Institute of Geodesy and Navigation
Institut für Erdmessung und Navigation

Universidad Zaragoza
1542

Proyecto Final de Carrera
Ingeniería en Informática

Desarrollo de una Interfaz Gráfica y Optimización de Algoritmos de Cálculo para una Herramienta de Análisis de Compatibilidad en Frecuencia

Eduard Porta Martín-Moreno

Directora: Iva Bartunkova
Ponente: Jorge Júlvez Bueno

Insitute of Geodesy and Navigation
University of Federal Armed Forces, Munich

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Zaragoza, Septiembre de 2011

Agradecimientos:

A Javier y María del Carmen, mis padres, por su apoyo incondicional desde que tengo memoria

A David, mi hermano, amigo y confidente, por hacerme comprender que siempre existe otro punto de vista

A Bea, por aguantarme, comprenderme y no decirme lo que quiero oír sino lo que es necesario que escuche

A Iva, mi directora, y Jorge, mi ponente, por su tiempo, sus consejos y su ayuda en los momentos críticos

Resumen

El proyecto *Frequency Compatibility Analysis Tools* (FCAT) constituye un desarrollo llevado a cabo en el *Institute of Navigation and Geodesy* (IGN) de la *University of Federal Armed Forces* en Múnich, y financiado por la Comisión Europea. El resultado de dicho proyecto consiste en una serie de herramientas de análisis de compatibilidad de señales basadas en MATLAB e integradas en dos aplicaciones gráficas. Dichas aplicaciones permiten calcular valores especificados por el cliente para sistemas de navegación vía satélite arbitrarios, visualizando los resultados mediante gráficas, tablas y texto. El presente proyecto final de carrera consiste en dos partes diferenciadas, ambas enmarcadas en el desarrollo de FCAT.

La primera parte del proyecto ha consistido en el análisis, diseño e implementación de los módulos correspondientes a la interfaz gráfica de las aplicaciones. Con la intención de hacer esta parte más interesante desde el punto de vista académico, a la par que mejorar la productividad en futuros proyectos, se ha implementado para esta tarea un *framework* genérico. Dicho *framework*, al que el autor ha decidido llamar *Forget About Complicated Interface Layouts* (FACIL), se ha diseñado tratando de maximizar el compromiso entre flexibilidad, simplicidad y escenarios de uso abarcados por la herramienta. FACIL permite dotar de interfaz gráfica a una colección de algoritmos de cálculo arbitraria con tan solo escribir unos pocos ficheros de configuración. Su validez y utilidad se han verificado mediante su aplicación al proyecto GASIP, otro proyecto del IGN con la Comisión Europea.

En la segunda parte del proyecto se ha tratado la optimización de algoritmos de cálculo intensivos en tiempo de CPU para el proyecto FCAT. Para esta parte se han evaluado distintas alternativas como la vectorización del código, técnicas de paralelización o el uso de la tarjeta gráfica del equipo mediante CUDA u OpenCL.

En definitiva, este proyecto final de carrera ha constituido toda una experiencia formativa, tanto desde el punto de vista académico como desde el profesional. A lo largo del mismo ha sido necesario cumplir plazos, coordinarse con otros miembros del instituto, tratar de satisfacer las expectativas y requisitos del cliente, y en general todo aquello que implica el trabajo en un proyecto real.

Tabla de Contenido

1

Introducción

Este capítulo ofrece una visión global del proyecto desarrollado, su contexto, metodología y objetivos, e introduce los restantes capítulos de la presente memoria

Estructura del Documento

A continuación se detalla la estructura de esta memoria:

Resumen

Resume brevemente en qué ha consistido el proyecto desarrollado, describiendo las tareas realizadas y explicando las implicaciones académicas y profesionales del mismo para el autor.

1 - Introducción

Es el capítulo en el que nos encontramos. Ofrece una visión global del proyecto desarrollado, su contexto, metodología y objetivos, e introduce los restantes capítulos de la presente memoria.

2 - FACIL (I): Descripción del *Framework*

Describe el *framework* FACIL, explica a qué tipo de proyectos es aplicable y expone las características más interesantes de su estructura y funcionamiento.

3 - FACIL (II): Desarrollo y Aplicación

Explica el desarrollo de FACIL, desde antes de constituirse como un proyecto separado hasta la versión final y su aplicación a los proyectos GASIP y FCAT.

4 - Optimizaciones

Documenta las posibilidades de optimización y las técnicas aplicadas a un algoritmo costoso en tiempo del proyecto FCAT.

5 - Resultados y Conclusiones

Explica los resultados obtenidos, su correspondencia con los objetivos marcados y las conclusiones que se derivan de los mismos.

Bibliografía

Lista las fuentes documentales empleadas para la realización del presente proyecto.

Anexos

Colección de documentos adicionales que se ha considerado interesante añadir y que complementan la información disponible en el documento principal.

Marco del Proyecto

El presente proyecto final de carrera (PFC) ha consistido en el análisis, diseño e implementación de diversos módulos y optimización de algunos algoritmos de cálculo dentro del proyecto *Frequency Compatibility Analysis Tools* (FCAT), desarrollado en el *Institute of Geodesy and Navigation* (IGN) de la *University of Federal Armed Forces* de Múnich. El citado PFC ha sido financiado mediante una beca Erasmus Prácticas, obtenida a través de la Fundación Empresa Universidad de Zaragoza (FEUZ), y una beca STIBET II, concedida por el Servicio Alemán de Intercambios Académicos (DAAD).

Institute of Geodesy and Navigation

Figura 1 – Observatorio del IGN

El IGN [W1] (Figura 1) trabaja desde hace veintiocho años en una larga lista de aplicaciones y temas de investigación relacionados con los sistemas de navegación vía satélite globales (GNNS) abarcando investigación teórica, análisis de sistemas, integración de navegación vía satélite con otros sensores, desarrollo de algoritmos y desarrollo de software, así como el desarrollo de sistemas de creación de prototipos. Actualmente el IGN juega un papel activo con respecto al desarrollo y la optimización del Sistema de Navegación Europeo GALILEO.

Proyecto FCAT

La Comisión Europea (CE) necesita una serie de herramientas de simulación de uso interno, con el propósito de realizar su propia valoración independiente acerca de los análisis de compatibilidad, los cuales son necesarios en el transcurso de los diversos procesos de coordinación entre GALILEO/EGNOS y otros GNSS.

Para poder asegurar el correcto funcionamiento con otros GNSS, la CE necesita asegurar que se ha alcanzado la compatibilidad entre los diversos sistemas. Cuando se habla de compatibilidad en este contexto, se entiende que un sistema no causará ningún tipo de interferencia perjudicial en el otro sistema [A1].

Hasta ahora, la CE cuenta con la asistencia la ESA y de expertos de estados miembros, en base a su disponibilidad y buena voluntad, lo que, por supuesto, no es una solución aceptable. Por esto, la CE no dispone, actualmente, de los medios para forjar una opinión independiente acerca de los resultados de compatibilidad siempre que lo necesite.

Tener sus propias herramientas de simulación de compatibilidad permitirá a la CE tener una mayor capacidad de reacción y estar mejor informada sobre los problemas potenciales de compatibilidad entre GALILEO/EGNOS y otros GNSS. Dichos problemas de compatibilidad conciernen a todos los servicios de GALILEO y EGNOS en todas las actuales y posibles futuras bandas de frecuencia.

El propósito del proyecto FCAT es, por tanto, desarrollar dichas herramientas, las cuales facilitarán a la CE la evaluación de la compatibilidad en radiofrecuencia entre GALILEO/EGNOS y otros GNSS. Dichas herramientas de simulación deben tener la capacidad de realizar el análisis de compatibilidad por medio de las metodologías aplicables acordadas con otras GNSS a nivel bilateral y multilateral, basándose en las pertinentes recomendaciones de la ITU [D1].

Proyecto GASIP

El proyecto GASIP (Galileo Signals Performance) es otro proyecto desarrollado por el Instituto para la CE. La herramienta de simulación GASIP es una aplicación que calcula características de una señal GNSS para evaluar su rendimiento, bajo una constelación de satélites, canal y receptor determinados.

La interfaz gráfica de las herramientas desarrolladas en el proyecto FCAT comparte muchas características con GASIP y algunos módulos han sido reutilizados. Además, a pesar de que en un principio no estaba previsto, las decisiones de diseño realizadas en FCAT permitieron al autor realizar una pequeña contribución al proyecto GASIP.

Objetivos, alcance y motivación

Según la propuesta del puesto de prácticas en el que el autor ha realizado su proyecto, la tarea a completar consistía en el desarrollo de una aplicación de análisis de compatibilidad de señales en MATLAB. Para ello, se requería en primer lugar, el desarrollo de una interfaz gráfica modular y bien estructurada; y, posteriormente, la integración con los módulos de cálculo desarrollados por otros miembros del equipo[1].

Adicionalmente, la necesidad de incrementar el interés del proyecto desarrollado desde el punto de vista académico, unida a la buena acogida de algunas de las ideas del autor por parte de la organización, dieron lugar a nuevos objetivos, expandiendo el alcance de dicha propuesta de forma horizontal y vertical.

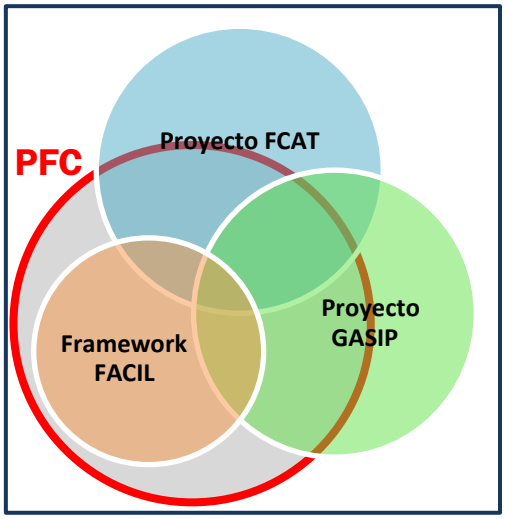


Figura 2 - Alcance del proyecto

De este modo el alcance de este proyecto puede verse como una intersección de conjuntos (Figura 2). En primer lugar se encuentran los objetivos enmarcados en el proyecto FCAT, de los cuales la mayor parte corresponde a la propuesta original. En segundo lugar cabe nombrar los objetivos fijados para el *framework*[2] FACIL, surgido en el seno del proyecto principal y cuyo desarrollo se justificará y explicará en detalle más adelante. Por último, citaremos los objetivos marcados para el autor respecto al proyecto GASIP. Los tres ámbitos están, como se verá en posteriores apartados, íntimamente relacionados y comparten un número importante de características entre sí.

Objetivos Dentro del Proyecto FCAT

A continuación se describirán los objetivos que se marcaron para el autor en su papel como desarrollador dentro del proyecto FCAT.

Por tratarse de un proyecto de características similares, muchos de los requisitos del proyecto FCAT coinciden con el proyecto GASIP. Por esta razón, gran parte de la lista de

[1] El contenido completo de dicha propuesta puede consultarse en el anexo A.1.

[2] Se emplea el anglicismo *framework*, por su uso ampliamente extendido en el desarrollo de software y por no existir un término con el mismo significado en castellano. Se entenderá como un esqueleto, esquema o patrón base para la implementación de una aplicación completa [L1].

requisitos puede resumirse en que la aplicación, al margen de las evidentes diferencias en los cálculos a realizar, debe ofrecer en su interfaz gráfica, como mínimo, todas las prestaciones que GASIP ofrece (Figura 3). Estos requisitos pueden resumirse en el siguiente listado:

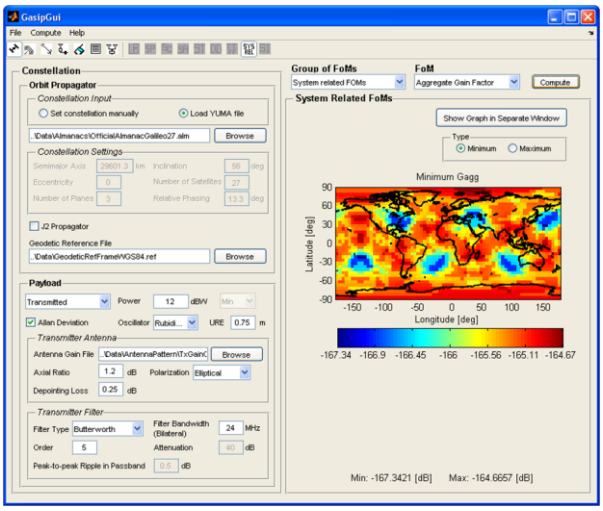


Figura 3 - Interfaz gráfica de GASIP

R1. Panel de entrada con los datos a pasar como parámetro a los algoritmos de cálculo.

