja y trabaja en un campo de la informático especializado como es la computación de alto rendimiento y la modificación dinámica de aplicaciones en ejecución. Esto hace que sea necesario, si bien no tener un conocimiento exhaustivo del tema, aprender algunos aspectos básicos, especialmente sobre las herramientas usadas como Dyninst o MPI.

2.5.1 Lenguaje de programación

Para poder participar del desarrollo de una aplicación es absolutamente necesario conocer el lenguaje en el cual esta ha sido escrita. En el caso de MATE este lenguaje es C++.

Debido al *background* de los participantes en este proyecto, no disponemos de experiencia en este

lenguaje y será necesario realizar un estudio de C++ así como de los patrones usados en MATE.

2.5.2 MPI

En cuanto al aspecto de MATE que trata con aplicaciones paralelas es relevante conocer los métodos de pase de mensajes entre este tipo de procesos ya que en esto se basa la arquitectura de la aplicación.

MATE se aplica sobre aplicaciones paralelas que implementan MPI (Message Passing Interface) como herramienta de comunicación entre sus diferentes procesos. Por tanto es importante conocer el funcionamiento de esta librería.

Aprendizaje realizado:

Para disponer de los conocimientos necesarios sobre MPI hemos asistido a una serie de clases teóricas y prácticas donde hemos aplicado los conocimientos teóricos adquiridos para crear pequeñas aplicaciones paralelas que se comunican con este protocolo.

2.5.3 Dyninst

Como hemos dicho, una parte importante de MATE es el hecho de que los cambios en la aplicación se realizan de forma dinámica y por tanto se requiere una herramienta que permita este tipo de acceso al programa objetivo.

En este caso usamos Dyninst, una librería que nos permite manejar un proceso en ejecución y insertar instrumentación en el. Dyninst es una herramienta compleja pero muy potente que nos permite encontrar puntos concretos en el código siendo ejecutado e insertar o modificar elementos en el.

Aprendizaje realizado:

Al igual que con MPI se ha asistido a una serie de clases donde se han adquirido conocimientos teóricos sobre Dyninst para posteriormente aplicarlos en laboratorios prácticos, con el resultado de pequeños programas (*mutator* y *mutatee*) que usan Dyninst para obtener datos de la ejecución de estos como el numero de veces que cierta función se ejecuta o el parámetro que se le pasa.

2.6 Planificación

2.6.1 WBS (Work Breakdown Structure)

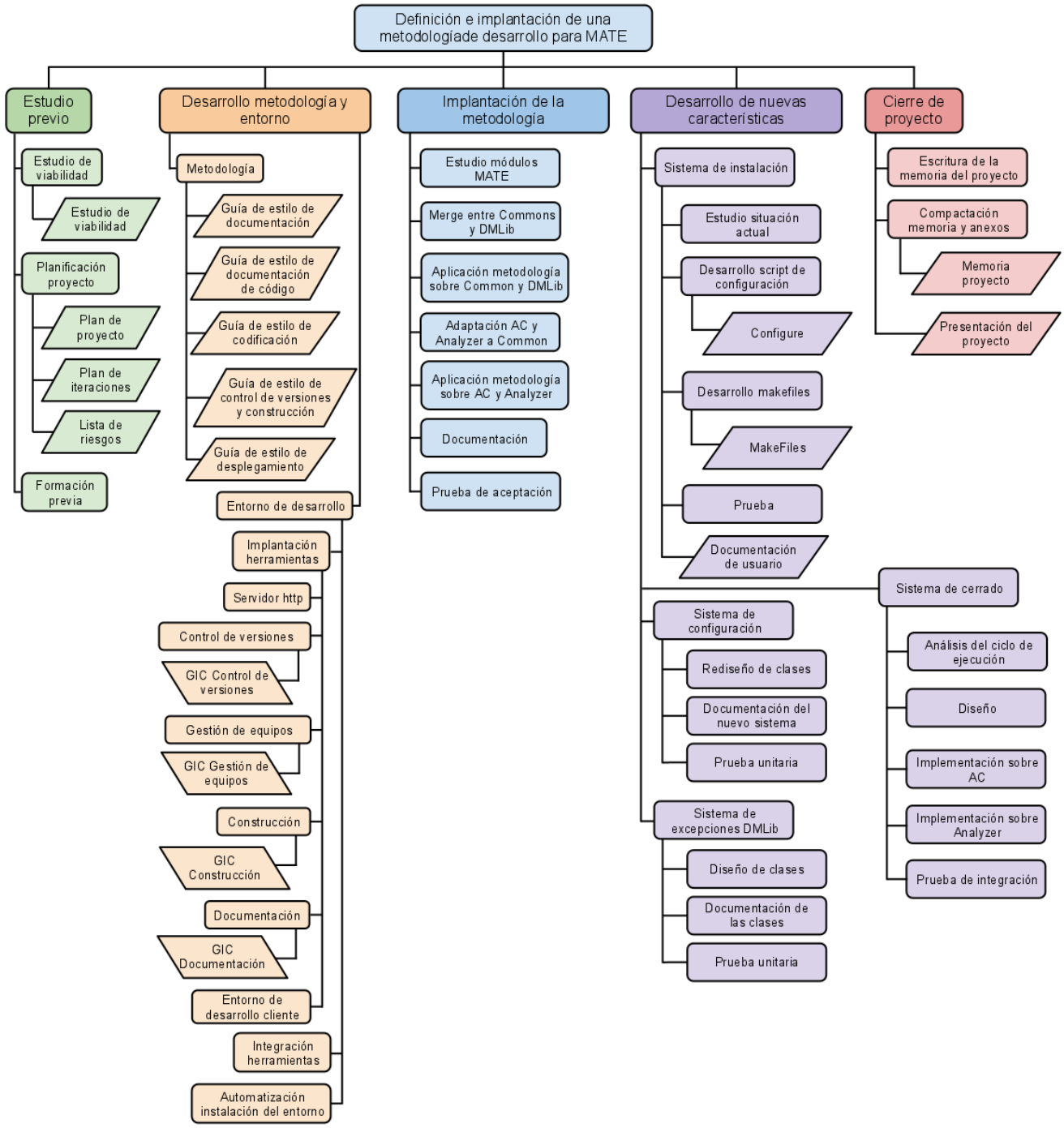


Figura 2 Diagrama WBS del proyecto

2.6.2 Fases y actividades del proyecto

Fase	Actividad	Descripción	Iterativa
Estudio previo	Estudio de viabilidad	Estudio para analizar las posibilidades y mejores alternativas para realizar el proyecto y si estás son posibles con los recursos disponibles en el tiempo requerido.	No
	Planificación del proyecto	Análisis sobre las tareas que compondrán el proyecto, su calendario, los recursos necesarios para ejecutarlas y los riesgos que comportan a la consecución del proyecto.	No
	Formación previa	Estudio sobre los temas relacionados con paso de mensajes (MPI) y sintnización de procesos (Dyninst)	No
Desarrollo metodología y entorno	Desarrollo guías de estilo	Creación de los documentos que conforman la base de la metodología a implantar: guías de estilo de documentación, codificación, construcción, control de versiones, etc.	Si
	Implantación entorno	Selección de las diferentes herramientas que forman el entorno de desarrollo, implantación e integración de las mismas.	Si
Implantación metodología	Estudio módulos MATE	Estudio sobre el código en su versión original de cada módulo de MATE.	No
	Combinación DMLib y Commons	Eliminación de las clases redundantes entre Commons y DMLib y refactorización de AC y Analyzer en consecuencia.	Si
	Documentación y refactorizacion	Documentación sobre el código de cada módulo y refactorización derivada de las guías de estilo.	No
	Prueba unitaria	Prueba unitaria de las clases que componen cada módulo.	No
	Documentación	Extracción y compilación de la documentación sobre código.	No
	Prueba de aceptación	Prueba del sistema completo para detectar posibles errores en la refactorización.	No
Desarrollo de nuevas características	Sistema de instalación	Desarrollo de un sistema de instalación capaz de automatizar la búsqueda de dependencias y la compilación en el mayor grado posible.	Si
	Sistema de configuración	Desarrollo de un sistema de lectura de configuraciones para MATE fácilmente ampliable con diferentes tipos de entradas.	Si
	Sistema de excepciones de DMLib	Desarrollo de un sistema para DMLib de detección y notificación de errores.	Si
	Sistema de cerrado	Desarrollo de un sistema capaz de cerrar el entorno de forma centralizada y controlada.	Si
Cierre de proyecto	Escritura y compilación de la memoria	Escritura de la memoria del proyecto y compilación del documento junto con los anexos que lo acompañan.	No
	Exposición del proyecto	Exposición del proyecto ante un tribunal para su evaluación.	No

Tabla 1: Fases de desarrollo del proyecto

2.6.3 Recursos del proyecto

Recursos humanos

- 3 Programadores-Analistas: Noel De Martín, Rodrigo Echeverría, Toni Pimenta
- 1 Project Manager: Joan Piedrafita

Recursos de infraestructura

- Servidor de proyectos, colaboración y builds virtualizado
  - Equipo: Dell PowerEdge R515
  - Procesador: 2 x AMD Opteron 4122 (4 Cores – 2.2 Ghz – L1 3MB / L2 6MB – 95W TDP)
  - Memoria: 8GB Memory for 2 CPUs, DDR3, 1333MHz (8x1GB Single Ranked UDIMMs)
  - Disco: 2x 250GB, SATA, 3.5-in, 7.2K RPM Hard Drive (Hot Plug)

- 8 Nodos de computo cluster DiskLess
  - Equipo: Dell PowerEdge R610
  - Procesador: 2 x Intel Xeon E5620 (4 Cores – 2,4 Ghz – 12 MB Cache – QPI 5,86 Gb/s)
  - Memoria: 12GB DDR3, 1333MHz ECC (12x1GB)

- SAN
  - Almacenamiento: DELL™ PowerVault™ MD3200i, 6 discos SAS 7.2k rpm 500 GB
  - Red de gestión: 2 x SWITCH ETHERNET DELL PowerConnect 5424

- Otros
  - Sistema de alimentación: SAI 8000VA APC
  - Switch control cluster:SWITCH ETHERNET DELL PowerConnect 5448
  - Switch gestión: SWITCH INFINIBAND SDR DE 8 PUERTOS, 4X, 1U.
  - Switch Red MATE: SWITCH ETHERNET DELL PowerConnect 5424
  - Rack PDU (8 Tomas + Ethernet)
  - Chasis Rack 42U
  - CABLE INFINIBAND 2 METROS CON CONEXION X4
  - Cable de interconexión - RJ-45 (M) - RJ-45 (M) - 2 m - UTP - (CAT 6)

Configuración de la infraestructura

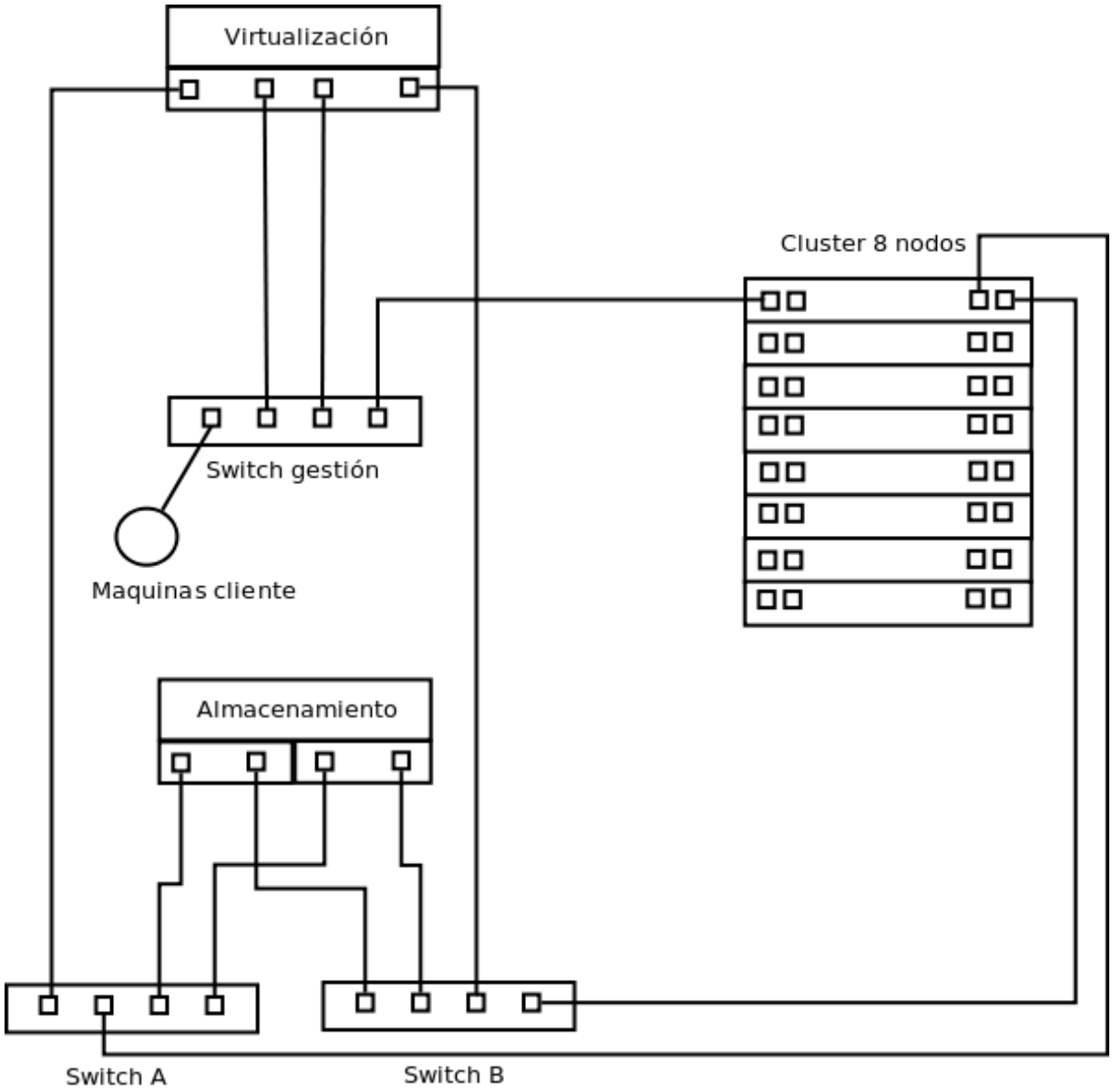


Figura 3 Esquematización del entorno de desarrollo

Calendario de los recursos

Los recursos humanos se utilizarán durante todo el proyecto, sin embargo el cluster y el almacenamiento solo se utilizarán en la segunda parte del proyecto haciendo las pruebas necesarias con aplicaciones paralelas para comprobar los resultados. El resto de recursos materiales también se utilizarán durante todo el proyecto.