R2. Comprobaciones de rango y tipo sobre estos valores de entrada antes de ejecutar las funciones de cálculo.

R3. Panel de salida con listas desplegables para seleccionar la función de cálculo a ejecutar y controles para visualizar los resultados mediante gráficas y texto.

R4. Estructura modular e interfaz sencilla con las funciones de cálculo.

R5. Funciones para crear a partir de una plantilla, cargar desde archivo o guardar los datos del panel de entrada.

R6. Opción para exportar los datos de salida a diversos formatos, entre ellos CSV (comma separated values), JPEG, TIFF (formato de imagen) o FIG (formato propio de MATLAB para almacenar figuras).

A estos requisitos se suman otros propios de FCAT, algunos de ellos especificados al inicio del desarrollo y otros surgidos a partir de ideas de algún miembro del equipo, peticiones del cliente o necesidades surgidas a partir de la implementación de los módulos de cálculo. A continuación se listan los más importantes:

R7. Además de las opciones de gráficas y texto plano, opción a visualizar los resultados en el panel de salida mediante tablas.

R8. Diálogos donde se visualice el contenido de algunos ficheros de entrada, concretamente aquellos que contienen las descripciones de los sistemas y señales GNSS, mediante texto, tablas y gráficas.

R9. Editor de matrices con el propósito de poder editar algunos de los ficheros de entrada en formato MAT.

R10. Optimización de los algoritmos de cálculo que se presten a ello de forma que se reduzca de forma visible la espera del usuario.

Motivación para el Desarrollo del Framework FACIL

Durante una fase temprana del desarrollo se planteó la posibilidad de separar el código y datos específicos de cada herramienta, de las partes comunes entre ellas y con GASIP, como la estructura de la interfaz gráfica o las funciones de manipulación de archivos. De este modo y con algunas generalizaciones, se pretendía generar un código unificado aplicable a las dos herramientas del proyecto.

Tras comprobar la viabilidad y previendo su potencial aplicación a otros proyectos, se decidió implementar las características comunes en un *framework* desarrollado simultáneamente, pero como un proyecto independiente. El autor decidió llamar a este proyecto *Forget About Complicated Interface Layouts* (FACIL). El desarrollo de un *framework* puede presentar ciertas desventajas, como un incremento temporal en costes derivado de las generalizaciones y la mayor complejidad en ciertos aspectos de la lógica interna. Sin embargo, los beneficios para la organización son numerosos [L2] [L3]:

- Estandarización del software dentro de la organización.
- Reducción de costes y tiempos de desarrollo en futuros proyectos.
- Mayor robustez de las futuras aplicaciones, ya que se parte de una base que se revisa de forma periódica.

Además, dado que debían implementarse dos herramientas muy similares, el incremento en costes del desarrollo se vio mitigado por la reducción en el numero de módulos a desarrollar, a casi la mitad.

Objetivos del desarrollo de FACIL

Cuando se decidió la separación entre *framework* y las partes específicas de cada aplicación, también fue necesario definir ciertos objetivos y requisitos que FACIL debía cumplir para rentabilizar su desarrollo. A continuación se resumen los más importantes:

R1. Ser sencillo de utilizar, ampliar y configurar.

R2. Cumplir los estándares de calidad de la organización (estándares de codificación, estructura modular, código reusable, robusto).

R3. Minimizar las pérdidas de eficiencia o rendimiento habitualmente derivadas de la generalización del código [L4].

R4. Ser suficientemente flexible para poder aplicarse a un elevado número de proyectos.

Planificación

El proyecto FCAT está planificado para llevarse a cabo con una duración de dieciocho meses, la estructura de esta planificación puede observarse en la Figura 4. En el momento de la incorporación del autor, algunos detalles del diseño de la aplicación se encontraban ya parcialmente definidos.

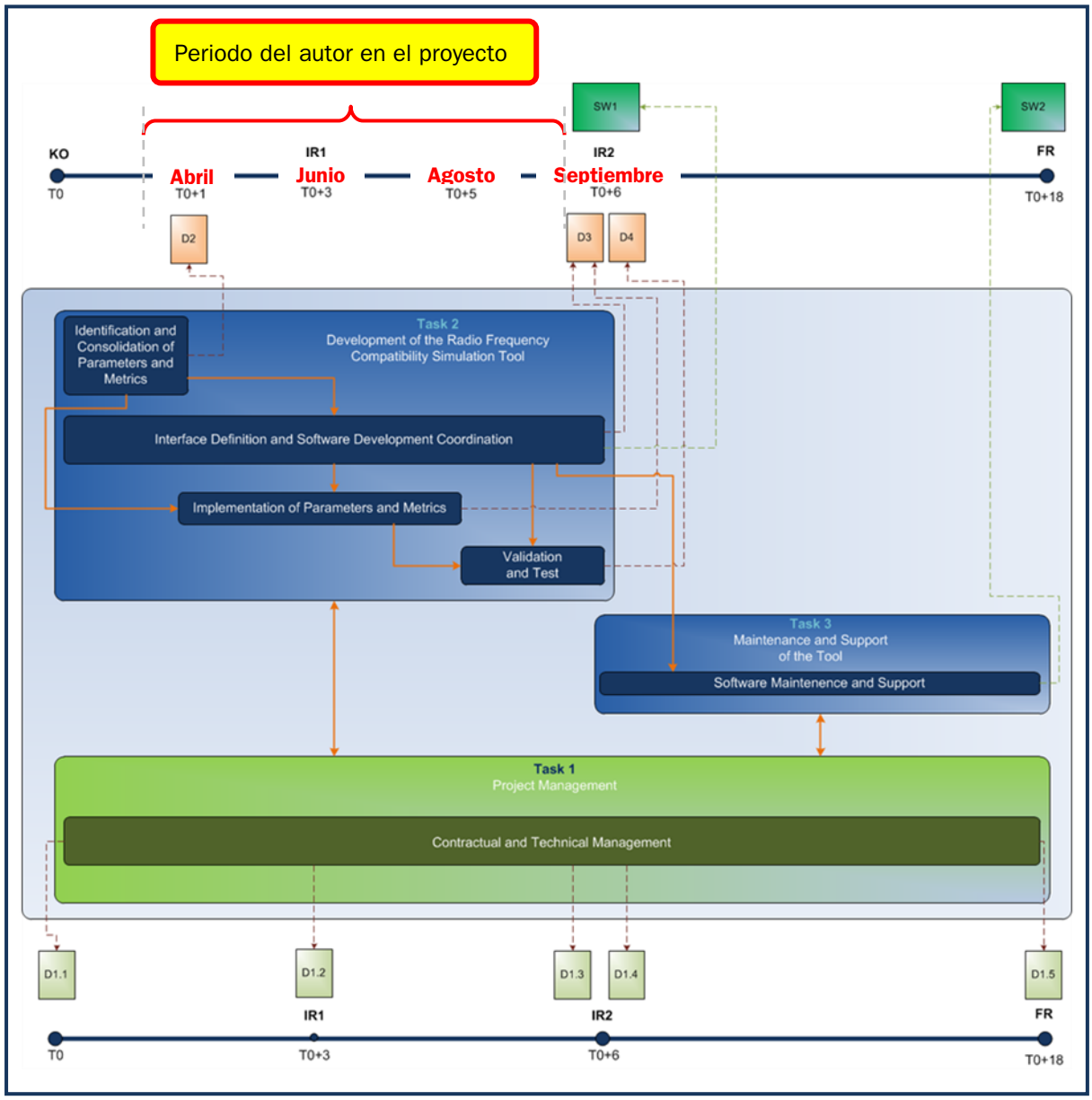


Figura 4 - Planificación del Proyecto FCAT

El autor ha participado principalmente en el diseño, implementación y pruebas de las dos interfaces gráficas de FCAT, dedicando la mayor parte de este tiempo al desarrollo del *framework* FACIL sobre el cual se sostienen. Adicionalmente, también ha realizado

optimizaciones en los algoritmos de cálculo implementados por otros miembros del equipo y desarrollado una versión reducida de otro proyecto mediante el uso de FACIL.

Inicialmente se planteó aplicar una metodología de desarrollo ágil. Se estudiaron principalmente dos alternativas: Scrum [L5] y Extreme Programming (XP) [L6], pero ninguna de las dos se adecuaba completamente a las necesidades y características del proyecto. Finalmente, se decidió aplicar una variante más flexible tomando las características interesantes de ambas metodologías de modo similar a [L7].

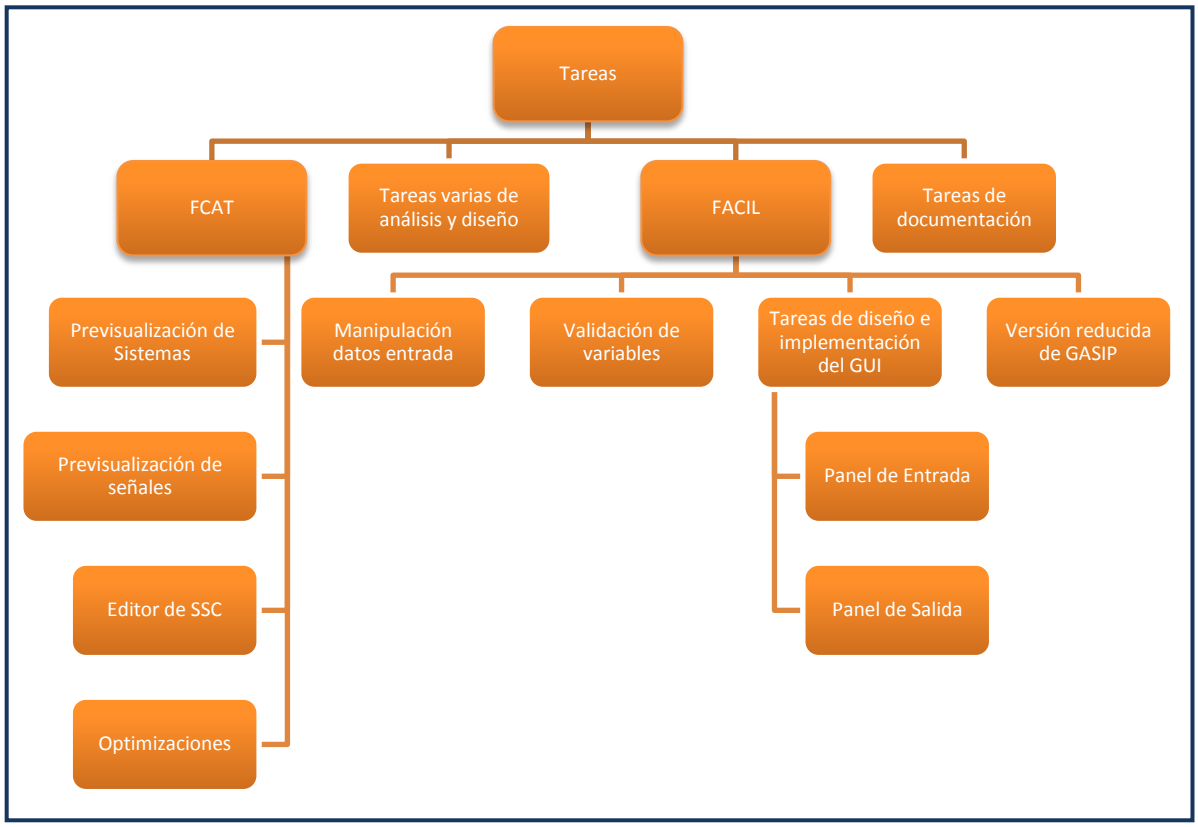


Figura 5 - Esquema de Tareas Planificadas

Por la naturaleza de este tipo de metodología de desarrollo, la planificación se ha tratado de forma dinámica, fijando en cada iteración los sub-objetivos a cumplir en base a un listado de objetivos fijo[3]. La división lógica de las tareas se muestra de forma esquematizada en la Figura 5.

[3] Dado que los sub-objetivos de esta lista han cambiado a lo largo del proyecto y es difícil expresar esta evolución en un documento estático, se ha decidido presentar únicamente el estado final de esta lista de objetivos (*backlog* en terminología Scrum), ésta puede leerse en el anexo A.2.

Fases del Desarrollo

En el apartado anterior se ha tratado la planificación del proyecto desde un punto de vista metodológico y mostrando una aproximación estática a los objetivos y sub-objetivos planteados a lo largo del mismo. Con el objeto de facilitar la comprensión, y por completitud, se presenta a continuación un listado secuencial por fases o hitos que agrupan los objetivos marcados de forma cronológica. También se intercalan las decisiones más importantes:

H0. Documentación, familiarización con la organización y el entorno de trabajo y análisis de alternativas.