2.6.4 Calendario del proyecto

El proyecto se realizará durante en el segundo cuatrimestre del curso 2010/11.

Dependencias

Al tratarse de un modelo lineal, o en cascada, cada fase empezará al terminar la anterior, a excepción de aquellas fases más delicadas que presentarán iteraciones para introducir cambios y correcciones una vez se hagan pruebas.

Las fases del proyecto están representadas en el primer nivel de la jerarquía del diagrama WBS (capítulo 3.1).

La primera fase será el estudio previo, que incluye entrevistas con el cliente, estudios de viabilidad y planificación, así como toda la formación previa necesaria para los analistas.

La segunda fase consiste en el desarrollo del entorno y la metodología. Está fase consiste, por una parte, en la selección de un conjunto de herramientas que conformen un entorno de trabajo para futuros desarrolladores de MATE y la implantación de las mismas (documentación del sistema de instalación y configuración y la automatización del mismo) y, por la otra, en la confección de una serie de manuales (guías de estilo) que sirvan como una referencia sobre el "como hacer" para los futuros desarrolladores.

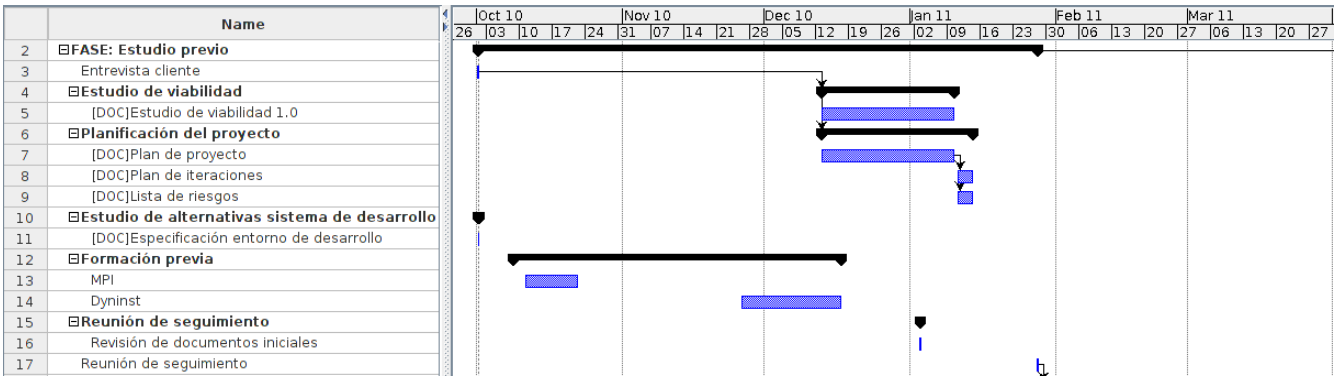
La tercera y cuarta fase se ejecutaran en paralelo. La tercera fase consiste en adaptar el código a las guías de estilo creadas y a la prueba unitaria del mismo. La cuarta fase consiste en el desarrollo de nuevas características para mejorar la calidad del software en general.

Finalmente, la fase de cierre de proyecto consiste en acabar la memoria del proyecto, compactarla con sus anexos y entregarla, además de la exposición del proyecto ante un tribunal.

2.6.5 Calendario temporal

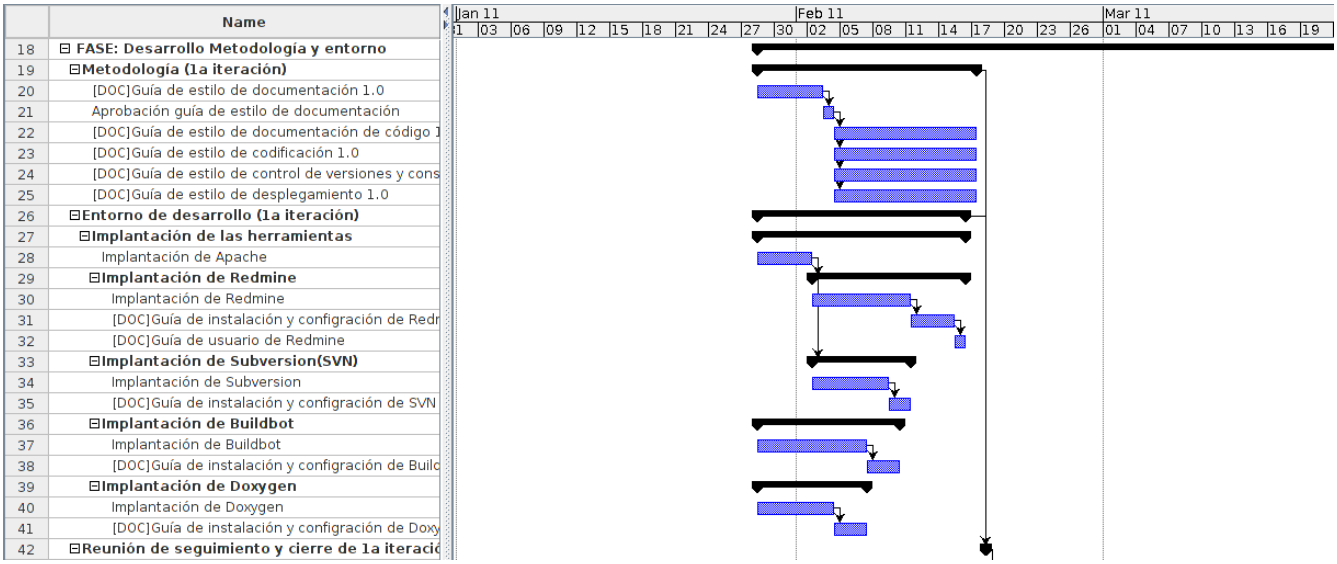
La duración total estimada del proyecto es 248 días que, con una dedicación media de 4h/día, implican 992 horas de trabajo a distribuir entre los tres miembros del equipo.

Fase 1: Estudio previo



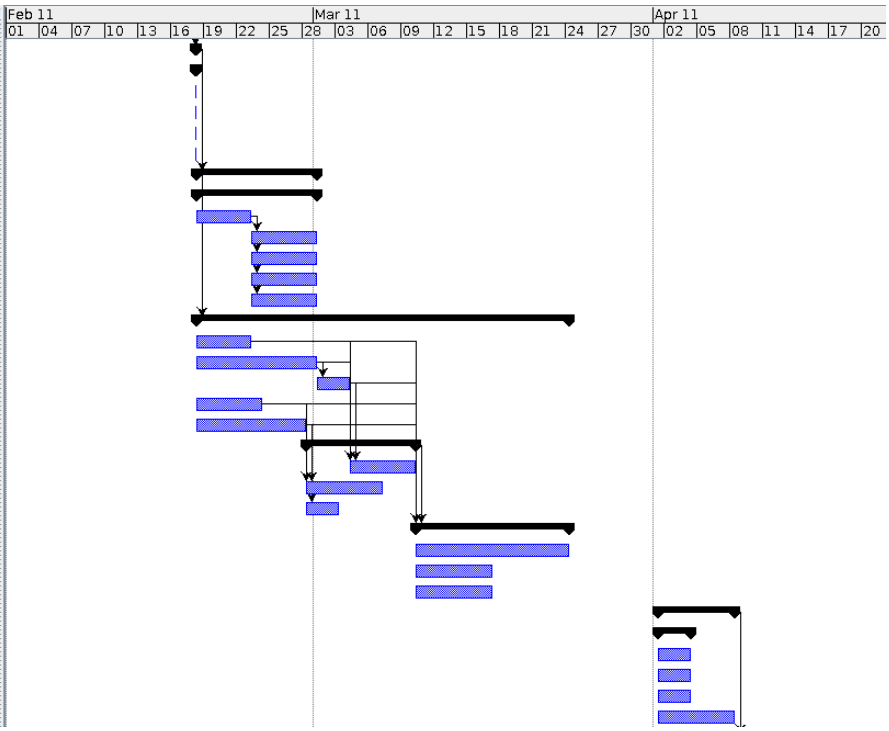
Fase 2: Desarrollo metodología y entorno

1ª iteración



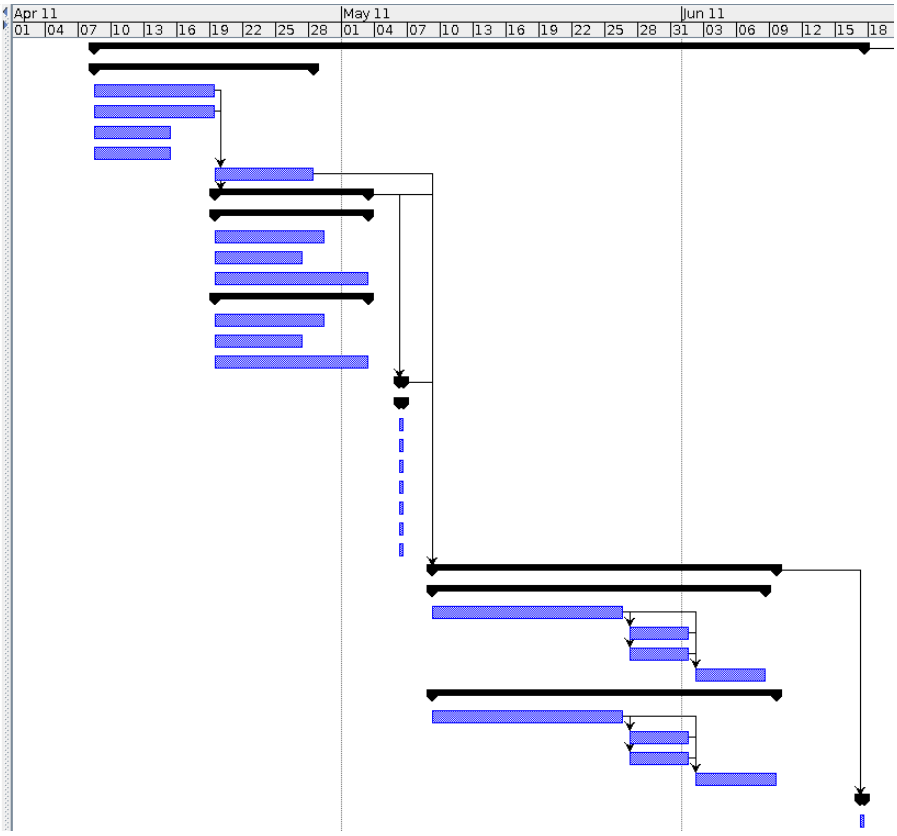
2ª iteración

	Name
42	⊟**Reunión de seguimiento y cierre de 1a iteració**
43	⊟**Actualización documentos iniciales**
44	[DOC]Plan de proyecto 2.0
45	[DOC]Plan de iteraciones 2.0
46	[DOC]Lista de riesgos 2.0
47	Revisión documentación generada hasta la fecha
48	⊟**Metodología (2a iteración)**
49	⊟**Actualización y corrección guias de estilo**
50	[DOC]Guía de estilo de documentación 1.0
51	[DOC]Guía de estilo de documentación de código
52	[DOC]Guía de estilo de codificación 1.0
53	[DOC]Guía de estilo de control de versiones y cor
54	[DOC]Guía de estilo de desplegamiento 1.0
55	⊟**Entorno de desarrollo (2a iteración)**
56	Integración Redmine-Apache
57	Implantación Eclipse
58	Integración Redmine-Eclipse
59	Integración Redmine-SVN
60	Integración SVN-Buildbot
61	⊟**Actualización y corrección de las guias de ins**
62	[DOC]Guía de instalación y configración de Redmi
63	[DOC]Guía de instalación y configración de SVN y
64	[DOC]Guía de instalación y configración de Buildb
65	⊟**Automatización instalación entorno**
66	Script de instalación de Redmine
67	Script de instalación de SVN
68	Script de instalación de Buildbot
69	⊟**Reunión de seguimiento y cierre de 2a iteració**
70	⊟**Actualización documentos iniciales**
71	[DOC]Plan de proyecto 2.0
72	[DOC]Plan de iteraciones 2.0
73	[DOC]Lista de riesgos 2.0
74	Revisión documentación generada hasta la fecha