H1. Prototipo de generación automática del panel de entrada.

Se decide generación en tiempo de ejecución frente a generación en tiempo de diseño.

H2. Sistema de complementos para los controles del panel de entrada. Complemento de previsualización de sistemas.

Se decide la separación del framework FACIL como proyecto independiente

H3. Segunda versión del panel de entrada: Controles agrupados en campos, se abstrae la lectura y escritura de los datos. Complementos de edición de coeficientes de separación espectral (SSC) y previsualización de señales.

H4. Primera versión del panel de salida con soporte para gráficas y texto plano para uno o más grupos de funciones y una o más funciones por grupo.

H5. Manipulación de datos de entrada: Abrir, guardar y cargar desde plantilla.

Se decide que es necesario poder mostrar tablas en el panel de salida

H6. Segunda versión del panel de salida con soporte para tablas y posibilidad de redimensionar la ventana.

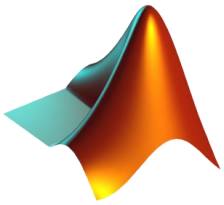
H7. Tercera versión del panel de entrada implementado con objetos: Se mejora la usabilidad, flexibilidad y posibilidades de ampliación.

H8. Versión inicial de FACIL. Implementación de una versión reducida de GASIP.

H9. Versiones Finales de FCAT y FACIL. Documentación y pruebas.

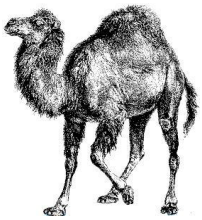
Tecnologías Empleadas

El presente proyecto consiste puramente en un desarrollo de software y, por lo tanto, todas las tecnologías listadas a continuación corresponden a aplicaciones informáticas usadas durante el desarrollo, obviando otros detalles como el equipo físico empleado:



- Mathworks MATLAB [W3]: Es el entorno de desarrollo y lenguaje de programación usado en prácticamente la totalidad del código de este proyecto. Está principalmente orientado a su uso por parte de personal científico e investigador. En este desarrollo se han usado las versiones 2007a, 2010b y 2011. La decisión de emplear este lenguaje fue impuesta por la CE en su convocatoria a licitación para el proyecto, así como la obligación de que funcionase en la versión 2007a del mismo [D1].

- Oracle Java [W4] y Netbeans [W5]: Java y Netbeans son, respectivamente, un lenguaje de programación orientado a objetos y un entorno de desarrollo para el mismo. Dado que MATLAB utiliza Java en sus objetos gráficos se ha empleado este lenguaje para realizar algunas extensiones necesarias a la interfaz gráfica.



- Perl [W6]: Este lenguaje de programación interpretado es extremadamente útil para el tratamiento de expresiones regulares. Se usó para las primeras tentativas de análisis de los ficheros de entrada. Posteriormente se ha pasado a utilizar el tratamiento de expresiones regulares incluido en MATLAB.

- Además de las anteriores tecnologías, usadas en la implementación de la aplicación, se han empleado diversas herramientas ofimáticas para las tareas de análisis, diseño y documentación; y el sistema de control de cambios Subversion. Tampoco hay que olvidar aquella que, desde el punto de vista del autor, es una de las herramientas más importantes y versátiles del ingeniero informático: papel y lápiz.

2

FACIL (I): Descripción del *Framework*

En este capítulo se describirá el *framework* FACIL, se explicará a qué tipo de proyectos es aplicable y se expondrán las características más interesantes de su estructura y funcionamiento

Introducción a FACIL

FACIL es un *framework* para MATLAB, pensado para facilitar el trabajo a científicos e investigadores que usan este entorno y que muchas veces se ven obligados a programar sus propias interfaces gráficas si desean una interacción más visual. FACIL aporta aquella funcionalidad habitualmente común a todas estas interfaces, además de un número de opciones de configuración y ampliación que lo hacen aplicable a un gran número de proyectos de diferente naturaleza.

Figura 6 - Ejemplo de Interfaz Gráfica Implementada con FACIL

El principal requisito para que FACIL sea aplicable a un proyecto, es que éste se componga de una colección de algoritmos de cálculo no interactivos, con una serie de variables de entrada, y resultados que puedan presentarse con gráficas, tablas y texto plano. No obstante, el usuario es libre de agregar interactividad o cualquier otra funcionalidad adicional mediante complementos.

La interfaz generada consta de un panel con todas las variables de entrada en el que se realizan automáticamente chequeos de rango y tipo; un panel de salida donde se presentan los resultados con opción a controlar la visualización de los mismos; y menús que permiten guardar o cargar los datos de entrada y salida, así como configurar diversos parámetros de la interfaz[4] (Figura 6).

[4] El funcionamiento de dicha interfaz se detalla en el Manual de Usuario, disponible en el anexo B.1.

Características Principales

A continuación se enumeran las características más destacables, de las que un desarrollo puede beneficiarse mediante el uso de FACIL.

Sencillo

Es posible desarrollar una aplicación completa con unos pocos ficheros de configuración y escasas líneas de código. La sintaxis empleada está pensada para que sea sencilla de entender y utilizar por parte del personal científico e investigador, posiblemente no familiarizado con conceptos avanzados de programación.

El panel de entrada se genera automáticamente a partir de un fichero de descripción de variables con sintaxis similar a JSON [A2]. El de salida se genera automáticamente a partir de un fichero de descripción de funciones o figuras de mérito. Este último comparte la sintaxis de las opciones de configuración del panel de salida con la usada para las variables de entrada.

La codificación necesaria para la interfaz entre FACIL y los algoritmos de cálculo existentes es mínima y sencilla de comprender y realizar.

Ampliable

Con el objeto de añadir nuevas opciones de visualización y entrada de datos, pueden añadirse complementos con funcionalidad específica a los *widgets*[5] de ambos paneles.

[5] A lo largo de esta memoria se emplearán los términos control, *widget* y campo, que pueden emplearse como sinónimos bajo la definición de [L8]: *"Un control es un objeto que reside en un panel accesible para el usuario y que permite visualizar información"*. Al margen de esta definición haremos las siguientes distinciones que facilitarán la compresión del texto:

- Los controles en nuestro proyecto son aquellos generados mediante las sentencias `uicontrol` o `javacomponent` e integrados en el propio MATLAB o la biblioteca Swing de Java.
- Los campos son objetos empleados en una implementación intermedia, formados por un control o un panel con varios controles, que representan una variable única, y que emplean métodos no estándar para acceder a sus propiedades y métodos.
- Los *widgets* están formados por un panel que contiene uno o más controles y, a diferencia de los campos, pueden ser usados como controles estándar con métodos y propiedades adicionales.

Pueden añadirse nuevos *widgets*, tanto para extender las capacidades de FACIL, como específicos para una aplicación, aprovechando mediante herencia de objetos la funcionalidad de los existentes.

Funcional

Se incluye en FACIL toda aquella funcionalidad que se ha considerado relevante y útil para proyectos de estas características:

- Se proporciona funcionalidad de barra de progreso y cancelación con una interfaz sencilla de utilizar, para los algoritmos costosos en tiempo.
- Los *widgets* incluidos realizan comprobaciones de tipo, rango, existencia y validez de ficheros de forma dinámica, mostrando de forma visual la corrección del dato contenido y deshabilitando el acceso a las funciones de cálculo mientras el error no sea subsanado.
- Los datos de entrada pueden guardarse y cargarse empleando ficheros de texto con sintaxis XML para que sean sencillos de editar fuera de la aplicación.
- Los datos de salida pueden guardarse en toda una variedad de formatos lo cual añade múltiples opciones de reutilización (como imágenes, como variables en ficheros Mat, como gráficas editables o en formato CSV).

Seguro

Se ha contemplado el caso de que algunos de los datos de entrada empleados en los cálculos sean confidenciales y no deban ser accesibles desde una versión ejecutable de la aplicación. Estos datos podrían ser incluidos en las propias funciones de cálculo, sin embargo, esto disminuye la flexibilidad y diluye la división entre funciones y datos.

Por esto, se ha creído interesante añadir la capacidad de definir constantes de entrada ocultas con datos predefinidos por el desarrollador, que no figurarán al guardar los ficheros de entrada y que pueden, opcionalmente, omitirse también de los ficheros de salida. Además, para evitar el acceso a estas constantes, existe la opción de obtener las descripciones de variables y funciones de ficheros MAT en lugar de texto, de forma que sean encriptados al generar el ejecutable.

Arquitectura

Dependiendo de cómo se utilice, un *framework* puede clasificarse como de *"caja negra"* o *"caja blanca"*. Mientras que con el primer tipo, la aplicación emplea los módulos disponibles en el mismo, el segundo tipo, permite la modificación o refinamiento (habitualmente mediante herencia de clases y abstracción) de su lógica interna para adaptarla a las necesidades del desarrollador [L3].

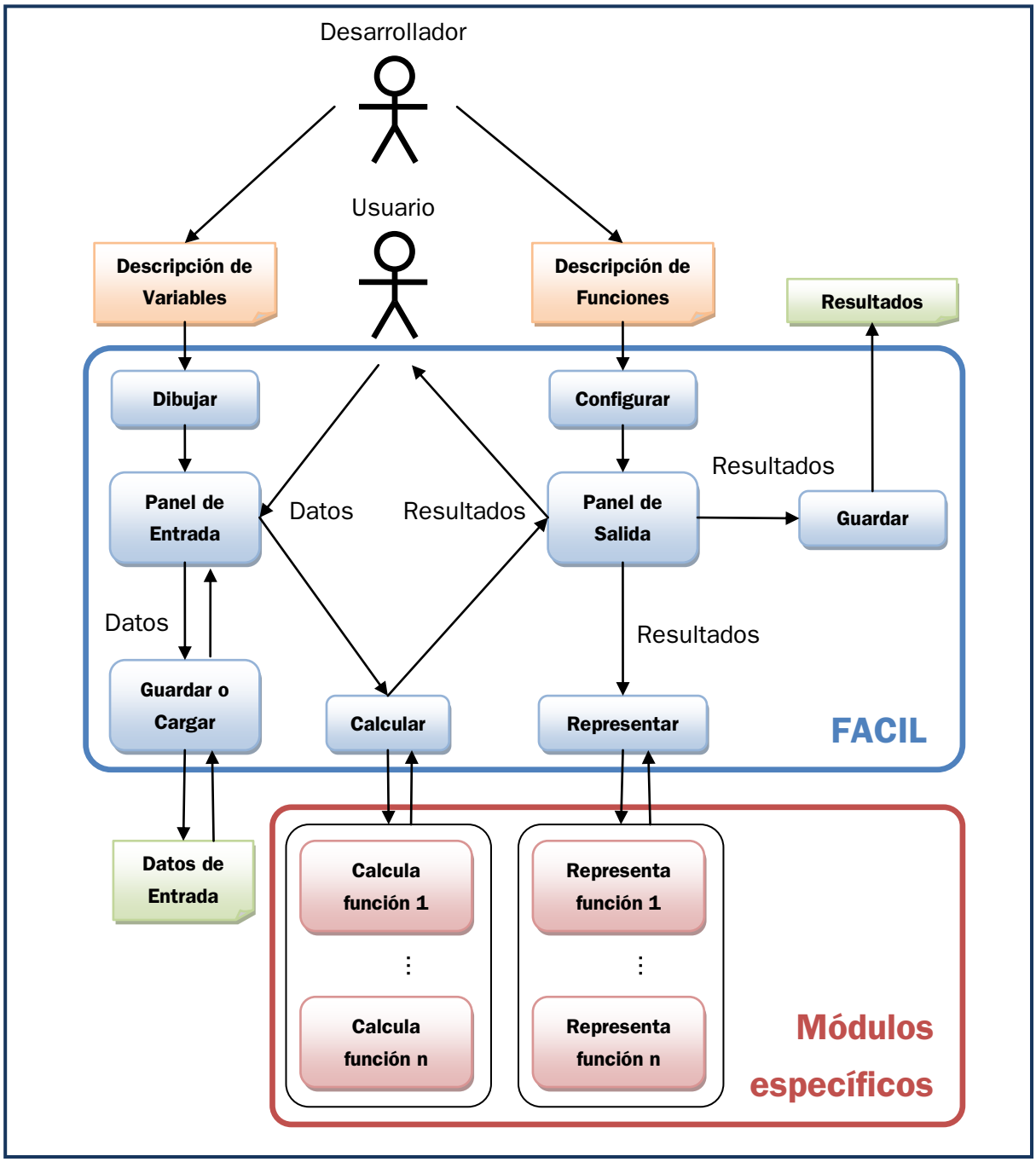


Figura 7 - Esquema simplificado de una aplicación implementada con FACIL

En este sentido, FACIL es un *framework* de *"caja gris"*, puesto que una parte del mismo funciona como una caja negra y, sin embargo, la funcionalidad de los paneles de entrada y salida puede ampliarse mediante la implementación de nuevos *widgets* y complementos a partir de los existentes.