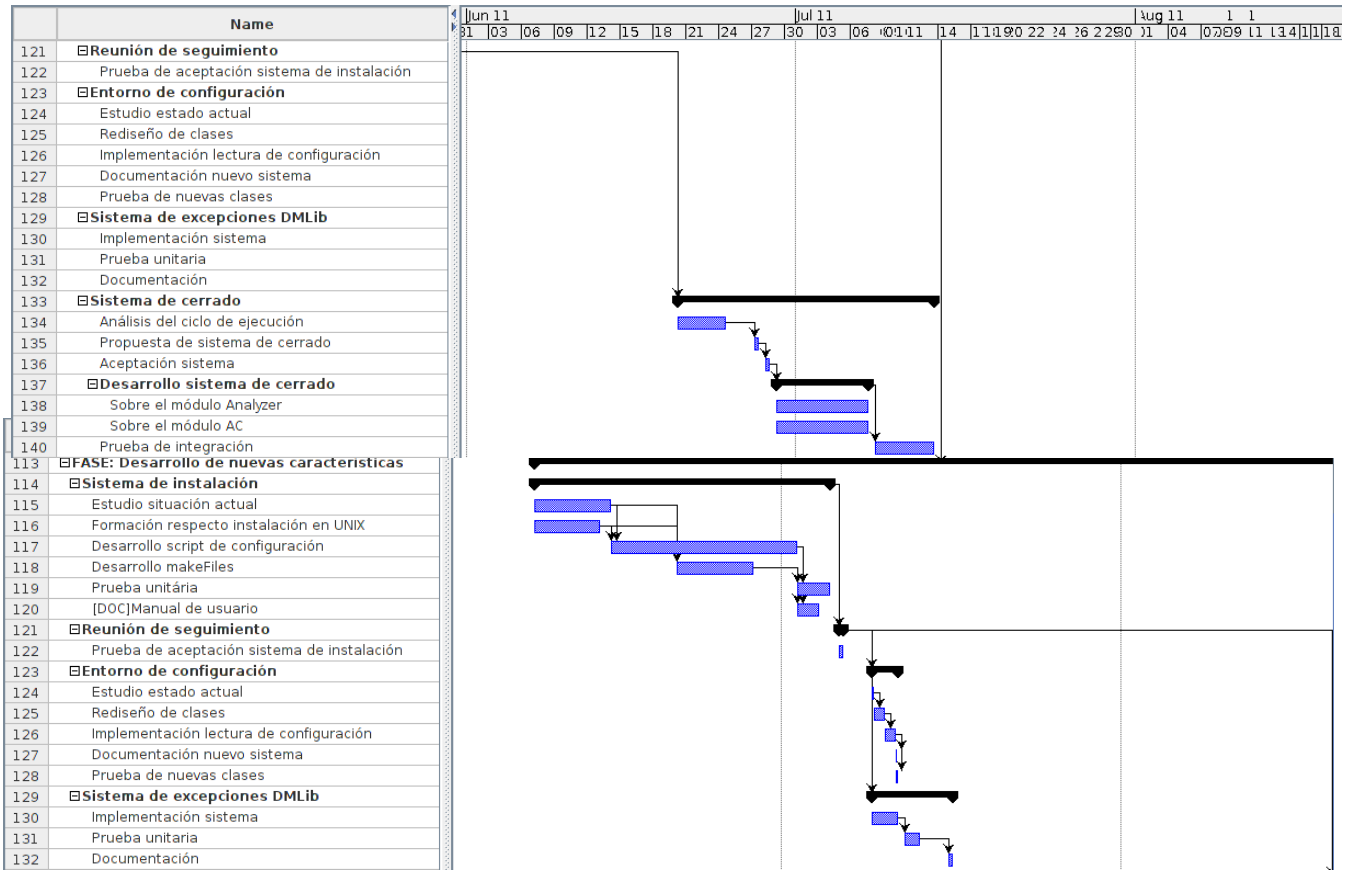


Fase 3: Implantación de la metodología

	Name
75	⊟**FASE: Implantación de la metodología**
76	⊟**Estudio módulos (1a iteración)**
77	Estudio sobre Commons
78	Estudio sobre DMLib
79	Estudio sobre AC
80	Estudio sobre Analyzer
81	Merge entre Commons y DMLib
82	⊟**Implantación (1a iteración)**
83	⊟ **Sobre Common**
84	Documentación
85	Refactorización
86	Prueba unitária
87	⊟**Sobre DMLib**
88	Documentación
89	Refactorización
90	Prueba unitária
91	⊟**Reunión de seguimiento y cierre 1a iteración**
92	⊟**Actualización documentos iniciales**
93	[DOC]Plan de proyecto 2.0
94	[DOC]Plan de iteraciones 2.0
95	[DOC]Lista de riesgos 2.0
96	Revisión sobre el merge Common-DMLib
97	Revisión documentación Common-DMLib
98	Revisión refactorización Common-DMLib
99	Revisión test unitário Common-DMLib
100	⊟**Implantación (2a iteración)**
101	⊟**Sobre AC**
102	Adaptación cambios sobre Common
103	Documentación
104	Refactorización
105	Prueba unitária
106	⊟**Sobre Analyzer**
107	Adaptación cambios sobre Common
108	Documentación
109	Refactorización
110	Prueba unitária
111	⊟**Reunión de seguimiento y cierre 2a iteración**
112	Seguimiento sobre los cambios realizados

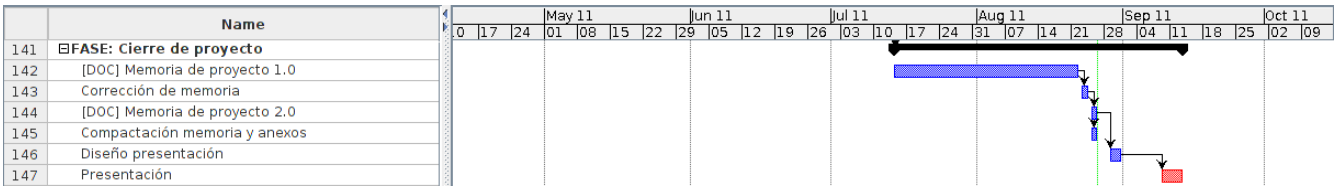


Fase 4: Desarrollo de nuevas características



	Name
121	**Reunión de seguimiento**
122	Prueba de aceptación sistema de instalación
123	**Entorno de configuración**
124	Estudio estado actual
125	Rediseño de clases
126	Implementación lectura de configuración
127	Documentación nuevo sistema
128	Prueba de nuevas clases
129	**Sistema de excepciones DMLib**
130	Implementación sistema
131	Prueba unitaria
132	Documentación
133	**Sistema de cerrado**
134	Análisis del ciclo de ejecución
135	Propuesta de sistema de cerrado
136	Aceptación sistema
137	**Desarrollo sistema de cerrado**
138	Sobre el módulo Analyzer
139	Sobre el módulo AC
140	Prueba de integración
113	**FASE: Desarrollo de nuevas características**
114	**Sistema de instalación**
115	Estudio situación actual
116	Formación respecto instalación en UNIX
117	Desarrollo script de configuración
118	Desarrollo makeFiles
119	Prueba unitária
120	[DOC]Manual de usuario
121	**Reunión de seguimiento**
122	Prueba de aceptación sistema de instalación
123	**Entorno de configuración**
124	Estudio estado actual
125	Rediseño de clases
126	Implementación lectura de configuración
127	Documentación nuevo sistema
128	Prueba de nuevas clases
129	**Sistema de excepciones DMLib**
130	Implementación sistema
131	Prueba unitaria
132	Documentación

Fase 5: Cierre de proyecto



	Name
141	**FASE: Cierre de proyecto**
142	[DOC] Memoria de proyecto 1.0
143	Corrección de memoria
144	[DOC] Memoria de proyecto 2.0
145	Compactación memoria y anexos
146	Diseño presentación
147	Presentación

2.7 Evaluación de riesgos

2.7.1 Lista de riesgos

1. Incumplimiento de plazos de entrega: Alguno de los plazos establecidos no se cumple.
2. Herramientas inadecuadas: problemas con las herramientas de desarrollo.
3. Incumplimiento de alguna norma: Repercusiones legales por el incumplimiento de alguna norma.
4. Abandono del proyecto de algún miembro: Uno de los miembros decide abandonar el proyecto.
5. Implementación incorrecta: Problemas a la hora de modificar MATE.

2.7.2 Catalogación de riesgos

Riesgo	Probabilidad	Impacto
1	Media	Marginal
2	Baja	Marginal
3	Baja	Crítico
4	Baja	Crítico
5	Baja	Marginal

2.7.3 Plan de contingencia

Riesgo	Plan de contingencia
1	Reuniones de seguimiento.
2	Planificación inicial.
3	Cotejar acciones futuras con las normativas presentes en el documento de viabilidad. El NDA hace...
4	Reorganizar planificación.
5	Consultar expertos, reunión de seguimiento.

2.8 Presupuesto

2.8.1 Estimación coste de personal

Recurso	Coste
3 Analista-programador	64.500 €
1 Project Manager	28.000 €
Total	**92.500 €**

2.8.2 Estimación coste de los recursos

Recurso	Coste
Servidor	2.220,76 €
8 nodos cluster	23.902,08 €
SAN	8.173,86 €
Otros	12.917,21 €
Software	0 €
Total	**47.213,91 €**

2.8.3Estimación coste de las actividades

Ninguna actividad tiene costes directos.

2.8.4 Estimación de otros costes

Recurso	Coste
Personal de soporte	24.000€
Alquiler local	25.488€
Total	**49.488 €**

2.8.5 Estimación costes indirectos

Recurso	Coste
Electricidad	5.389,65 €
Consumibles	1.475 €
Telefonía	708 €
Limpieza	4.141,8 €
Mantenimiento	1.062 €
Gestión	2.124 €
Total	**14.900,45 €**

2.8.6 Resumen y análisis coste beneficio

Coste total = 92.500 + 47.213,91 + 49.488 + 14.900,45 = 204.102,36 €

Estos costes, pese a que parecen altos, son teóricos y en la realidad el coste del proyecto es mas reducido dado que el personal somos nosotros, y como parte del proyecto realizados las tareas de desarrollo. Además el *cluster* construido será usado para otras tareas y por tanto su coste se amortiza. Por último los beneficios que comporta nuestro proyecto, aunque no económicamente, compensan por la inversión.

3. Calidad del software

En esta sección se define en que términos hablamos de calidad de software y se describen varios modelos que nos ayudan a asegurar la calidad de un producto. Finalmente escogeremos el mas adecuado a nuestro proyecto que será el que apliquemos.

3.1 SQA

Un aspecto importante dentro del proyecto es la utilización de SQA, del inglés Software Quality Assurance. Se trata de un modelo sistemático y planeado de todas las acciones necesarias para asegurar la calidad esperada del producto final, así como la correcta aplicación de estándares y procedimientos adoptados. Es algo que se aplica durante todo el proceso de desarrollo, y se rige por el SQAP (Software Quality Assurance Plan), dónde se definen las actividades específicas a llevar a cabo dentro del proyecto. Dado el proceso de desarrollo de MATE, que se espera que continué más allá del proyecto actual, es necesario establecer una serie de métodos que consoliden los objetivos marcados en cada etapa.

Según los modelos de ampliación de defectos, el coste de los fallos detectados en un producto software es mayor cuanto más tarde detecta. Para ilustrar la reducción del coste con la detección anticipada de errores, podemos considerar una serie de costes relativos que se basan en datos de proyectos de software rea[1]les. Suponiendo que un error descubierto en la fase de diseño del producto cuesta 1,0 unidad monetaria, este mismo error descubierto antes de realizar el proceso de test costará 6,5 unidades, durante las pruebas 15 unidades, y después de la entrega entre 60 y 100 unidades. El mismo razonamiento se puede aplicar a otros recursos de un proyecto como pueden ser tiempo o rendimiento. Es aquí donde reside la importancia de un buen proceso de SQA ya que nos permite tener un buen seguimiento durante todo el proyecto.

A la hora de desarrollar un SQAP dentro de un proyecto hay diferentes alternativas y modelos de los que escoger. Todos ellos listan una serie de aspectos importantes a tener en cuenta en el momento de evaluar la calidad del software. Una vez detectados estas cualidades críticas, se pueden cuantificar con una serie de métricas y así poder determinar la calidad actual del producto.

1 Defect amplification model [IBM81] "Implementating Software Inspections", Notas del curso, IBM Systems Sciences Institute, IBM Corporation, 1981

3.1.1 Modelos estándar

Por un lado podemos encontrar diferentes modelos estándar. Estos modelos son aplicables a cualquier proyecto y determinan el nivel de calidad con atributos generales. Además han sido utilizados en diferentes proyectos y por lo tanto se tiene una perspectiva de los resultados esperados.

Uno de los primeros modelos existentes y en el que se basan la mayoría de los actuales es el modelo de McCall, que, en un principio, fue creado para las fuerzas aéreas de los Estados Unidos en 1977. Principalmente está enfocado a los desarrolladores del sistema y al proceso de desarrollo. En este modelo McCall intenta unir la perspectiva de usuarios y desarrolladores, centrándose en unas características de la calidad del software que reflejan tanto la visión de los usuarios como las prioridades de los desarrolladores. El modelo presenta tres características para medir la calidad del software: revisión (habilidad para adoptar cambios), transición (habilidad para adaptarse a otros entornos) y operación (sus características operativas).