Funcionalidad Básica: *Framework* de Caja Negra

FACIL puede ser usado como un *framework* de caja negra empleando los módulos ya disponibles para generar una aplicación completa. El esquema de la Figura 7 presenta de forma simplificada la estructura de una aplicación implementada con FACIL y la interacción entre el usuario, el desarrollador y los distintos componentes de la misma:

- Los módulos de FACIL.
- Los módulos específicos de la aplicación desarrollada.
- Los ficheros de configuración (escritos por el desarrollador).
- Los ficheros de entrada y salida (que son leídos y posiblemente modificados por el usuario).

Funcionalidad Avanzada: Framework de Caja Blanca

Como ya hemos dicho, la funcionalidad de caja blanca de FACIL estriba en que el desarrollador puede implementar sus propios *widgets* y complementos a partir de los existentes. Para ello existe toda una colección de funciones que facilitan esta tarea y que pueden ser consultadas en el manual del desarrollador[6]. El nivel de la jerarquía de objetos del que partirá y por tanto, la dificultad de la implementación, dependerá principalmente de las necesidades del desarrollador y su habilidad como programador (Figura 8).

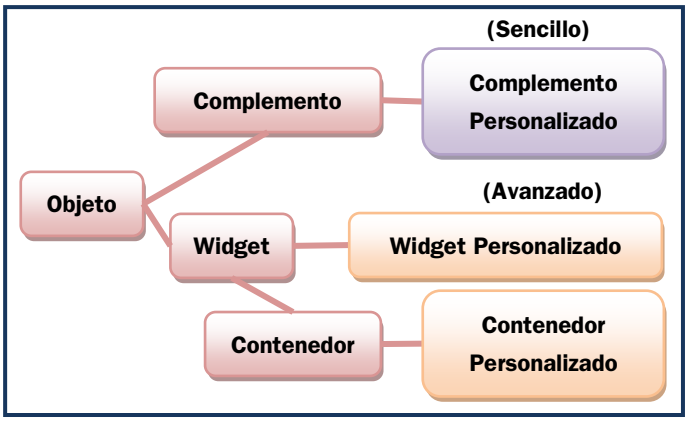


Figura 8 - Posibilidades de Herencia Simplificadas

[6] Este manual puede ser consultado en el anexo B.2.

Widgets y Complementos

Los *widgets* y los complementos son una parte fundamental de FACIL puesto que forman la mayor parte de la interfaz gráfica. Los primeros se encargan por sí mismos de la validación, control de dependencias entre variables y conversiones de tipos de datos entre otros; y los segundos permiten extender la funcionalidad del *framework* base. Por la forma en que se han implementado, es posible emplear estos objetos de forma independiente como una biblioteca de objetos para diseñar interfaces gráficas en MATLAB.

Jerarquía

Los *widgets* implementados se dividen en dos grandes grupos, los *widgets,* propiamente dichos, como equivalentes gráficos de un único dato; y los contenedores, que descienden de un *widget* especial, con métodos y propiedades que le permiten albergar otros *widgets*. Los complementos descienden, junto con los *widgets,* de una clase objeto directamente superior debido a ciertas diferencias en la implementación. Puede verse la jerarquía implementada completa (exceptuando complementos y *widgets* específicos de GASIP o FCAT) en la Figura 9.

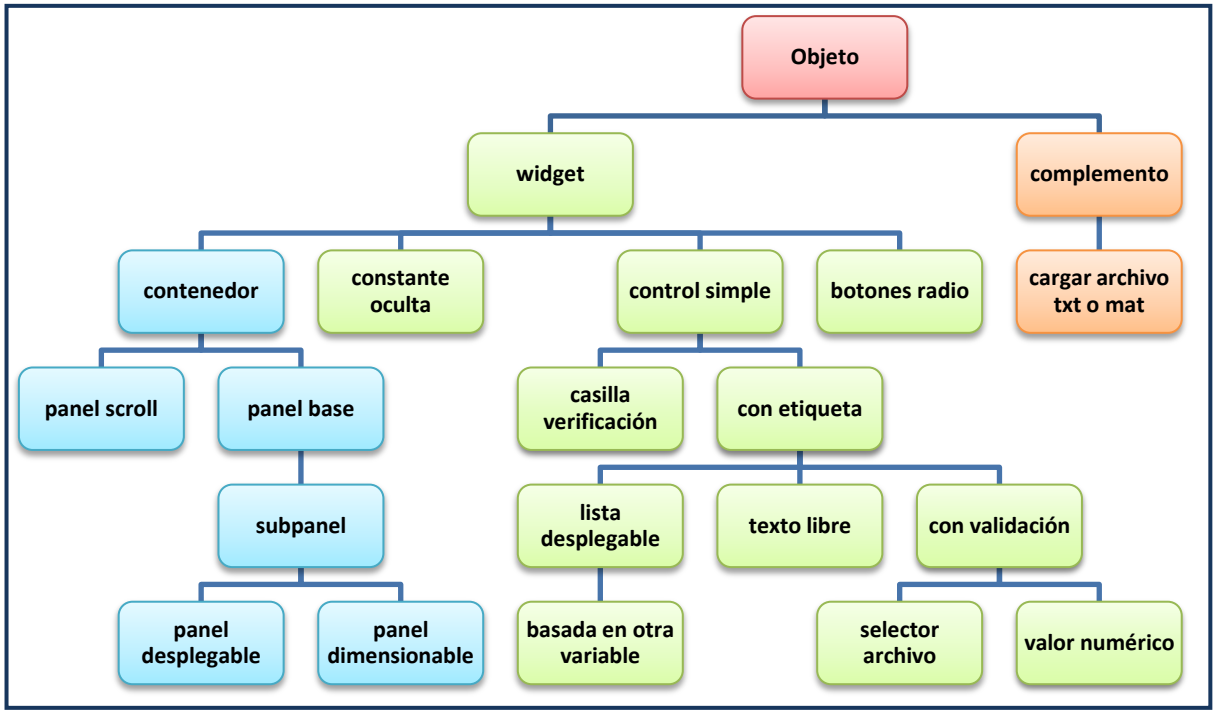


Figura 9 - Jerarquía de Objetos de FACIL

Propiedades

Los *widgets* comparten con los complementos ciertas propiedades heredadas de la clase objeto. Las más importantes son:

P1. **Enable:** Valor booleano que controla si el elemento y sus hijos son accesibles.

P2. **Valid:** Valor booleano que indica si el dato contenido es válido.

P3. **Ready:** Valor booleano dependiente de los anteriores que indica si el estado del *widget* es aceptable para ser usado (si es válido o inválido pero deshabilitado).

P4. **Key:** Nombre de la variable asociada. En un *widget* se genera automáticamente a partir del nombre del *widget* y el prefijo del tipo de datos, mientras que, en un complemento puede ser establecido manualmente.

P5. **Value:** Valor del *widget*.

P6. **Name:** Nombre del *widget*.

P7. **Type:** Tipo de dato de la variable.

P8. **MatlabType:** Tipo de la variable desde el punto de vista de MATLAB, que puede ser distinto del tipo de dato real de la variable (por ejemplo un entero sigue siendo un *double* en MATLAB).

P9. **DefaultValue:** Valor por defecto para el *widget* cuando se inicializa. En algunos es una constante, mientras que en *widgets* de rango variable, como una caja de valor numérico, puede variar.

Los *widgets* tienen además otras propiedades para controlar su etiqueta, comentario emergente y complemento asociado, si hay alguno. El aspecto gráfico de los complementos, por otra parte, depende completamente del programador, que deberá además establecer un valor a su propiedad posicionamiento (por defecto, debajo) para controlar el posicionamiento en relación a su *widget* asociado.

Métodos

A diferencia de las propiedades, los métodos de *widgets* y complementos son prácticamente los mismos, ya que cada objeto particular emplea sobrecarga o extensión si las acciones a realizar son diferentes de las de su objeto padre. Los métodos más importantes son aquellos que trabajan sobre el valor del objeto: inicializar, validar, leer y escribir; estos dos últimos con tres variantes para trabajar con cadenas de texto, el tipo de la variable o el tipo interno del control que contiene el dato. Además todos los objetos tienen tres slots (ver punto siguiente), a los que se puede conectar cualquier otro objeto para activarlo/desactivarlo, redibujarlo o ajustarlo a la anchura del objeto padre.

Comunicación

Los *widgets* y los complementos disponen de dos mecanismos de comunicación que pueden ser combinados entre sí. La elección de uno, otro o ambos dependerá del número de objetos implicados y la visibilidad entre estos (Figura 10).

El primer mecanismo es una implementación en MATLAB del sistema de *signals* y *slots* empleado en Qt [W9]. Este sistema utiliza dos funciones, `connector` y `disconnector`[7] , para asociar una señal (evento) emitida por un objeto o un cambio en una de sus propiedades con un slot de otro objeto, que puede ser una función dedicada o una función anónima definida dentro de la declaración del conector.

El segundo método consiste en un bus de comunicación donde los objetos pueden publicar sus métodos y propiedades, o leer, llamar y conectarse a las de los demás.

[7] No se emplean las palabras *connect* y *disconnect*, usadas en Qt, debido a que ya existen en MATLAB funciones con dichos nombres.

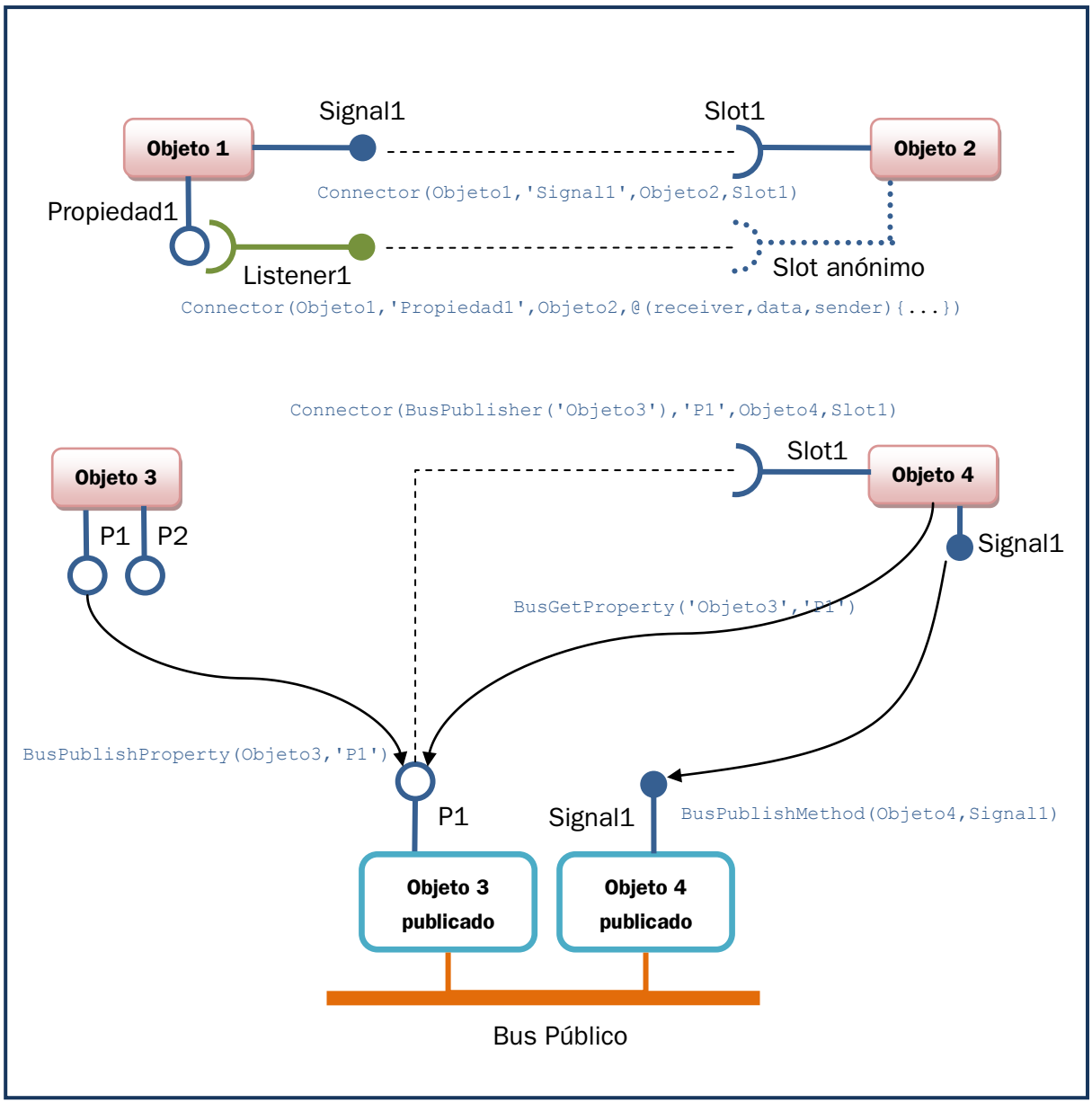


Figura 10 - Mecanismos de Comunicación entre Objetos: Ejemplo de combinaciones posibles

3

FACIL (II): Desarrollo y Aplicación

En este capítulo se describirá el desarrollo de FACIL, desde antes de constituirse como un proyecto separado, hasta la versión final y su aplicación a los proyectos GASIP y FCAT

Fase 1: Generación Dinámica del Panel de Entrada

Al inicio de esta fase se presentó al autor el proyecto FCAT y se planteó el problema del dibujado de las interfaces gráficas. La organización proponía la generación automática de un esqueleto de las mismas en tiempo de desarrollo para agilizar el proceso. El autor, por su parte, propuso la idea de que estas interfaces se generasen de forma dinámica en tiempo de ejecución, lo cual aportaría gran flexibilidad ante cambios en las variables de entrada. La idea fue acogida con interés y ambas posibilidades fueron estudiadas.