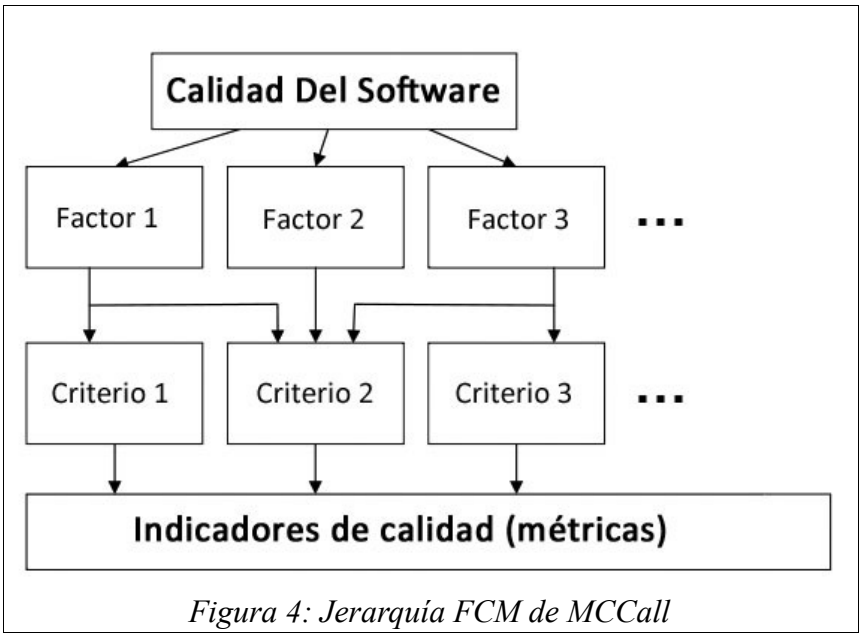


Figura 4: Jerarquía FCM de MCCall

El modelo también es llamado FCM (Factor, Metrics, Criteria) porque detalla tres tipos de características en una jerarquía (figura 1) dónde las más importantes se denominan factores. Por debajo podemos observar diferentes subcaracterísticas denominadas criterios y, finalmente, tenemos las métricas para determinar el nivel de satisfacción de cada una de estas subcaracterísticas.

Otro modelo a mencionar es el FMEA (Failure Mode and Effects Analisis). Como su nombre indica se basa en analizar problemas potenciales, principalmente en una época temprana del ciclo de desarrollo donde es más fácil tomar acciones para solucionarlos. FMEA se utiliza para identificar fallos potenciales en los sistemas, para determinar su efecto sobre la operación del producto, y para identificar acciones correctivas para atenuar las faltas. Podemos encontrar diferentes tipos siguiendo este modelo según su enfoque: Sistema (enfocado a funciones globales del sistema), Diseño (enfocado a componentes y

subsistemas), Proceso (enfocado a procesos de fabricación y ensamblamiento), Servicio (enfocado a funciones del servicio) y Software (enfocado a funciones del software).

Para terminar, los modelos estándar más seguidos son las normas ISO (International Organization for Standardization). Esta organización ha desarrollado una serie de normas y modelos para la evaluación de la calidad aplicables a productos generales, adaptándose en ciertos casos a la producción de software. En este modelo los conceptos de calidad se aplican más en el producto terminado que en el proceso de desarrollo. Estas normas hacen posible que se sigan patrones de calidad generalmente aceptados con los que se logran métricas para determinar las cualidades de un producto, teniendo en cuenta que en la práctica existen dos tipos de calidad: externa (referente a los clientes) e interna (referente a funcionalidad del software).

De todos los estándares que presenta, el más interesante para el proyecto es el ISO 9126, que está enfocado directamente a productos software. Este modelo está basado en el de McCall, ya que determina la calidad del software en base a una herencia de características. Como podemos observar en la figura 2, posee 6 características principales (factores siguiendo el modelo de McCall). Las 6 características principales son funcionalidad, fiabilidad, usabilidad, eficiencia, Mantenibilidad y portabilidad. Además de esto cada aspecto tiene diferentes subcaracterísticas a evaluar.

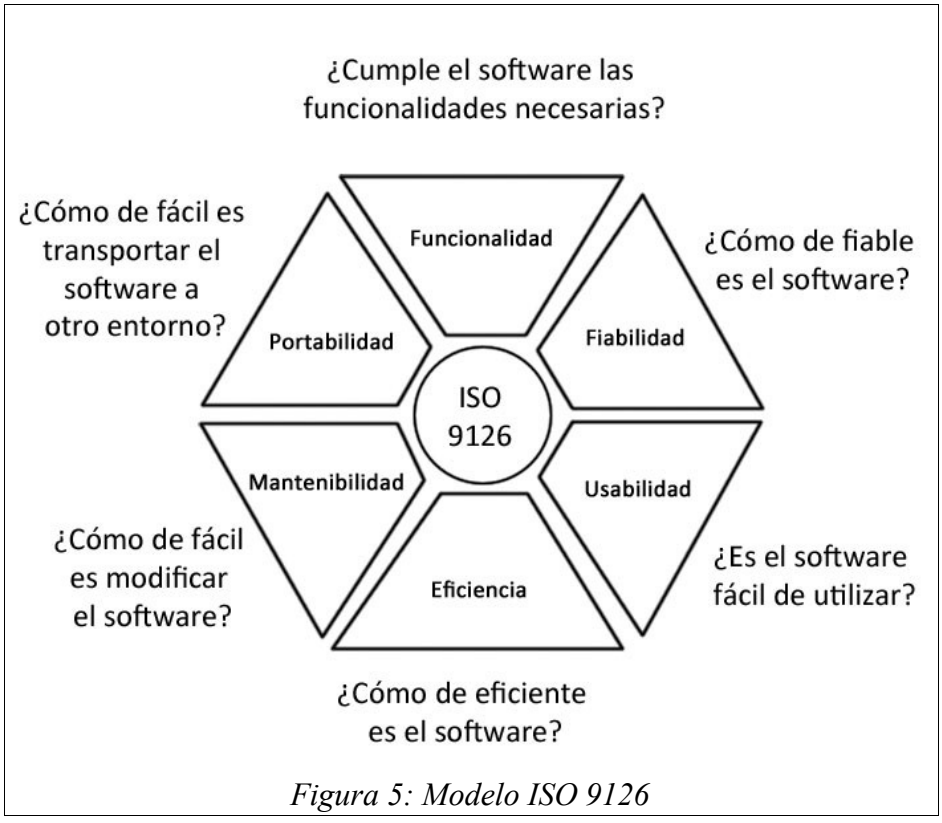


Figura 5: Modelo ISO 9126

Dentro del proyecto se da énfasis a los aspectos más importantes para el software. Por una parte dentro de fiabilidad es importante tener en cuenta la tolerancia a fallos y la facilidad de recuperación, en el

proyecto cubre este aspecto añadiendo ciertas nuevas características al programa (capítulo 4.5) y realizando una serie de tests (capítulo 5). También se ha trabajado en el factor de usabilidad, concretamente en las subcaracterísticas de comprensibilidad y facilidad de aprendizaje. Esto se ha mejorado con la documentación generada para MATE y los diferentes elementos de soporte. El tema de la eficiencia es algo importante, tratándose de un software de aumento de rendimiento, y es algo que se ha acabado de definir consolidando todos los aspectos del programa en su estado previo. Finalmente el hecho de haber creado la metodología con diferentes guías y directrices de trabajo hace que el mantenimiento del software en el futuro sea mucho mayor y facilita el análisis y test del producto.

3.1.2 Modelos de aplicación o específicos de compañías

Además de los modelos estándar vistos en el capítulo anterior, existen los de aplicación específicos para compañías. Estos no hablan de características generales sino que detallan controles de calidad ajustables a necesidades particulares.

Uno de los que vale la pena mencionar es el llamado modelo CMMI (Capability Maturity Model Integration). Este modelo determina una serie de procesos y actividades a llevar a cabo para asegurar la calidad del producto, sin entrar en detalles de cómo realizar estas actividades. Según se van completando las actividades propuestas se dice que el proyecto está en diferentes niveles de madurez: incompleto (nivel 0), inicial, administrado, definido, cuantitativamente administrado y optimización (nivel 5). Dispone de una lista de áreas de acción, y en cada una de ellas se proporciona la lista de prácticas a llevar a cabo. Actualmente incluye de 22 áreas de proceso[2] que podemos observar en la figura 3.

2CMMI for Acquisition versión 1.2, CMMI for Developement versión 1.2 y CMMI for Services versión 1.2

CMMI® for Development, Version 1.2 (CMMI-DEV, V1.2)		
Causal Analysis and Resolution (CAR)	Configuration Management (CM)	Decision Analysis and Resolution (DAR)
Integrated Project Management +IPPD (IPM+IPPD)	Measurement and Analysis (MA)	Organizational Innovation and Deployment (OID)
Organizational Process Definition +IPPD (OPD+IPPD)	Organizational Process Focus (OPF)	Organizational Process Performance (OPP)
Organizational Training (OT)	Product Integration (PI)	Project Monitoring and Control (PMC)
Project Planning (PP)	Process and Product Quality Assurance (PPQA)	Quantitative Project Management (QPM)
Requirements Development (RD)	Requirements Management (REQM)	Risk Management (RSKM)
Supplier Agreement Management (SAM)	Technical Solution (TS)	Validation (VAL)
Verification (VER)		

Tabla 2: Areas de proceso del modelo CMMI, versión 1.2

El último modelo a comentar es el modelo GQM (Goal-Question-Metric). A diferencia de los modelos vistos anteriormente, este modelo no se basa en características generales de los productos, sino que es aplicable solamente al proyecto en concreto. Toma este enfoque con la idea de que un programa de mesura de calidad puede dar mejores resultados si se diseña con las metas en mente. Simplemente se trata de trabajar realizando tres pasos: crear una lista con las metas del proyecto, a partir de la lista generar unas preguntas que determinen si la meta se ha cumplido y finalmente decidir qué atributos hacen falta medir para responder estas preguntas.

4 Especificación de la metodología

En esta sección se describen los diferentes componentes de la metodología que ha sido desarrollada para MATE, se especifica la partición de trabajo entre los diferentes miembros del equipo de trabajo y se exponen los resultados obtenidos describiendo tanto las guías creadas para el entorno como los documentos de especificación.

4.2 Especificación del entorno de desarrollo

4.2.1 Perspectiva general

Como se ha mencionado anteriormente este proyecto tiene como objetivo primario establecer una metodología de trabajo para dar continuidad a MATE. Una buena metodología es crucial para que un proyecto de desarrollo de software sea exitoso y dada la naturaleza de MATE entendemos que se puede beneficiar mucho de la creación de una.

Hemos diseñado una metodología adaptada a MATE, que dotara a los futuros desarrolladores de una

base para continuar con el desarrollo de esta aplicación. Como parte de esta metodología nosotros proponemos una serie de herramientas para ser usadas como entorno de trabajo por los desarrolladores. Esto es importante porque trabajar en un entorno especifico y especialmente diseñado para la tarea en cuestión aumenta la productividad de los trabajadores y reduce el tiempo de trabajo. Además de producir un producto homogéneo y de calidad como resultado.

4.2.2 Contribución personal

Este es un proyecto conjunto en el que se da una división de trabajo entre los tres estudiantes que formamos parte de este. La tarea que hemos realizado es de gran envergadura y por tanto cada miembro del equipo se ha encargado de un aspecto distinto del proyecto, ya sea en el diseño y desarrollo de la metodología como en la aplicación de esta. Esta división de trabajo se muestra en la tabla siguiente.

Miembro	Tareas asignadas
Rodrigo Echeverría	• Instalación de Redmine • Guía de instalación y configuración de Redmine • Integración de Redmine con Apache • Integración de Redmine con Eclipse • Script de instalación de Redmine
Toni Pimenta	• Instalación de Apache • Instalación de SVN • Instalación de Doxygen • Integración de Apache y SVN • Guía de instalación y configuración de Apache y SVN • Integración de SVN y Redmine • Script de instalación de SVN • Script de instalación de Doxygen
Noel De Martin	• Instalación de Buildbot • Guía de instalación y configuración de Buildbot • Integración de SVN y Buildbot • Script de instalación de Buildbot

Tabla 3: División de tareas relativas al entorno de trabajo

A continuación se especifican los elementos con los que se ha contribuido a nivel personal a la creación del entorno de desarrollo.

4.2.3 SVN

SVN (Subversion[3]) es una herramienta de control de versiones y repositorio. Se trata de una herramienta imprescindible en cualquier proyecto de desarrollo, especialmente cuando existen varios desarrolladores trabajando simultáneamente en el mismo proyecto. Un VCS (Version Control System) como SVN

3Subversion (sistema de control de versiones) Página oficial: http://subversion.apache.org/

permite el acceso a los archivos del proyecto por parte de diferentes entidades de forma que se eviten conflictos y la consiguiente perdida de información que puede resultar en baja productividad o incluso dañar críticamente el proyecto. En nuestro caso se contempla la posibilidad de que varios programadores trabajen sobre el código o documentos de MATE de forma simultanea y por tanto se implanta un repositorio SVN como solución a los conflictos.

SVN permite el almacenamiento de archivos en un entorno seguro y controlado donde los cambios realizados se pueden organizar en versiones y modificaciones en los archivos pueden ser aplicadas al repositorio sin peligro de que se pierdan modificaciones realizadas por otros desarrolladores. Esto se hace mediante un proceso de merging que conserva los cambios realizados por ambos desarrolladores si estos son compatibles y proporciona la posibilidad de resolver colisiones entre estos.

Con SVN los desarrolladores siempre disponen de las últimas modificaciones realizadas por sus compañeros a la vez que pueden revisar antiguas versiones o recuperar el estado de un archivo. Esto es de vital importancia cuando existen dependencias entre el trabajo realizado por varios desarrolladores ya que se evitan periodos de poca productividad mientras se espera a obtener los cambios realizados por el resto del equipo.

En nuestro caso usaremos SVN integrado con el resto del entorno de modo que aumentamos su versatilidad y integramos sus funciones en el entorno general.

A nivel personal dotamos a SVN de la capacidad de proporcionar acceso web al repositorio. Esto se lleva a cabo integrando SVN con Apache, un servidor gratuito, que nos permite acceder a los archivos del repositorio SVN a través de internet, mediante una interfaz web. Además aplicaremos un protocolo SSL para que el acceso de los archivos sea seguro y que datos sensibles estén protegidos.

La guía y script creados para este proyecto corresponden a la versión 1.6.15 de SVN (1.6.12 para Apache).

Funcionalidades

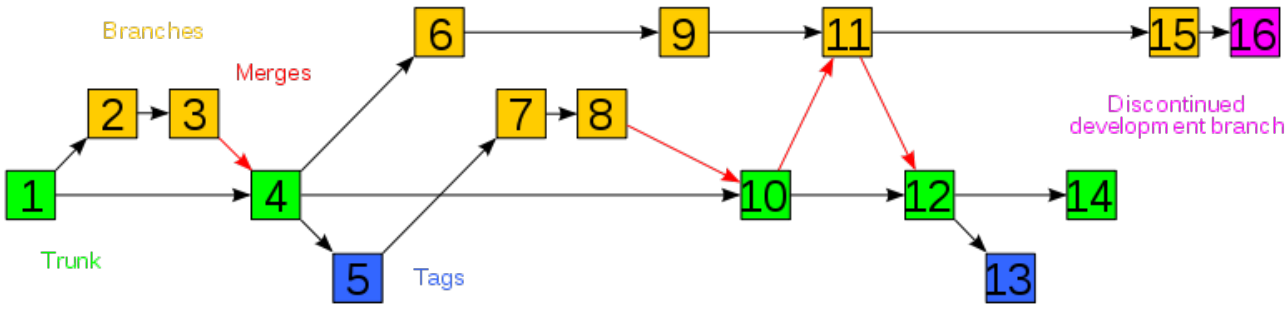


Figura 6 Representación de la vida de un proyecto en SVN.

Algunas de las principales funcionalidades que nos ofrece SVN son las básicas de cualquier VCS, como unir dos ramas de desarrollo resolviendo conflictos entre ellas (merging).

En esta imagen podemos observar las funciones básicas de SVN. La nomenclatura está especificada en la guía de control de versiones y *build*. Podemos observar que la rama principal (Trunk) es la central de color verde y contiene el estado actual del proyecto. Las ramas amarillas, llamadas *Branches ,* representan copias del código que se están modificando. Vemos que cuando una de estas ramas de desarrollo finaliza, se aplican los cambios a la rama principal mediante el proceso de *merge*.

Este proceso aplica los cambios realizados en la *branch* que sean compatibles con la rama principal y trata de resolver los conflictos que se den para poder mezclar las dos ramas completamente. A través de este procedimiento obtenemos en el punto 4 la versión base con los cambios de la rama 2-3 aplicados. En este punto generamos otra rama de desarrollo (6) y generamos una versión numerada (Tag). Las versiones numeradas son normalmente versiones estables que se desea liberar. Vemos mas adelante que las ramas no tienen porque mezclarse siempre con el *trunk* sino que pueden ser anuladas o pueden mezclarse dos *branches.*

En los estados 10 11 y 12 vemos un procedimiento habitual, llamado *Rebasing*, que consiste en hacer un *merge* entre la rama de desarrollo y el código principal pero de modo que el resultado de este quede en la *branch*, de modo que cualquier modificación que se haya realizado en la rama principal sea añadida a la *branch* antes de hacer la mezcla con *trunk*. Esto mantiene actualizada la rama a la vez que aleja el procedimiento de *merge* de la rama principal.

Subversión nos proporciona algunas *features* extra, como:

- SVN versiona directorios: los directorios en los que se almacenan los archivos están versionados lo que nos permite llevar un control preciso de los cambios en estos.
- *Commits* atómicos: la instrucción *commit* que nos permite aplicar las modificaciones locales al repositorio se ejecuta de forma atómica, como si se tratara de una única instrucción, para evitar modificaciones parciales que corromperían el repositorio.

Además de permitir la modificación y llevar un control de los archivos SVN no sirve como herramienta de seguimiento. Subversion almacena los cambios que se producen en los archivos en lugar de los archivos en si, y nos permite averiguar cuando se modificó cada uno de ellos y quien fue el responsable del cambio.Otras herramientas útiles son *diff* y *revert* que nos permiten respectivamente, ver las diferencias entre dos ficheros y deshacer cambios en algún archivo.

Distribución de archivos

Como resultado de usar SVN obtenemos un repositorio ordenado con todos nuestros archivos ordenados por versiones y un historial de cambios realizados.

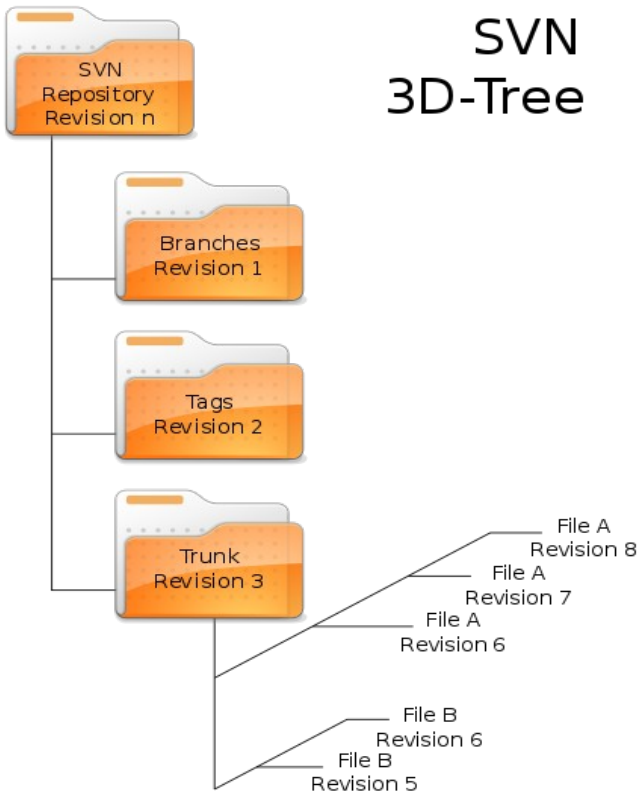


Figura 7: Esquema de distribuciñon de archivos en el repositorio SVN

En esta figura podemos ver un ejemplo de como quedaría el repositorio, vemos que cada carpeta tiene su versión pero que a su vez los archivos dentro de estas pueden haber sido revisados mas veces.

Guía de instalación y configuración

Como documento adjunto al proyecto, añadimos las guías de instalación de los diferentes componentes del entorno, para que se pueda ser reproducido por el equipo de desarrollo. La guía de instalación y configuración de SVN y apache, incluye los datos necesarios para instalar y configurar un repositorio SVN bajo un servidor apache con SSL. Y incluye una pequeña muestra de como crear un repositorio y mantenerlo.

El documento contiene una breve descripción de la aplicación seguida de los requisitos y dependencias de esta. A continuación se describe el proceso de instalación, se ofrecen distintas alternativas para instalar la aplicación. Una vez explicada la instalación de SVN se describe un método para integrarlo con Apache y configurar el repositorio. Finalmente explicamos como configurar y usar el programa

Script de instalación

Para facilitar la tarea de los desarrolladores se ha diseñado un script para realizar de forma automática la instalación de SVN y apache, aunque la configuración del servidor requiere de algunos datos del usuario.

El *script* es simple y consiste de una serie de instrucciones que descargan la versión indicada de la aplicación y la instalan en el sistema.

El uso del *script* requiere acceso a internet y permiso de superusuario para realizar las instalaciones. El *script* debe ser ejecutado por un interprete *bash*.

4.2.4 Doxygen

Documentar el código es necesario para que futuros desarrolladores, y nosotros mismos después de un tiempo sin trabajar en MATE podamos entender fácil y rápidamente el código. Normalmente el código no es trivial y cuesta comprender cual es la función de cada elemento y como se relacionan entre ellos y estos es necesario para poder participar en el desarrollo del proyecto. El proceso de generar una documentación detallada del código es pesado y lleva tiempo y normalmente no se le da prioridad en proyectos de orientación académica como MATE.

Doxygen[4] es una herramienta que nos permite extraer automáticamente documentación del código. Mediante unos comentarios especiales Doxygen nos permite describir los elementos del programa en el código como si se tratase de comentarios *inline*. Estos comentarios serán usados por Doxygen junto con la información que este extrae del código en si para generar una completa documentación con diagramas y descripciones de los elementos que componen la aplicación.

La guía y script creados para este proyecto corresponde a la versión 1.7.3 de Doxygen que es la actual en este momento.

Funcionalidades

Doxygen puede generar, a partir del código, documentación en forma de pagina web (on-line) o manuales de referencia en formatos como pdf. No solo clasifica y organiza los elementos de la aplicación sinó que genera una serie de diagramas que muestran la jerarquía entre estos, y proporciona la posibilidad que crear una interfaz web completa y intuitiva con toda la información.

Doxygen se adapta al lenguaje de programación elegido y extrae información directamente del código, como el tipo de cada variable o los métodos de cada clase. Se puede configurar mediante un archivo de configuración llamado *doxyfile* (por defecto) que contiene una amplia gama de opciones que nos permiten modificar su comportamiento y el resultado obtenido.

Doxygen además cuenta con una interfaz gráfica de configuración y incluso se integra con eclipse, nuestro IDE, mediante un *plug-in* llamado Eclox[5] que nos permite gestionar los doxyfiles de cada proyecto y modificar la configuración de cada uno.

4Pagina oficial de Doxygen: http://www.stack.nl/~dimitri/doxygen/
5 Eclox, plugin para eclipse. Pagina oficial: http://home.gna.org/eclox/

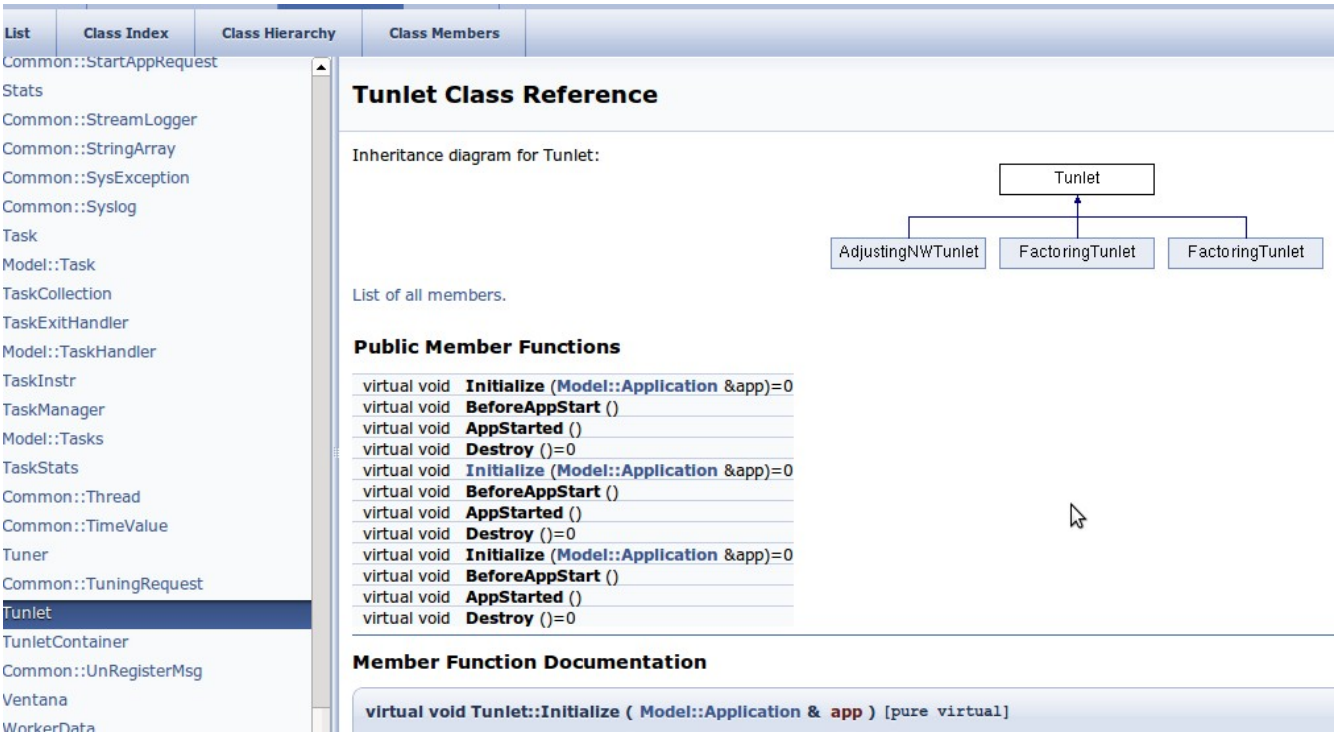


Figura 8: Ejemplo de HTML resultante de extraer documentación con Doxygen

La especificación de como se deben escribir los comentarios de código se encuentra en la guía de documentación de código.

Guía de instalación y configuración

Como en el caso de SVN, para Doxygen también se ha creado una guía de instalación que se incluirá como documento adjunto, esta incluye los datos específicos de la aplicación así como simples instrucciones de uso y configuración.

Contiene:

- La especificación de los requisitos y dependencia de Doxygen.
- Guía de instalación (dos alternativas para instalar Doxygen).
- Configuración de la aplicación (doxyfile).
- Guía de uso de la aplicación.