Formato de la Descripción de Variables

En el momento de la incorporación del autor al proyecto, existían, surgidos de un análisis anterior y con propósito de documentación, dos ficheros con una sintaxis similar a JSON [A2] especificando las variables de entrada de cada una de las herramientas de la aplicación.

Estos ficheros se tomaron como base para la generación de las interfaces, decidiendo qué partes de la información podíamos aprovechar y para qué, y realizando posteriormente ciertas normalizaciones (para facilitar su interpretación) y ampliaciones (para poder reflejar todas las necesidades del dibujado) a la sintaxis de los mismos (Figura 11).

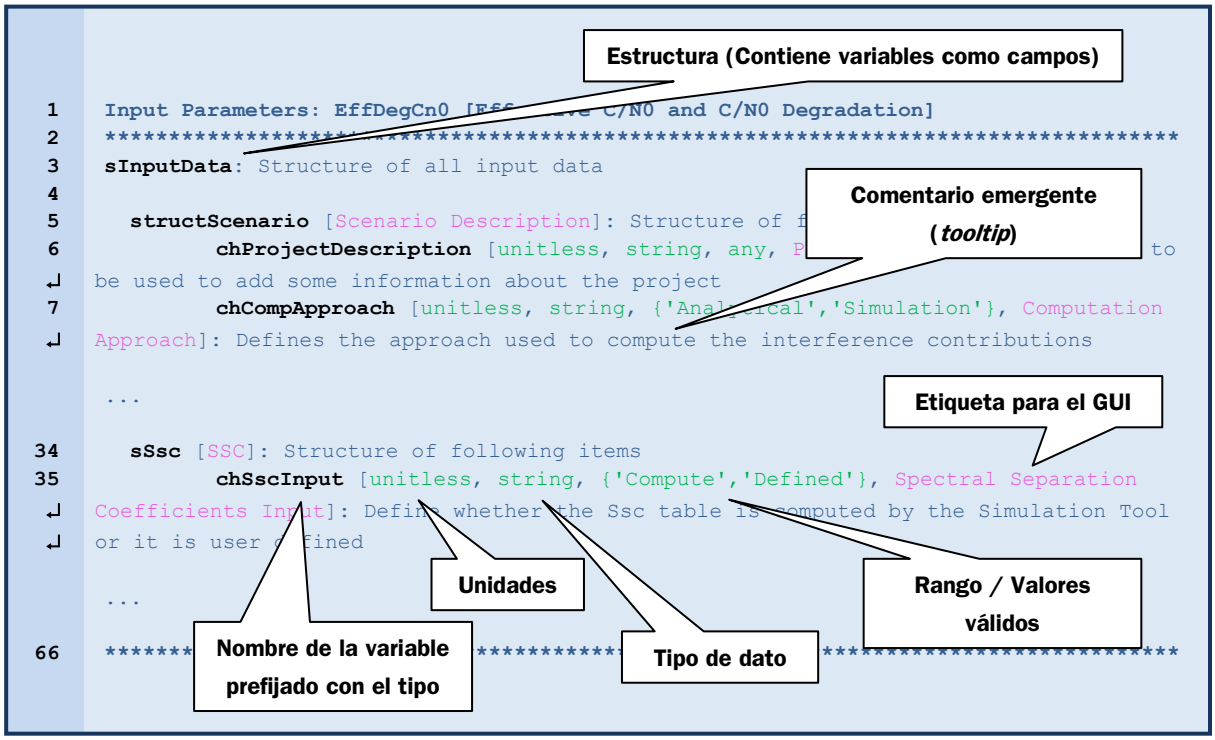


Figura 11 - Ejemplo de Fichero con las Descripciones de las Variables de Entrada

El Analizador

Desde el primer momento se decidió realizar el análisis de los ficheros de descripción de variables con expresiones regulares. Con el objeto de hacer más factible la idea de generación dinámica, se estudiaron las distintas posibilidades dando lugar a tres posibles soluciones:

S1. Usar el intérprete de Perl integrado en MATLAB

S2. Usar el analizador de expresiones regulares de MATLAB línea por línea

S3. Usar el analizador de expresiones regulares de MATLAB sobre el fichero completo y refinar la estructura de datos resultante.

Mientras que S1 presentaba problemas para el intercambio de estructuras complejas de información entre Perl y MATLAB, S3 podía resultar poco flexible ante futuras ampliaciones, así que se decidió implementar S2.

Prototipo de Generación Automática

Con el analizador completo se realizó un primer prototipo de generación de la interfaz. Se decidió que se crearía para cada variable de tipo estructura, un panel que contendría los controles correspondientes a cada uno de sus campos escalares (Figura 12). Esto se implementó mediante un sencillo bucle de dibujado, y para ello fue necesario estudiar sobre el diseño de interfaces gráficas en MATLAB [L9]. Tras comprobar que este prototipo se ejecutaba en un tiempo razonable, se tomó la decisión de abandonar la generación en tiempo de desarrollo a favor de la generación dinámica.

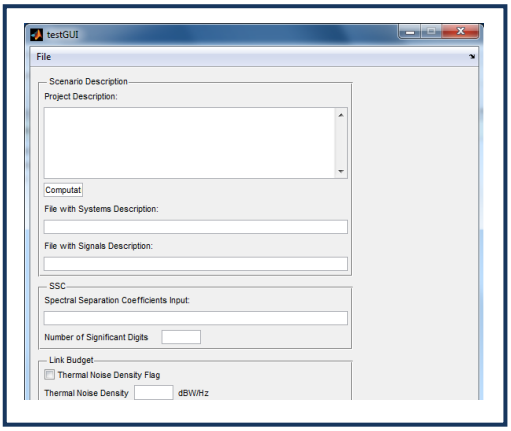


Figura 12 – Primera Interfaz Generada

Refinamiento del Panel de Entrada

En este punto se trataron los principales problemas existentes en la implementación del panel de entrada:

- Era necesario poder guardar y cargar los datos de entrada en ficheros de texto, por lo que se resolvieron los obstáculos presentes en el modelo de datos, se elaboró una

sintaxis sencilla de utilizar y se implementaron las funciones y menús correspondientes.

- Algunos controles debían ofrecer la posibilidad de realizar funciones adicionales (por ejemplo, visualizar el contenido de un fichero con un formato concreto). Se decidió, dada la difícil estandarización y la posibilidad de que surgieran nuevas necesidades de funcionalidad en este sentido, implementar un sistema de complementos. Dicho sistema permite aportar funcionalidad extra mediante un fichero fuente o una librería especificados en la descripción de la variable. Se implementó, mediante este sistema, el primer complemento para FCAT, consistente en la previsualización de las propiedades de un sistema de navegación (Figura 13).

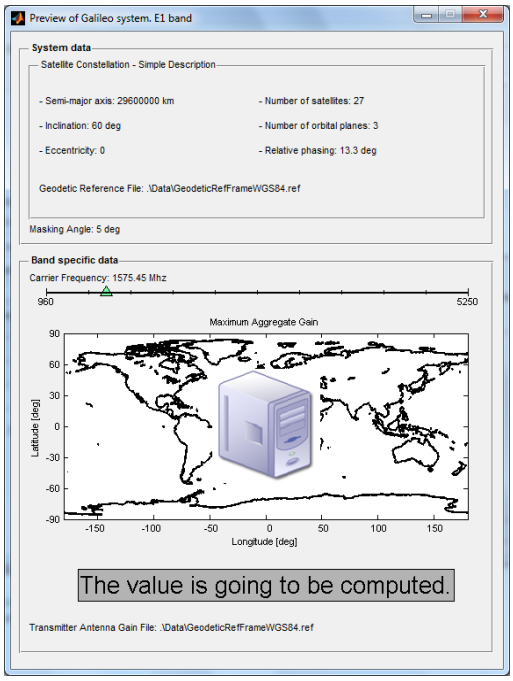


Figura 13 - Previsualización de Sistemas

- La longitud del panel de entrada se veía limitada por la altura máxima de una ventana. Se estudiaron distintas alternativas para mostrar los distintos paneles de forma exclusiva, como una barra de herramientas con botones o barras de pestañas a la izquierda o en la parte superior. Sin embargo, la decisión final fue adoptar un concepto visto en la interfaz de 3D Studio Max [W7], como alternativa mucho más adecuada para facilitar la localización y el trabajo con los paneles (Figura 14): Paneles de altura variables, cuya etiqueta de título permite ocultar el contenido, reduciéndolos a la altura del texto.

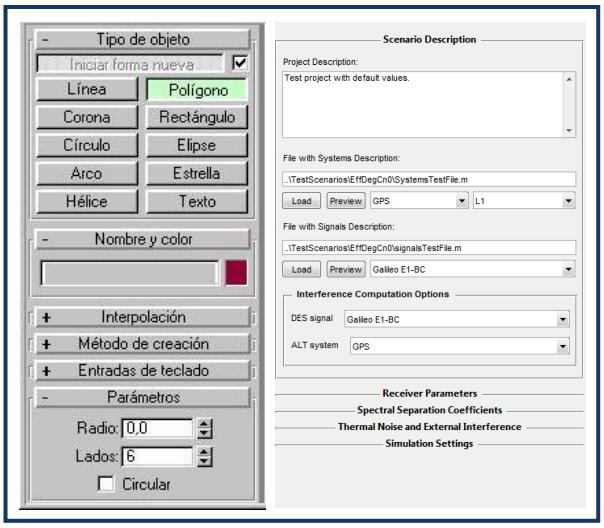


Figura 14 - Paneles de 3D Studio Max y FCAT

Fase 2: FACIL como *Framework* Independiente

Esta fase se inicia cuando, al observar las posibles aplicaciones de la generación dinámica de interfaces a otros proyectos, se decide separar la funcionalidad más general de la específica del proyecto FCAT. Este trabajo resultó bastante sencillo de realizar, requiriendo principalmente una reestructuración de los directorios y la creación de un fichero de parámetros[8] para almacenar algunos datos específicos de cada herramienta.

Abstracción de los controles: Campos

A medida que aumentaba la complejidad de los controles y los datos se hizo impracticable la conversión y chequeo de datos en las funciones de guardar y cargar. Es por esto que se decidió homogeneizar el tratamiento de los mismos mediante unos elementos gráficos a los que se llamó campos.

Un campo es, en esencia, un control o un panel con un conjunto de controles que representa un solo dato. Lo primero que se necesitaba era una forma de almacenar información arbitraria, además de su valor, en estos campos. Para ello se implementaron las funciones `getUd` y `setUd`, que hacían uso de la propiedad `UserData`, presente en todo objeto gráfico de MATLAB, para almacenar conjuntos clave-valor.

Usando este método se asociaron una serie de propiedades personalizadas a todos los campos con referencias a funciones que permitían activarlos y desactivarlos, o acceder a su valor o a una representación del mismo en formato cadena, sin necesidad de atender al tipo de datos o controles que contengan. Con este sistema fue relativamente sencillo implementar botones de opción y cuadros de lista para variables consistentes en un conjunto finito de cadenas.