Script de instalación

Para facilitar la tarea de reproducir el entorno de trabajo hemos creado un script automático que instala Doxygen y sus dependencias.

Este script descarga el código fuente de la web oficial la versión 1.7.3 de Doxygen y instala los ficheros en el directorio definido para el proyecto (/opt/MATE/). A continuación genera *links* simbólicos hacia /usr/local/bin/ para que la aplicación se pueda ejecutar desde linea de comandos.

Uso del *script*:

Solo se requiere conexión a internet y ejecutar el *script* con un interprete de *bash*.

4.3 Documentos de especificación

4.3.1 Perspectiva general

Una vez hemos diseñado el entorno de trabajo que usaremos en el proyecto, es importante establecer una metodología de trabajo común. Esto se lleva a cabo mediante la creación de unas serie de documentos que especifican aspectos concretos de la metodología de trabajo a seguir por todos los miembros del equipo de desarrollo. Esto es necesario para lograr una homogeneidad que, además de simplificar el trabajo en equipo, ofrece un resultado de calidad y la aplicación de una metodología especifica resulta en un producto mas compresible y facilita el trabajo realizado por terceras partes.

Estos documentos especifican diferentes aspectos de la forma de trabajar que se debe aplicar a MATE. Desde como se debe escribir la documentación de la aplicación hasta el proceso de construcción y test de esta.

4.3.2 Contribución personal

A nivel personal se ha contribuido a la generación de esta serie de documentos que, juntos, componen la metodología de trabajo establecida para MATE. De nuevo, hacemos hincapié en la envergadura de esta tarea, y entendemos que esto justifica la división de trabajo entre los componentes del grupo.

La división del trabajo se realiza según especifica la tabla siguiente:

Miembro	Guías confeccionadas
Noel De Martin	• Guía de estilo de codificación • Guía de estilo de control de versiones y construcción (construcción)
Rodrigo Echeverría	• Guía de estilo documentación • Guía de estilo de documentación de código
Toni Pimenta	• Guía de estilo de desplegamiento • Guía de estilo de control de versiones y construcción (control de versiones)

Tabla 4: División de trabajo relativa a los documentos de especificación

A continuación se describen los documentos desarrollados a nivel personal, que son los que especifican el proceso de control de versiones y construcción *(build)* de MATE y el de *deployment* de la aplicación.

4.3.3 Especificación de control de versiones y build

En proyectos de gran complejidad es necesario llevar un estricto control del trabajo realizado, especialmente si se trabaja en grupo, y suele ser necesario automatizar algunas tareas. MATE es una aplicación compleja y como consecuencia está formada por muchos archivos, que deben tratados meticulosamente ya que cualquier modificación no deseada podría influir en el funcionamiento de la aplicación.

Por este motivo trabajamos con una herramienta de control de versiones (SVN) y aplicamos una metodología adecuada para que las modificaciones hechas en el programa no tengan efectos colaterales no deseados. Esto es, una serie de pasos a seguir para asegurarnos de que existe una sincronización entre los miembros del equipo y se trabaja con la máxima eficiencia.

MATE está compuesto por varios módulos y cada uno de ellos se divide en clases que están codificadas en diferentes archivos (código fuente). La distribución de clases en diferentes archivos, en lugar de trabajar con un único fichero de código, tiene obvios beneficios pero como consecuencia de esto obtenemos muchos componentes donde pueden existir errores. Por otra parte MATE es un aplicación multiplataforma, lo que requiere que funcione bajo diferentes configuraciones y su funcionamiento debe ser coherente en ellas. Para comprobar que MATE satisface estos requisitos existen tests que se pueden realizar una vez aplicados los cambios. El problema es que trabajando con tantos archivos la tarea de compilarlos, enlazarlos, ejecutarlos y testearlos en varias plataformas se hace muy pesada.

Para esto usamos una herramienta que automatiza este proceso, llamada Buildbot. Esta herramienta requiere de una configuración y un proceso especifico de funcionamiento que también se describe en el documento.

Contenido

Este documento se compone de dos secciones, la primera de ellas describe el procedimiento de control de versiones que se aplicará a MATE y especifica un modelo a seguir para que todos los componentes del equipo trabajen de forma sincronizada.

En esta sección se definen las herramientas que serán utilizadas para realizar dichas tareas y se especifica de que modo se configurará esta para obtener el resultado deseado.

Por ultimo esta sección proporciona una propuesta de dinámica de trabajo que los miembros del equipo de desarrollo deberán seguir para que el proyecto evolucione de forma adecuada.

La segunda sección especifica el funcionamiento de la herramienta de Construcción automática que usaremos para compilar y testear MATE a medida que se hacen cambios importantes en la aplicación.

Está sección empieza como la anterior determinando cual será la herramienta usada para este propósito, en este caso Buildbot. A continuación se describen los componentes principales de la herramienta y se explica cual es la función de cada uno dentro del programa en general.

Por ultimo se propone una dinámica de trabajo para este proceso.

4.3.4 Especificación de deployment

El *deployment* de un producto software son el seguido de procesos y tareas que se requieren para que el programa este disponible para su uso, empezando por la distribución de la aplicación a los usuarios una vez desarrollado el software. Entre estas tareas se encuentra la instalación de la aplicación, su activación, actualización y otros procesos que han de darse durante la vida del producto.

MATE, dado que se trata de un proyecto no comercial, no dispone de todos estos procesos, pero hemos intentado que los mas relevantes estés integrados en la aplicación final. Uno de los módulos esenciales es el de instalación, para que el programa sea utilizable por los usuarios debe estar instalado y el hecho de automatizar este proceso aumenta el alcance de la aplicación ya que facilita su implantación.

En el proceso de *deployment* toman parte, no solo la aplicación que se desea utilizar, si no que participan algunos programas externos que asisten en los procesos necesarios. Por tanto es necesario especificar cuales son estos programas y de que se encargan.

Contenido

En el caso de MATE el *deployment* básicamente se reduce al proceso de instalación dada su orientación en este momento del desarrollo.

Este documento se compone de 4 secciones relativas al *deployment* de MATE. En la primera sección se definen las fases que se deberían considerar a la hora de preparar la aplicación para los usuarios. Algunas de estos procesos no están implementados en MATE pero se definen por tal de establecer el estándar para futuro desarrollo de la aplicación.

A continuación se especifican las herramientas usadas en el desarrollo de los módulos que si que se han generado para MATE. Estos son, no solo las herramientas externas que se usan sino también los *scripts* que puedan asistir en el proceso.

En la siguiente sección se determinan los diferentes componentes que se obtienen como resultado del *deployment*.
Como ultima sección se determinan los responsables de crear cada uno de los módulos necesarios. De forma que se especifica, dentro del equipo de desarrollo, los componentes que se encargarán de producir cada elemento.

5 Aplicación de la metodología

Esta sección explica detalladamente el proceso de aplicar la metodología previamente creada a MATE. Se empieza especificando sobre que parte de MATE se han aplicado los cambios y se detalla el trabajo realizado en el.

5.1 MATE: el módulo AC

A nivel personal en este proyecto se ha aplicado la metodología desarrollada al módulo AC de MATE.

El funcionamiento y la lógica del módulo AC se ha definido anteriormente, en esta sección describimos con detalle cuales son sus componentes para tener claro sobre que parte de MATE se ha trabajado.

Este módulo se construye a partir de los siguientes ficheros de código:

Header (.h)	Implementación (.cpp)	Clases que contiene
cmdLine.h	-	CommandLine
ctrl.h	Ctrl.cpp	Controller
InstrSet.h	InstrSet.cpp	SnippetHandler, InstrGroup
-	Main.cpp *	-
Monitor.h	Monitor.cpp	Monitor
PTPAcceptor.h	PTPAcceptor.cpp	PTPAcceptor (EventHandler)
PTPHandler.h	PTPHandler.cpp	PTPHandler (EventHandler)
ShutDownSlave.h	ShutDownSlave.cpp	ShutDownSlave (ActiveObject)
SnippetMaker.h	SnippetMaker.cpp	SnippetMaker
Task.h	Task.cpp	Task
TaskInstr.h	TaskInstr.cpp	TaskIntr
TaskManager.h	TaskManager.cpp	TaskExitHandler, TaskManager
Tasks.h	-	TaskCollection
Tuner.h	Tuner.cpp	Tuner

Tabla 5: Ficheros de código del AC y clases que contienen

*El fichero Main.cpp contiene la función principal por la que la aplicación inicia su ejecución. Esta archivo no tiene una pareja de *header* ya que no requiere de definiciones previas, simplemente se trata de iniciar el programa y recoger las posibles excepciones que se hayan podido producir.

Además de estos ficheros que son usados exclusivamente por el módulo AC, este requiere de otras clases implementadas en el conjunto Common. Common es el nombre de un paquete de ficheros codificados en C++ que contienen la implementación de clases comunes para varios módulos.

El AC obtiene estos ficheros ya compilados y se ocupa de incluirlos en el enlace del binario final.

Los ficheros objeto de Common requeridos por el AC son los siguientes:

DateTime.o Address.o di.o

TimeValue.o Socket.o ConfigReader.o

Exception.o SysException.o PTPProtocol.o Utils.o

Reactor.o PTPMsg.o Env.o

Process.o PTPMsgHeader.o, Syslog.o Paths.o

Config.o	ConfigException.o	Thread.o
ConfigMap.o	ECPMsg.o	
ByteStream.o	ActiveObject.o	

Estos ficheros objeto contienen las definiciones de clases y sus respectivos métodos, que AC utiliza junto con los otros módulos para representar elementos comunes como Eventos o Fechas.

Mediante los ficheros compilados del AC y los que se importan de Common se crea el binario final que corresponde al módulo AC.

Una vez compilado y enlazado el ejecutable obtenido realiza las funciones descritas por el siguiente diagrama:

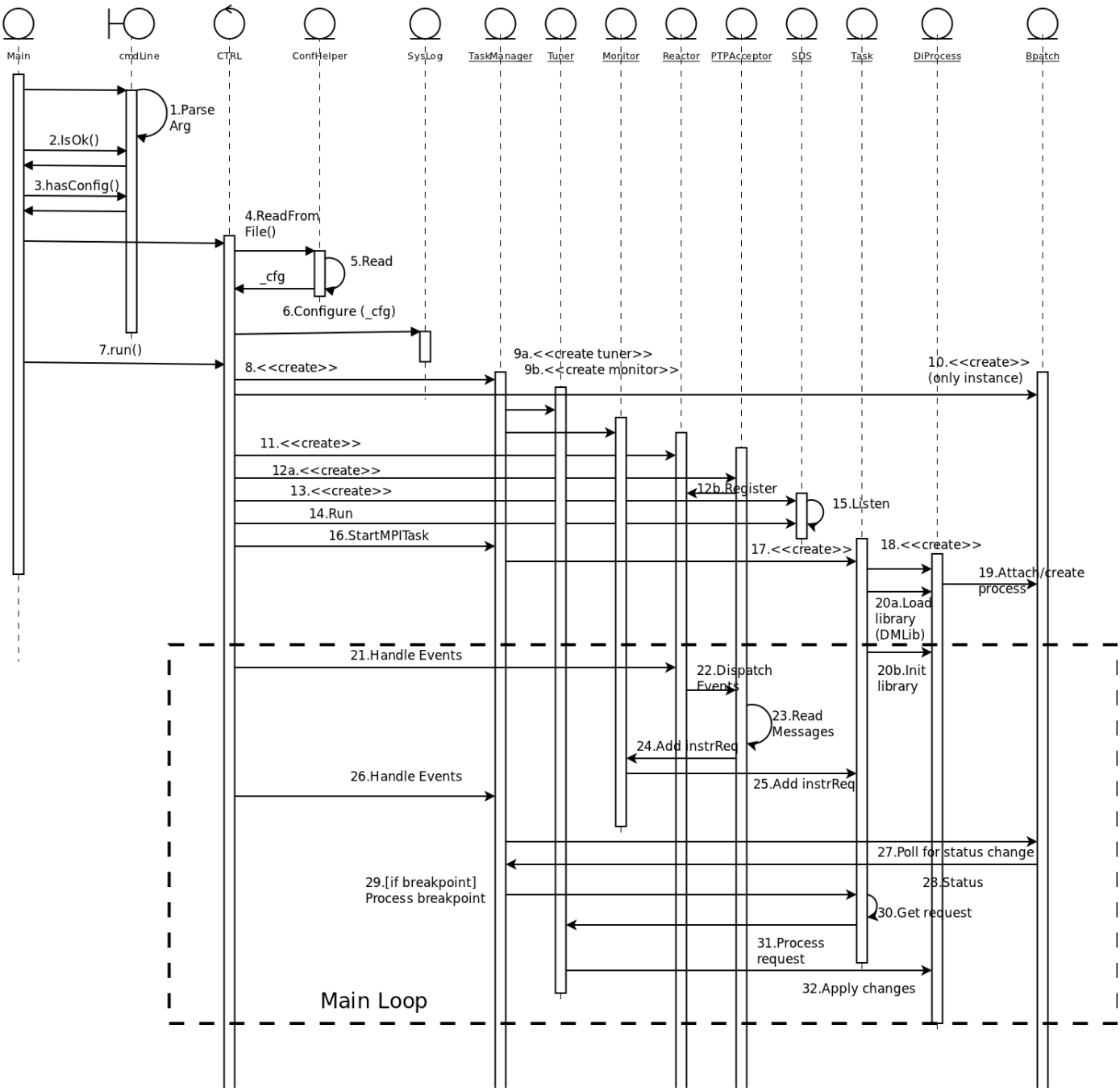


Figura 9: Diagrama de secuencia del módulo AC.*

*Se adjunta como anexo una imagen de alta resolución de este diagrama.