Evolución del Sistema de Complementos

La implementación de los complementos cambió para funcionar de manera similar a los campos, empleando propiedades personalizadas para la comunicación con su campo asociado. Estos cambios facilitaron que se pudiera empezar a implementar dos nuevos complementos: el primero debía permitir la de edición de matrices bidimensionales con coeficientes de separación espectral (SSC) y el segundo la previsualización de ficheros de

[8] Ver estudio sobre el uso de constantes y variables globales en el anexo C.1.

señales GNNS. También se realizaron mejoras en el complemento de previsualización de sistemas.

Variables con Múltiples Niveles de Anidamiento

Al contemplar la posibilidad de aplicar el recién creado *framework* en otros proyectos, se vio la necesidad de soportar múltiples niveles de anidamiento en las variables de tipo estructura, caso que se da en el proyecto GASIP. Para ello, se extendió primer lugar la sintaxis de los ficheros de variables con llaves para agrupar los contenidos de cada elemento (por conformidad con JSON). Posteriormente, se implementó recursividad en el analizador, almacenando la información de las variables en una meta-estructura en forma de árbol (Figura 15).

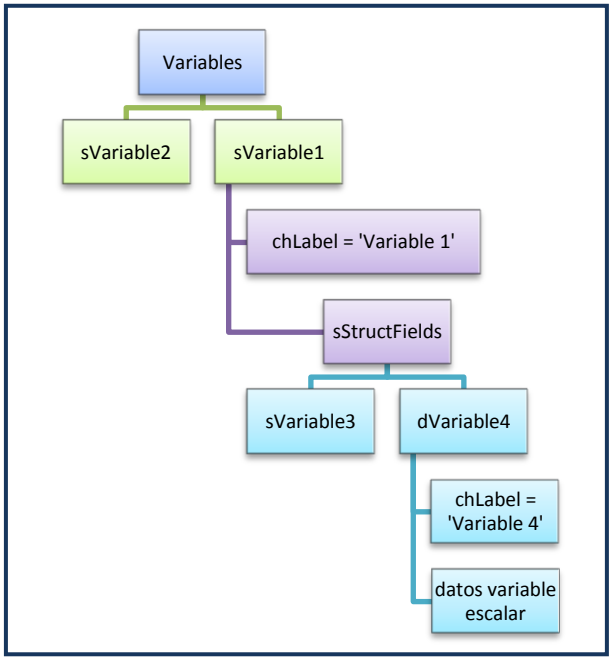


Figura 15 - Descripción de Variables

Esta nueva estructura precisó de cambios en el dibujado del panel de entrada, que pasó a funcionar también de forma recursiva empleando paneles normales para las estructuras de nivel dos en adelante. Con el objeto de hacer el código más sencillo de reutilizar, se separó el panel de entrada y sus pestañas como un módulo independiente. Además se implementó una barra de desplazamiento (una tarea no demasiado trivial en MATLAB [W8]) para el caso en que los paneles sean más largos que la ventana o sea interesante poder mantener más de un panel abierto.

También fueron necesarios cambios en la forma de almacenar los datos de entrada en ficheros de texto. Después de barajar diversas opciones se decidió usar XML [A3] por ser sencillo, de uso extendido y apropiado para representar estructuras de datos arbóreas.

Fase 3: El Panel de Salida

Esta fase del desarrollo completa la funcionalidad básica del *framework*. A lo largo de la misma se mejoró la documentación del código, se generó una aplicación de ejemplo y se depuraron diversos errores. Con el primer prototipo de las herramientas de FCAT implementado, se celebró una reunión con el cliente para su revisión, en la que éste expresó su satisfacción con el diseño de la interfaz. Además se utilizó FACIL para desarrollar un primer prototipo de una versión reducida del software GASIP, lo cual permitió identificar posibles mejoras a realizar.

Los Ficheros de Descripción de Funciones

Al igual que pasaba con las variables de entrada, las funciones de cálculo estaban definidas en ficheros destinados a la documentación del proyecto. Cada definición constaba de los nombres corto y largo de la función y opcionalmente de un número de parámetros de visualización, especificado por un entero. Dado que esta sintaxis resultaba bastante sencilla para la automatización, el único cambio sustancial que se realizó fue adaptar los parámetros de visualización a la de las variables de entrada. De esta forma se consiguió una sintaxis más sencilla y homogénea, además de ser posible reutilizar parte del analizador existente (Figura 16).

Las funciones pueden ser seleccionadas desde el panel de salida de la aplicación. Para ello se añadieron a la interfaz dos controles de lista desplegable que se muestran u ocultan en función de que exista una función, un grupo de funciones o múltiples grupos de funciones.

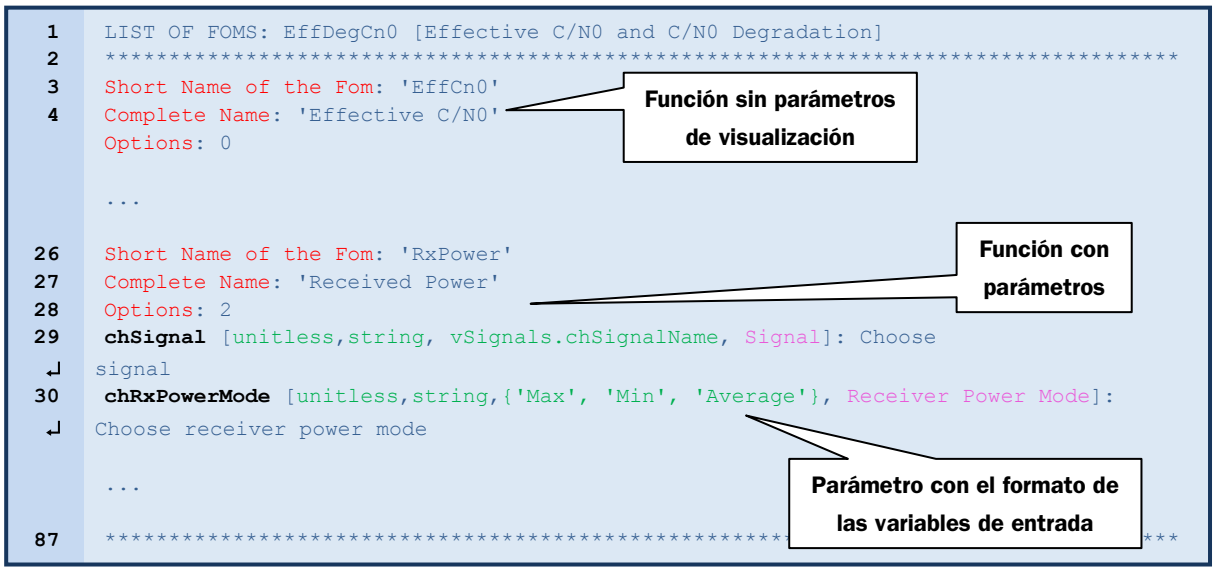


Figura 16 - Ejemplo de Fichero con las Descripciones de las Funciones de Cálculo

Interfaz con las Funciones de Cálculo

La interacción con las funciones de cálculo se definió en una fase de análisis anterior a la incorporación del autor al proyecto, y refinada con decisiones de diseño y requisitos en una fase posterior[9]. Esta se realiza en dos pasos: el cálculo propiamente dicho y su representación. Gracias a esta separación, podemos ejecutar el cálculo una sola vez, almacenar el resultado y luego llamar a la rutina de representación cada vez que la función en cuestión sea seleccionada en la lista desplegable.

En esta etapa, la configuración de parámetros visuales desde la función de representación, tales como la relación de aspecto de las gráficas, se realiza mediante las funciones `getUd` y `setUd`.

Visualización de Resultados Mediante Tablas

Al probar la interfaz gráfica con sus algoritmos de cálculo, uno de los miembros del equipo vio la necesidad de representar algunos de sus resultados en forma de tabla. Tras barajar diversas opciones, se decidió usar el elemento `uitable`, disponible en MATLAB. Lamentablemente, este elemento gráfico no está presente en la versión 2007a del software (la versión del cliente), por lo que fue necesario realizar una implementación propia para esta versión (Figura 17).

La implementación de este objeto se realizó de forma parcial en Java. De hecho, el modelo de la tabla y el renderizador para las celdas se implementaron en clases Java compiladas empleando Netbeans. La mayoría de los problemas de dibujado y limitaciones de representación existentes en MATLAB fueron resueltos, por lo que al final se decidió usar esta tabla también para las versiones más recientes.

ID	Health	Eccentricity	Time of A...	Orbital Inc...	Rate of Ri...
1	0.0	0.0	90145.1	0.95993109	0.0
2	0.0	0.0	90145.1	0.95993109	0.0
3	0.0	0.0	90145.1	0.95993109	0.0
4	0.0	0.0	90145.1	0.95993109	0.0
5	0.0	0.0	90145.1	0.95993109	0.0
6	0.0	0.0	90145.1	0.95993109	0.0
7	0.0	0.0	90145.1	0.95993109	0.0
8	0.0	0.0	90145.1	0.95993109	0.0

Figura 17 - Ejemplo de uso de la Tabla Implementada

Distribución de Elementos: Ventana Redimensionable y Vista Separada

Al distribuir los elementos (gráfica, tabla y cuadro de texto) en la interfaz, se observó que el tamaño de las gráficas podía ser, con frecuencia, demasiado pequeño. Por otra parte, era

[9] Puede consultarse la especificación de esta interfaz en el manual del programador, anexo B.2.

necesario encontrar una forma de distribuir los elementos dependiendo de la visibilidad e importancia de los mismos.

MATLAB no dispone de una manera sencilla de redimensionar ventanas cuando se utilizan unidades absolutas (en nuestro caso píxeles), por lo que fue necesario programar manualmente todos los cambios. Se resolvió fijar la anchura del panel de entrada a fin de dejar el máximo espacio posible a la visualización de los resultados. Por otra parte, se escribieron una serie de rutinas que calculan la altura óptima de cada elemento del panel de salida, con el objetivo de mostrar la mayor cantidad de información posible al tiempo que se mantiene la relación de aspecto en las gráficas (Figura 18).

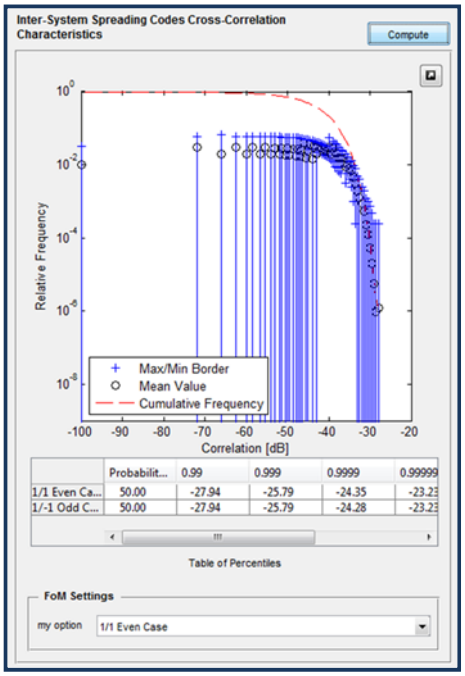


Figura 18 - Panel de salida con Todos sus Elementos Activos

Finalmente se implementó, tal y como especificaban los requisitos de FCAT, un pequeño botón que permite visualizar los resultados en una ventana independiente, empleando de este modo todo el espacio disponible en la pantalla.

Panel de Configuración

Para algunas funciones es necesario un panel de configuración que permita modificar las propiedades de visualización (por ejemplo, mostrar una gráfica lineal en lugar de logarítmica, o cambiar entre valores máximos y mínimos). Aprovechando el analizador existente, como ya se ha explicado, para la recogida de los parámetros y los campos usados en el panel de entrada para el dibujado, la implementación de este panel resultó prácticamente inmediata.

Fase 4: Implementación de *Widgets* como Objetos

Conforme la aplicación iba madurando se observó que era necesario hacer algunos ajustes para que los campos pudiesen generar y recibir eventos. Además, muchos de ellos compartían características comunes, lo que hacía que un cambio en la implementación tuviese que ser aplicado en muchas ocasiones a varios de ellos. Por todo esto, se decidió convertir dichos campos en clases, con herencia, métodos, propiedades, eventos y un protocolo de comunicación similar al sistema de *signals* y *slots* empleado en Qt [W9]. A partir de ahora, nos referiremos a las instancias de dichas clases como *widgets*.

Framework para Programación Orientada a Objetos en MATLAB

Si hay una cosa que caracteriza la programación orientada a objetos en MATLAB, sobre todo en versiones no muy recientes, es la poca homogeneidad y las múltiples opciones existentes, a falta de un consenso sobre el *framework* y la sintaxis a usar (incluso en el propio código fuente de MATLAB)[10]. Tras analizar las diversas opciones posibles, se optó por una sintaxis propia que aprovecha parte de la que se usa para las clases UDD, pero con modificaciones que permiten tomar el elemento `uipanel` como base para la herencia. Se han implementado con este propósito una serie de funciones que permiten:

- Definir métodos, sobrecargarlos y extenderlos.
- Definir un sistema de comunicación mediante *signals* y *slots* similar al de Qt.
- Definir un bus de comunicación pública, donde los objetos pueden leer y publicar propiedades y métodos, además de escuchar en busca de cambios. Este bus se ha implementado sobre la estructura de datos de aplicación, a la cual se puede acceder en MATLAB a través de cualquier objeto gráfico de la misma, evitando el uso de variables globales[11].

[10] Ver anexo C.2 donde se resume el estudio llevado a cabo sobre los diferentes sistemas de programación orientada a objetos en MATLAB.

[11] Ver anexo C.1 donde se reflexiona sobre el uso de variables globales en MATLAB.

Migración de los Campos a *Widgets*

La migración del código existente al nuevo sistema basado en objetos no resultó excesivamente compleja, siendo invertido gran parte del esfuerzo en organizar la jerarquía de los campos, paneles y complementos existentes (Figura 19).

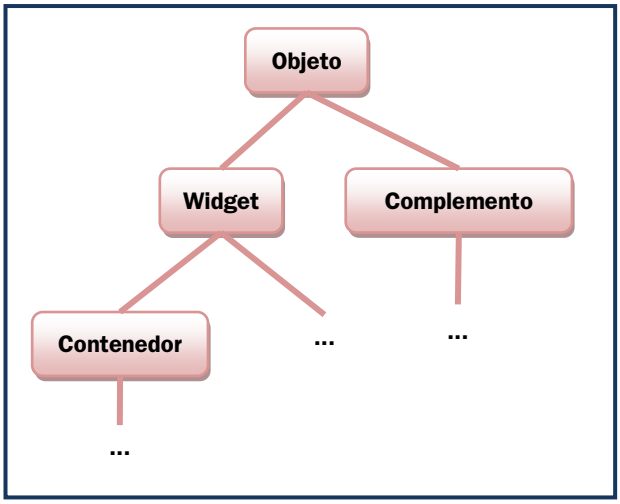


Figura 19 - Jerarquía Base de los *Widgets*

Como primer paso, se migraron todos los campos sin añadir funcionalidad adicional. Tras esto, se dividió el código del panel de entrada en dos clases de tipo contenedor, una de ellas aportando la funcionalidad de panel con barra de desplazamiento y la otra la de pestaña desplegable. Como paso final, se desarrolló una clase base para los complementos y se migró y completó la implementación de los mismos (Figura 20).

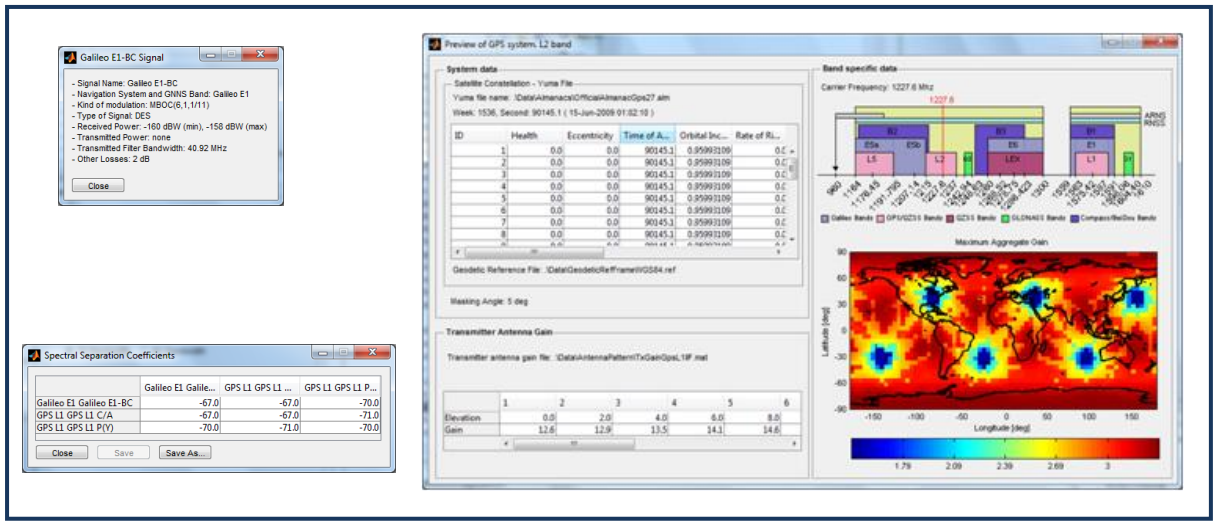


Figura 20 - Complementos de Edición y Previsualización Implementados para el Proyecto FCAT

Resultado

La implementación de *widgets* hubiera supuesto una pérdida de tiempo cuantiosa de no ser por las ventajas aportadas. Entre ellas podemos nombrar que se hizo posible terminar de implementar el panel de configuración en el panel de salida, además de añadir algunas características de comunicación al panel de entrada, que de otro modo hubieran provocado múltiples problemas durante su implementación. También se simplificó la interfaz con las funciones de representación, pues se pasó a emplear propiedades reales (y por tanto los métodos `get` y `set`) para la configuración.

Fase 5: Aplicación a los Proyectos GASIP y FCAT

En esta fase se completó la implementación de las dos herramientas de FCAT, así como la versión reducida de GASIP y se produjo la entrega al cliente de esta última. Para ello, además de terminar las labores de documentación y realizar una serie de baterías de pruebas, se completaron algunas funciones y se realizaron ciertas ampliaciones en FACIL.

Almacenamiento de Resultados

El principal aspecto de la aplicación, todavía pendiente de implementar, era el almacenamiento de los resultados calculados a fichero. Los formatos a emplear habían sido especificados en una fase temprana de análisis, y la mayor parte del código necesario para esta parte pudo obtenerse del proyecto GASIP.

Versión Reducida de GASIP

La versión reducida de la herramienta GASIP (Figura 21) está destinada a su distribución en formato ejecutable al público general a través de la página web de la Comisión Europea. Esto requiere una serie de medidas de seguridad adicionales que supusieron integrar algunas

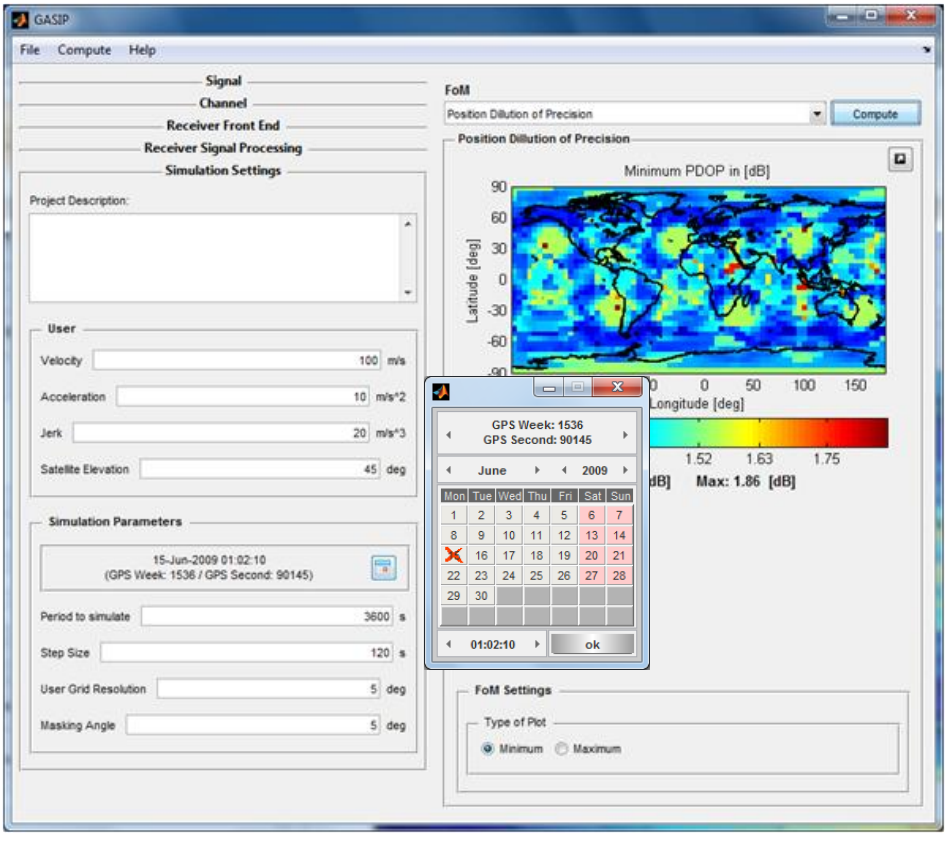


Figura 21 - Aplicación GASIP Implementada con FACIL

funciones nuevas en FACIL:

- Se añadió la posibilidad de definir constantes invisibles en el fichero de variables de entrada para poder fijar algunos parámetros que son variables en la versión completa.
- Se implementó la opción de cargar las descripciones de variables y funciones mediante ficheros MAT en lugar de ficheros de texto. De este modo, estos datos quedan encriptados y por tanto ocultos al usuario al generar una versión ejecutable del programa.

Además, mejorando el comportamiento que se presenta en la versión completa y aprovechando para ello la herencia presente en los *widgets*, se implementó un selector de fecha con un calendario desplegable en los formatos gregoriano y GPS. Para esto fue necesario añadir en FACIL la opción de implementar y cargar *widgets* específicos de la aplicación.

Herramientas de FCAT

De las dos herramientas a implementar para el proyecto FCAT (Figuras 22 y 23), solo una (la que calcula la correlación cruzada entre dos juegos de códigos PRN) fue completamente

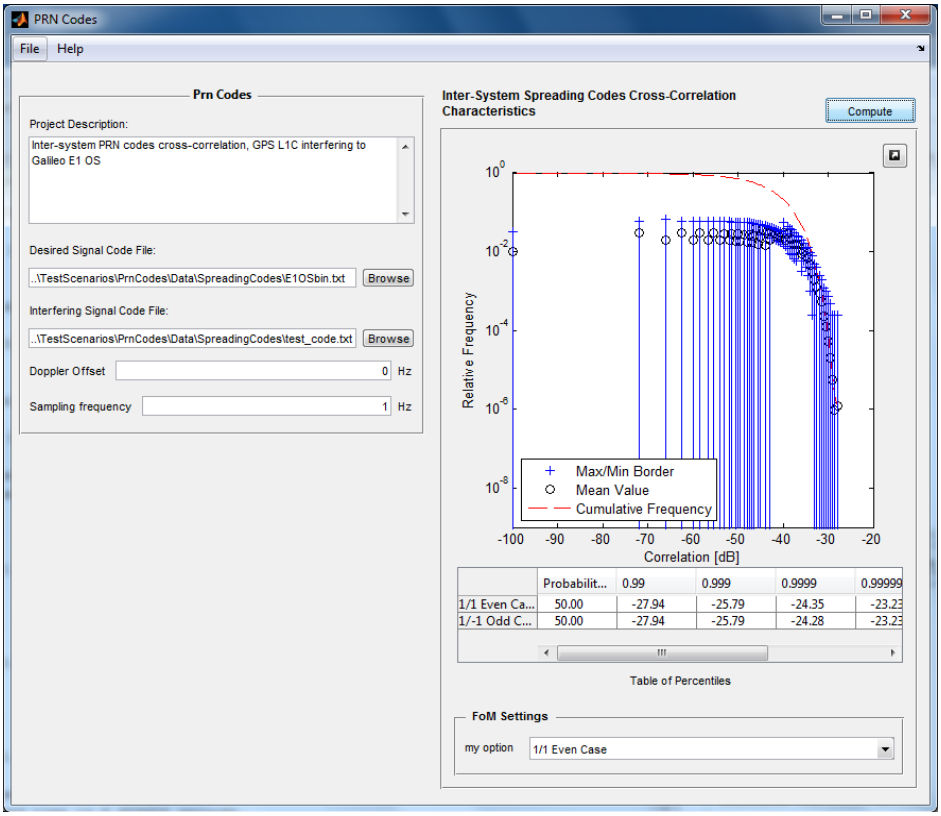


Figura 22 - Herramienta de cálculo de la correlación cruzada entre códigos PRN

terminada en esta fase, debido a que algunos de los algoritmos de cálculo de la segunda todavía no habían sido implementados por el miembro del equipo responsable de los mismos. Además de algunas conexiones que no hubieran sido posibles antes de la implementación de los *widgets,* se añadió la opción de poder insertar parámetros de visualización en el panel de salida desde una función de cálculo, en base a una petición de un miembro del equipo.