5.2 Estado inicial del módulo

MATE, antes del inicio de este proyecto, era una aplicación totalmente funcional, que se ejecutaba correctamente. Debido a su naturaleza de proyecto académico, varios programadores habían introducido cambios en el código pero la documentación de estos, era escasa o inexistente. Incluso el código inicial no dispone de una documentación o manual de usuario formales.

Además de no estar documentado, el código contenía muchos fragmentos comentados, que habían sido excluidos de esta forma del proceso de compilación y substituidos por nuevas secuencias de código.

El proyecto data del año 2003 y por tanto muchas de sus funciones están des-actualizadas (*deprecated*). Esta des-actualización no solo afecta a las funciones de C++ sino que las librerías usadas como Dyninst de terceras partes también han sido substituidas por versiones mas modernas y por tanto algunas de sus funciones han cambiado o dejado de existir.

Es esencial solucionar estos problemas porque pueden afectar al funcionamiento del programa, pero hay otros que también suponen un riesgo aunque este sea posible a mas largo alcance.

En el futuro MATE será estudiado y posiblemente actualizado por otros desarrolladores y por tanto es importante que el código esté escrito de forma homogénea por tal de facilitar su comprensión.

El código carece de comentarios explicativos de las clases y los métodos, y aunque si contiene algunos comentarios *inline* que describen las secuencias de instrucciones as complejas el código en general es difícil de comprender.

5.3 Cambios propuestos

Como solución a los problemas nombrados se ha creado una metodología para trabajar con proyectos, específicamente orientada a MATE. Esta metodología, definida con anterioridad en este documento, describe una serie de procesos para dotar al proyecto MATE de un código homogéneo y formal, con comentarios explicativos y actualizado a las ultimas versiones de sus dependencias. Además del trabajo realizado sobre el código fuente también se propone la creación de una documentación y un manual de usuario para la aplicación MATE.

Esta metodología se traduce en una serie de cambios que junto con los otros dos miembros del equipo aplicaremos a los diferentes módulos de MATE.

En primera instancia se intentaran eliminar las llamadas a funciones des-actualizadas, y sustituiremos las librerías usadas por su versión actual.

A continuación, procedemos a repasar los archivos fuente y las dependencias entre ellos, resolviendo redundancias y gestionando los ficheros de forma que estén organizados por modulo, y haciendo que los ficheros comunes solo aparezcan una vez.

Una vez contamos con el conjunto de ficheros final procedemos a aplicar las guías de estilo al código de modo que todo quede homogéneo y fácil de leer.

A continuación, una vez comprendemos bien el código, insertamos los comentarios de cada método en los archivos de cabecera (*header*) y los comentarios *inline* necesarios en los archivos de implementación.

Después de cada paso es conveniente recompilar para comprobar que se mantiene la funcionalidad inicial.

Una vez realizados estos cambios el código debería ser mucho mas comprensible y ahora gracias a los comentarios de Doxygen podemos extraer automáticamente la documentación del código.

5.4 Cambios aplicados

5.4.1 Adaptación a las nuevas librerías

Dado que se han actualizado las librerías que se usaran como dependencias de MATE debemos modificar las llamadas a funciones que existen en el código para adaptarlas a los cambios posibles.

En nuestro caso hemos pasado de usar la versión 6.1 de Dyninst a la versión 7.0. Esto es importante porque siempre hay que intentar mantener los componentes del programa lo mas actualizados posible, para aumentar la compatibilidad y favorecerse de las mejoras realizadas por los proveedores de este software, que a menudo incrementan la calidad de nuestra aplicación.

En nuestro caso este cambio tiene como consecuencia la necesidad de cambiar algunas llamadas a funciones, que han variado de una versión a la siguiente.

En el caso del AC se ha tenido que substituir el método `getPid()` que obtenía el identificador de un proceso. El cambio producido es que a partir de la versión 6.0 de Dyninst esta función se descontinuó en objetos de tipo thread, siendo exclusiva para procesos. A partir de esta versión se debe usar la función `getTid()` para obtener el identificador de un thread.
Los cambios mas comunes son los relativos a procesos y threads. Otro de los cambios relativos a Dyninst se produce al intentar llamar funciones propias de procesos en un objeto thread, como en el caso del *TaskManager* que intenta obtener, mediante la función `stopSignal()`, la señal con la que se ha finalizado un thread. En nuevas versiones de Dyninst esto no es posible de forma directa y se debe acceder al proceso que representa el thread primero y luego a través de este obtener la señal de parada.

5.4.2 Cambios en cascada

Debido a que se producen cambios en varios módulos de forma simultanea, existe la posibilidad, dado que los módulos son dependientes entre ellos, que algunos de los cambios realizados por otros miembros del equipo de trabajo tengan una repercusión en los demás módulos.

En nuestro caso los cambios realizados en el modulo C*ommon*, que es el paquete que contiene las clases comunes para varios módulos, son los que mas impacto tienen sobre el AC.

En el caso de la clase *Controller*, existe un ejemplo de cambio en cascada. En *Common* se decidió cambiar el método de lectura de los archivos de configuración y por tanto este cambio ha provocado que se hayan de cambiar la forma en que la clase *Controller* obtenía los datos de configuración.

En lugar de usar un objeto de clase *Config* para leer el archivo de configuración, ahora este objeto solo almacena los datos obtenido y el encargado de leer y tratar este fichero es una clase nueva llamada *ConfigHelper* que contiene un método llamado `ReadFromFile(path del documento)`.

Antes:

```
_cfg.LoadFromFile (_configFile);
```

después:

```
_cfg = ConfigHelper::ReadFromFile(_configFile);
```

5.4.3 Cambios generales

En adición a los cambios específicos también se han realizado cambios necesarios debido a la evolución del mismo lenguaje en que el programa está escrito. C++ como lenguaje de programación también evoluciona y por tanto debemos ir actualizando los ficheros de código para que se adecuen a los cambios del lenguaje.

El cambio mas significativo de este tipo es el referente a las cadenas de texto. En general se recomienda el uso de la clase *String* en lugar de usar punteros a caracteres.

Este cambio se ha aplicado de forma numerosa en todo el código.

Un ejemplo es la función *`StartTask(...)`*, método perteneciente a la clase *TaskManager*, que recibe como parámetro el *path* de la aplicación a iniciar. Este *path* es una cadena de caracteres pero en lugar de tratarla como tal usamos la clase *String* para representarla y por tanto cambiamos el tipo del parámetro que se admite.

Pasamos de esto:

```
bool TaskManager::StartTask (char const * path, char
** args, int envc, char ** env);
```

a:

```
bool TaskManager::StartTask (const std::string& path,
char ** args, int envc, char ** env);
```

Como consecuencia a la hora de llamar a la función usaremos este tipo de variable para pasar la cadena `path`.

Antes:

```
taskMngr.StartMPITask (_cmdLine.GetAppPath (),
_cmdLine.GetAppArgv ())
```

Después:

```
taskMngr.StartMPITask ((char *const)
_cmdLine.GetAppPath (), _cmdLine.GetAppArgv ())
```

5.4.5 Cambios concretos

En algunos casos, pese a que no son cambios motivados por la aplicación de la metodología se ha decidido modificar pequeñas cosas.

Uno de estos casos concretos en el caso del módulo AC es el método `FindByPid(int pid)` de la clase *TaskCollection*. Por alguna razón existía una duplicidad de métodos, ya que además del nombrado la clase *TaskCollecion* disponía de otra función llamada `Find` que realiza exactamente la misma tarea que `FindByPid`.

En este caso optamos por eliminar uno de ellos en concreto `Find`, ya que `FindByPid` es un nombre mucho mas descriptivo para esta función. Esto tiene como consecuencia trazar todas las llamadas a esta función y substituirlas por llamadas a la que conservamos.

5.4 Comentarios añadidos

Como parte de la documentación de MATE se han añadido al código fuente, esto es el conjunto de ficheros en C++ que forman la aplicación, comentarios que facilitan la comprensión de los elementos y

como se relacionen entre ellos.

Estos comentarios se dividen en dos tipos distintos, los comentarios *inline* y los comentarios Doxygen (descriptivos). Cada uno de ellos cumple una función distinta en en conjunto forman la documentación de código de MATE.

5.4.1 Comentarios *inline*

Este tipo de comentarios está incrustados entre las instrucciones del código y tienen como objetivo principal proporcionar una detallada descripción de cada linea o conjunto de estas a medida que se van leyendo. Esto permite que se describa con detalle el flujo de un conjunto de instrucciones a medida que vamos leyendo cada una de ellas.

El código original contenía comentarios de este tipo y se han conservado ya que explican de primera mano cual es la función de cada elemento.

Algunos ejemplos de comentarios *inline* son los siguientes:

```
void ShutDownSlave::Run (){
    char buf[10];
    char intSig[] = "Interrupt";
    // Loop that allows for more than one message received
    while(1){
         // Create client socket
         SocketPtr clientSocket = _SSock.Accept();
         // Blocking call that receives the message
         clientSocket->Receive(buf, sizeof(buf));
         // Memory comparison rather than strcmp
         if( memcmp(buf, intSig, strlen(intSig)) == 0 ){
              // Call interrupting function of the AC
              _ctrl.Interrupt();
              // Exit loop
              break;
         }
    }
    exit(0);
}
```

5.4.2 Comentarios Doxygen

Como se ha explicado con anterioridad en este documento, Doxygen es una herramienta que nos permite extraer documentación de una aplicación directamente de su código. Esto es posible gracias a que Doxygen analiza la sintaxis y semántica del código y además puede extraer información de ciertas lineas de comentario especificas. Este tipo de comentarios son especiales para Doxygen, este las identifica como tal, y están formateadas de forma que se pueda extraer información.

Con palabras clave como **@brief** o **@param** especificadnos a Doxygen que es cada componente del comentario y este identifica cada uno de ellos para poder procesarlos y generar la documentación con el formato deseado.

Estos son los comentarios mas numerosos y que tiene mas contenido. Todos los ficheros de cabecera contienen este tipo de comentarios para describir cada una de las clases implementadas y sus métodos.

Cada clase tiene un comentarios que incluye su definición así como los datos de creación y versión de esta.

Cada método contiene una breve descripción de su funcionamiento así como las especificación de parámetros, elementos de retorno o excepciones que lanza entre otros.

Algunos ejemplos de este tipo de comentarios:

```
/*
 * @brief Creates a new task.
 *
 * Creates a new task that represent the execution
 * of the program passed
 * as a parameter and adds it to the list of
 *  stored tasks.
 *
 * @param path Absolute location of the
 *  executable containing the app.
 * @param args Arguments that should be
 *  passed the application when it
 * is started.
 * @param envc Environment count. (Unused)
 * @param env Environments. (Unused)
 *
 *  @return true if the task has been
 *  successfully started, false otherwise.
```

```
     */
    bool StartTask (const std::string& path, char **
args, int     envc, char ** env);
```

6 Nuevas características

En esta sección se explican las características que se han añadido a MATE. Estas no son relativas a su funcionamiento básico sino que son referentes a la calidad de software.

6.1 Instalador

Una de las piezas clave en cualquier producto software que se quiera distribuir a terceras personas es un instalador que sea compatible con cualquier versión del sistema objetivo, y que permita de esta forma la implantación sencilla y rápida de la instalación sin exigir un alto conocimiento de esta.

Para MATE, ya que se trata de una aplicación orientada a UNIX, hemos optado por desarrollar un sistema de instalación basado en Make. Make es una herramienta de UNIX que permite, mediante una serie de sencillos *scripts*, la compilación de una aplicación y implantación de esta en el sistema.

La instalación de MATE se lleva a cabo a partir del código fuente de la aplicación. Sobre este se realizan unos procedimientos para generar los ejecutables que forman MATE como aplicación.

Fases de instalación:

- **Configuración:**

La primera fase de la instalación de cualquier producto software es la configuración del instalador para adaptarlo a la maquina en la cual se esta llevando a cabo la instalación. Para poder compilar el programa se deben conocer las características del sistema y este ha de disponer de las herramientas adecuadas.

El configurador de MATE, un *shell-script* llamado “configure”, se encarga de recolectar esta información y almacenarla en un fichero.

La información se compone, entre otros, del tipo de arquitectura que usa el sistema, la localización del compilador en el sistema de archivos, la existencia en el sistema de las dependencias de MATE y el *path* de instalación personalizado si se desea.

- **Compilación:**

En esta fase se usa la herramienta Make para generar los ejecutables a partir de los diferentes ficheros de código fuente. Compilando los archivos obtenemos los archivos objeto que mas tarde se enlazaran en librerías o directamente en el ejecutable que se usara para iniciar la aplicación.

Make usa una serie de ficheros (*makefiles*) que contienen las instrucciones para generar cada unos de los archivos intermedios que toman parte en la creación del ejecutable final.Estos archivos usan la información recolectada en la fase de configuración para hacer posible la compilación.

– **Instalación:**

Una vez los ejecutables han sido compilados y enlazados con las librerías estáticas de las que dependen es posible que el usuario quiera que se distribuyan estos binarios en carpetas predeterminadas para facilitar el acceso.

El instalador procede en esta fase a mover los archivos generados a su correspondiente directorio en el sistema de ficheros del usuario.

Requisitos del configurador:

- Soporte a usuario (opción -help)
- Opciones de selección
  - Compilador de C (opción -cc).
  - Compilador de C++ (opción -c++).
  - Directorio de cabeceras de MPI (opción -mpiinc).
  - Librerías de MPI (opción -mpilib).
  - Librería especifica de MPI (opción -mpilibrary).
  - Directorio de cabeceras de Dyninst (opción -dyninstinc).
  - Librerías de Dyninst (opción -dyninstlib).
  - Directorio de librerías de Dwarf (opción -dwarf).
  - Directorio de librerías de Binutils (opción -with-binutils).
  - Makefile objetivo (opción -with-make).
  - Binario de Doxygen (opción -with-doxygen).
  - Arquitectura del sistema (opción -arch).
- Control de dependencias.
- Búsqueda y configuración de dependencias en función de la arquitectura.

Requisitos del instalador:

- Compilación de los archivos fuente.
- Copia de los archivos ejecutables en el sistema.
- Limpieza de residuos de instalaciones previas.
- Generación de la documentación a partir del código.

Ficheros generados

Tal y como se aprecia en la descripción previa del proceso de instalación, en este toman parte varios ficheros diferentes que se encargan de cada una de las fases de la instalación.

- **Configurador** [configure]:

El configurador es un script escrito en shell (*sh*) que extrae los datos necesarios para la instalación del sistema. Además incluye la posibilidad de forzar muchos de los parámetros de forma que el usuario puede sobrecargar las elecciones que se toman y así elegir cual es la configuración que desea.

- **Instalador** [makefile(s)]:

MATE dispone de un *makefile* para cada uno de sus módulos además de uno general que es el encargado de ejecutar cada uno de los demás en un orden adecuado.

Estos ficheros contienen las instrucciones para crear cada uno de los ficheros intermedios y finales que forman MATE. Para cada fase de la compilación existen unas dependencias que se deben cumplir antes de pasar a la siguiente y el *makefile* se encarga de que la compilación se ejecute de forma adecuada.

Además dispone de unos objetivos (*targets*) adicionales que sirven para limpiar la instalación previa, es decir, eliminar los archivos residuales de previas compilaciones, y generar la documentación a partir del código.

Ficheros adicionales

El instalador hace uso de algunos archivos adicionales que proporcionan información sobre el sistema.

- **Archfind**: es un ejecutable que nos permite averiguar la arquitectura del sistema.
- **IsAbs**: se trata de un pequeño *script* que comprueba si un camino de directorio es absoluto.

Codificación

La implementación de este modulo se ha realizado usando principalmente las dos herramientas ya nombradas, *shell-scripts* y Make. Para generar el configurador se podría haber optado por hacerlo automáticamente con *autotools* pero para simplificar el profeso y el *script* final se optó por realizarlo manualmente a partir de un *script* parecido existente.

Los *makefiles* se generaron a partir de los que ya existían para MATE añadiendo nuevos para adaptarse a la trasformación de la aplicación.

Como contribución personal al proyecto se han realizado las tareas relativas a los *makefiles*.

Testeo

Como parte de MATE el instalador debe ser testeado para comprobar que su funcionamiento es correcto. Dentro del instalador la parte mas importante y a la vez mas susceptible a fallos es el configure ya que este debe adaptarse a diferentes situaciones en distintos sistemas.

Con este objetivo se han generado una serie de tests que comprueban que el *script* "configure" recolecta siempre la información adecuada y sobrecarga los valores de los parámetros cuando el usuario lo desea.

Estos tests se han realizado mediante Make, que nos permite producir los resultados de diferentes ejecuciones del *script* y con distintos parámetros de entrada y en diferentes sistemas y comprobar que son los resultados esperados.

6.2 Mecanismo de parada

Otro aspecto importante en producto software es la habilidad de este para terminarse de forma segura. Hasta el momento no existía un método de apagado de la aplicación sino que se tenia que abortar por medio de comandos.

Requisitos

El mecanismo de parada debe encargarse de cerrar el programa de manera limpia y sin que se corra peligro de perder información o provocar daños al sistema.

- Apagado controlado de MATE
- Difusión de la señal de apagado a los AC
- No influir en el tiempo de ejecución de la aplicación.
- Proporcionar control directo al usuario.

Diseño

Con estos requisitos en mente se ha diseñado un mecanismo que, aprovechando la clase *socket* del proyecto original usa este tipo de protocolo de comunicación para transmitir la señal de cerrado a todos los nodos del sistema distribuido y así se asegura de que todos los AC's activos se cierran en el momento que el usuario desea finalizar MATE.

El sistema de cerrado consiste simplemente de un *thread* que se ejecuta de forma silenciosa en cada uno de los nodos del *cluster* i en la máquina que ejecuta el Analyzer.

El thread de los nodos AC ejecuta una rutina de cliente que servidor que espera la señal del nodo principal para indicarle al AC que debe finalizarse. De forma complementaria en el nodo principal se ejecuta un thread que actúa como cliente y difunde la señal de cerrado cuando sea adecuado.

De este modo cuando la máquina central recibe la orden del usuario de que MATE debe cerrarse, esta envía por *socket* la señal a todos los nodos AC.

Este método permite reducir prácticamente a 0 la influencia del mecanismo de cerrado en el tiempo total de ejecución ya que los threads se bloquean esperando señales y no realizan esperas activas de modo que

el procesador queda libre para realizar las tareas relativas a la aplicación y a MATE.

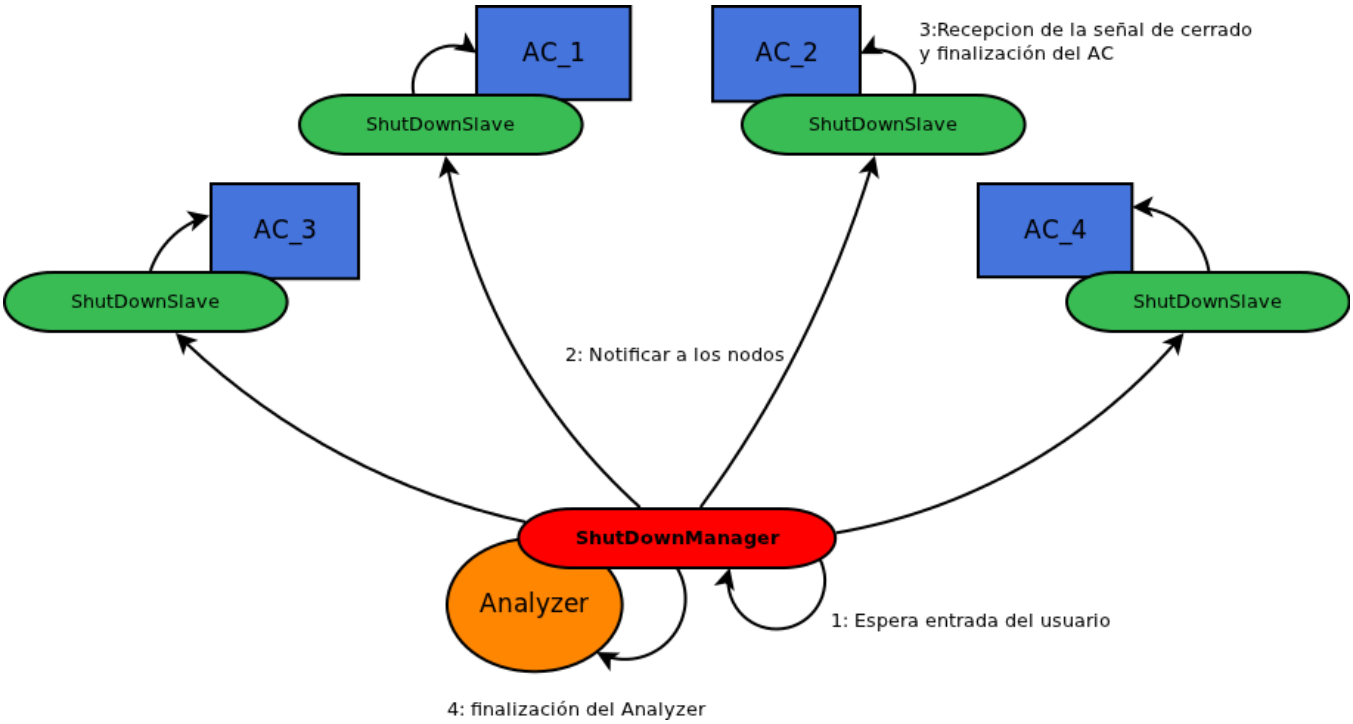


Figura 10 Diagrama que muestra el funcionamiento del mecanismo de parada.

Codificación

El mecanismo de parada se ha integrado en el código de MATE y por tanto está escrito en C++. En concreto la clase creada para el AC, como aportación personal, se llama `ShutDownSlave`.

Esta clase es una subclase de `ActiveObject` lo que permite que se comporte como un thread.

Utiliza los objetos definidos por los desarrolladores originales de MATE para representar *sockets*.

Testeo

Las clases usadas en el desarrollo de este componente han sido testeadas de forma unitaria, es decir, que funcionen bien por si solas. Debido a problemas con el entorno las pruebas del mecanismo de parada se han limitado a comprobaciones del funcionamiento de los *sockets* y del sistema de parada del AC.

7 Conclusiones

Este es un proyecto inusual que no tiene como objetivo la creación de una aplicación sino la de aplicar garantías de calidad a una ya creada. En este sentido el proyecto ha sido un éxito.

Se ha diseñado un entorno de desarrollo especialmente adaptado a las necesidades de MATE. Esperamos que esto represente el estándar para futuros proyecto con MATE y de esta forma se facilite el trabajo de los realizadores de este. Todos los componentes del entorno cuentan con un mecanismo de instalación automático (*scripts* de instalación) para que su implantación pueda ser realizada lo mas rápidamente posible.

Se han creado una serie de documentos que conjuntamente describen detalladamente una metodología que regula todos los aspectos del desarrollo de MATE. Esto hará posible la colaboración de terceras partes a la evolución del proyecto unificando el código obtenido.

El código de MATE ha sido actualizado siguiendo las normas de estilo especificadas en las guías y se ha intentado obtener el código mas limpio y correcto posible.

Además se ha documentado el código para que este sea más comprensible lo que reducirá notablemente el esfuerzo de futuros desarrolladores a la hora de familiarizarse con la aplicación.

MATE como aplicación era totalmente funcional al inicio de este proyecto, no obstante el trabajo realizado durante estos meses ha dotado al proyecto de características que, si bien no repercuten en las funcionalidades de esta, son una parte muy importante para su evolución. No obstante se han añadido algunas características nuevas, que no modifican el funcionamiento base pero hacen mas cómodo el uso de la aplicación.

En definitiva podemos decir que se han cumplido los objetivos planteados, y que el resultado de haberlos cumplido es transcendental para la evolución de MATE.

MATE es un proyecto con mucho futuro, la aplicación en si representa un paso adelante en la optimización de aplicaciones paralelas, y por tanto debe darse continuidad a la evolución de esta. Esperamos que con nuestra aportación se facilite a futuros desarrolladores la tarea de seguir trabajando para hacer de MATE la herramienta mas completa posible.

7.1 Posibles mejoras

Pese a que el proyecto se ha llevado a cabo de forma exitosa, existen posibles mejoras que se podrían haber aplicado, pero que debido a falta de tiempo o del conocimiento adecuado no han sido posibles.

MATE tiene en su código algunas partes incompletas que en la actualidad son prescindibles debido al uso que se le da pero que harían de MATE un programa mas completo.

- Mejoras de funcionalidad:

En el modulo AC en la clase Tuner solo está implementado el cambios de valores para variables de tipo *float*. Este es un parche que hace que MATE funcione para ciertos casos en que otros tipos son innecesarios pero seria mas completo si estuviera implementado para cualquier tipo. Completar esta parte del AC no era viable debido a la falta de experiencia con Dyninst y no se ha podido realizar.

- Licencia:

Otro factor importante que debe ser definido es bajo que términos se puede usar modificar y distribuir MATE. La licencia que se le aplique a la aplicación será relevante en el futuro de esta y por tanto debe ser responsablemente escogida.

- Testing:

Una parte importante de todo desarrollo software es asegurarse de que el producto final cumple con los requisitos establecidos. Para comprobar que el producto se comporta de la forma esperada se deben realizar una serie de *tests* en el entorno en que la herramienta debería funcionar una vez liberada. En nuestro caso este entorno es un *cluster* de ordenadores. De modo que el siguiente paso serie realizar una serie de *tests* de MATE en diferentes sistemas para corroborar los resultados obtenidos en los *tests* unitarios. Y además validar las características añadidas como el sistema de configuración o el de cerrado.

8 Bibliografía

Links

- Página oficial de Dyninst: http://www.dyninst.org/
- Paradyn (pagina de Dyninst 7.0): http://www.paradyn.org/html/dyninst7.0-software.html
- The Message Passing Interface (MPI) standard: http://www.mcs.anl.gov/research/projects/mpi/
- Apache server project: http://httpd.apache.org/
- Página oficial de Doxygen: http://www.stack.nl/~dimitri/doxygen/
- Página oficial de SVN: http://subversion.apache.org/
- C++ referencia: http://www.cplusplus.com/reference/
- An Introduction to the UNIX Make Utility: http://frank.mtsu.edu/~csdept/FacilitiesAndResources/make.htm
- Make manual: http://www.gnu.org/software/make/manual/make.html
- Creating UNIX libraries: http://www.cs.duke.edu/~ola/courses/programming/libraries.html

Documentos

- Anna Morajko, “Dynamic Tuning of Parallel/Distributed Applications”, UAB, 2003.
- Andrea Martínez, "Sintonización dinámica de aplicaciones MPI", UAB, 2010.
- Eduardo César, Definition of Framework-based Performance Models for Dynamic Performance Tuning, UAB, 2006.
- Paul Glezen, Branching with Eclipse and CVS, Part 2: Rebasing, IBM 2007.
- Juan Soulié, C++ Language tutorial, 2007

Índice de anexos