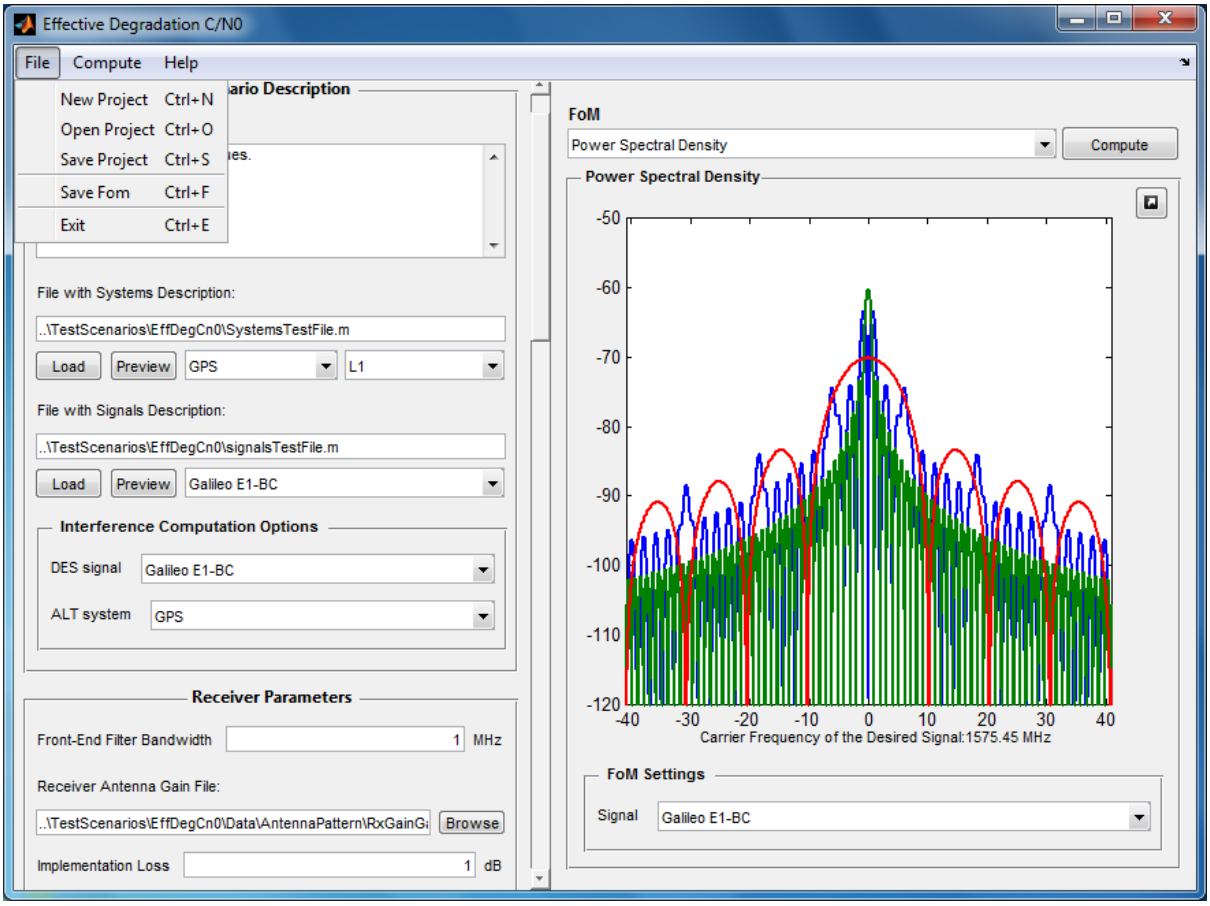


Figura 23 – Herramienta de métricas relacionadas con C/N_0

4

Optimizaciones

Este capítulo describe las posibilidades de optimización y las técnicas aplicadas a un algoritmo costoso en tiempo del proyecto FCAT.

Algoritmos a Optimizar

En el momento de realizar las optimizaciones, es decir, en la última etapa del autor en la empresa, solo se disponía de dos algoritmos de cálculo completos, el cálculo de la densidad espectral de potencia (PSD) de una señal y de la correlación cruzada entre los códigos pseudo-aleatorios (PRN) empleados por diferentes señales.

El primero de ellos no suele superar el segundo de duración, por lo que no se consideró invertir esfuerzos en el mismo. De este modo, la única opción posible fue estudiar las posibles optimizaciones aplicables al cálculo de la correlación cruzada entre códigos PRN apreciablemente costoso, para conjuntos de códigos relativamente pequeños.

Cálculo de la Correlación Cruzada entre Códigos PRN

Los códigos de ruido pseudo-aleatorio (PRN), en el contexto de la navegación vía satélite, son secuencias más o menos largas de chips[12] que se usan para identificar un satélite concreto. Para que esta identificación se lleve a cabo de manera satisfactoria es importante que la correlación de cada código con otros y con versiones desplazadas en el tiempo de sí mismo sea lo más baja posible [A4] [W10].

El algoritmo implementado por otro miembro del equipo y optimizado por el autor, calcula esta correlación cruzada entre dos conjuntos de códigos identificativos de dos señales de navegación distintas y condensa los resultados empleando percentiles de correlación. Dado que es necesario superponer todos los chips de todos los códigos para cada posible desplazamiento en el tiempo, y con la posibilidad de compresión y expansión de la señal por el efecto Doppler, el cálculo resulta intensivo tanto en tiempo de proceso como en memoria.

Localización del Cuello de Botella

Según la ley de Amdahl, la localización del cuello de botella de un algoritmo es fundamental para que el esfuerzo invertido en la mejora redunde en el mayor incremento de rendimiento posible. Para este algoritmo, la búsqueda se realizó tomando medidas de tiempos a lo largo del algoritmo. Como era de esperar, el cuello de botella se localizó en el núcleo del

[12] Un chip es en la práctica lo mismo que un bit, representado por un uno o un cero, la distinción de nombre se produce para remarcar que la señal transmitida no transporta datos.

algoritmo, el bucle en el que se calcula la correlación cruzada multiplicando la transformada de Fourier de cada código[13] para cada una de las combinaciones de bits posibles.

Vectorización

La parte a optimizar consistía originalmente en tres bucles anidados. Dado que MATLAB está diseñado y optimizado para trabajar con matrices [L11], una primera mejora podía obtenerse de la vectorización de estos tres bucles. De este modo, además se reduce el número de llamadas a función y se eliminan los retardos introducidos por la indexación.

Para esta tarea se trabajó desde el interior al exterior, comprobando en cada paso que el resultado del algoritmo seguía siendo el mismo que el original. Las funciones clave empleadas para llevar a cabo esta tarea son:

- Reshape: Permite modificar la geometría de la matriz.
- ShiftDim: Permite reordenar las dimensiones de la matriz.
- Bsxfun: permite aplicar una función escalar (en nuestro caso el producto) a los elementos de dos matrices, si una de ellas es más grande que la otra, la más pequeña se repite tanto como sea necesario. Esto es muy útil para vectorizar operaciones con matrices invariantes en un bucle.

Una vez realizada la vectorización[14] de los bucles se comprueba que se obtienen mejoras realizando mediciones experimentales. Sin embargo, las matrices generadas aumentan de tamaño rápidamente con el tamaño de los códigos de entrada, de forma que la mayoría de ficheros de los que disponemos provocan un error por falta de memoria. Esto no significa que el trabajo realizado no haya resultado útil, puesto que ahora podemos fragmentar nuestro código vectorizado en muchas menos iteraciones de un único bucle trabajando con fragmentos más grandes, consiguiendo todavía una mejora en el algoritmo.

Paralelización

Hoy en día prácticamente cualquier ordenador doméstico tiene dos o más núcleos en su procesador, e incluso la capacidad de ejecutar varios hilos por núcleo. Por ello se decidió

[13] Es posible explotar la transformada de Fourier para calcular la correlación cruzada como un mero producto [L10].

[14] Puede consultarse el código original y la versión vectorizada (solo el bucle correspondiente al cuello de botella) en los anexos D.1 y D.2.

explotar esta posibilidad paralelizando la ejecución del bucle vectorizado anteriormente[15]. Esta tarea resultó bastante sencilla empleando la sentencia `parfor` de la biblioteca de paralelización de MATLAB, dado que las iteraciones del bucle son independientes entre sí.

Por otra parte fue necesario elaborar una política de fragmentado eficiente. Al no fragmentar lo suficiente, se corre el riesgo de no emplear todos los núcleos o procesadores disponibles. Por otro lado, una excesiva fragmentación puede ser contraproducente al perder la mejora obtenida con la vectorización y añadir costes adicionales de comunicación. Además se comprobó que era preferible emplear un número de iteraciones que no fuese múltiplo del número de procesos a realizar una iteración residual, ya que MATLAB se encarga de repartir el trabajo pendiente entre los procesos que van quedando desocupados.

Resultados

Para la recogida de resultados se empleó la versión 2011 de 32 bits de MATLAB, bajo el sistema operativo Windows 7 en un equipo con un procesador intel Core Quad de cuatro núcleos y 4Gb (3 de forma efectiva) de memoria DDR2 en configuración de doble canal. Cada prueba ha consistido en una ejecución del algoritmo original, una del algoritmo vectorizado y diversas ejecuciones del algoritmo paralelo con diferente número de *workers* (procesos

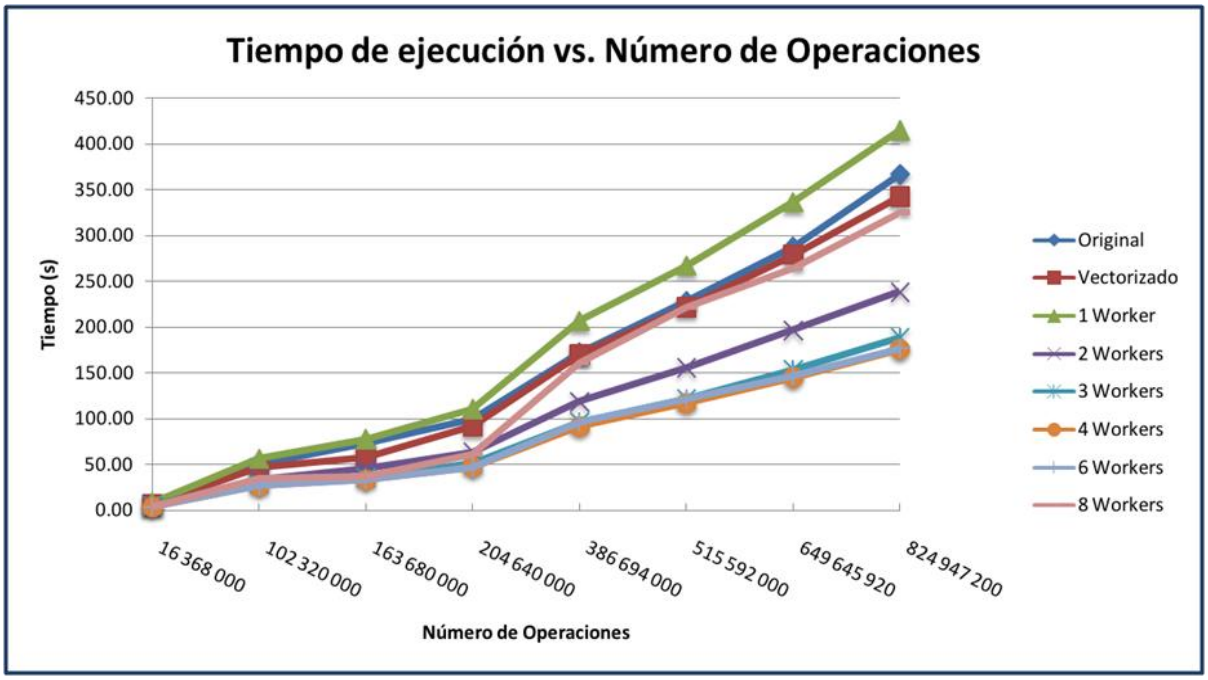


Figura 24 - Gráfica Comparativa del Tiempo de Ejecución frente al Número de Operaciones

[15] El código vectorizado incluyendo la política de fragmentado puede consultarse en el anexo D.3.

paralelos de MATLAB). Estas pruebas se han ejecutado diez veces calculando posteriormente el tiempo de ejecución medio.

En la Figura 24 se muestra el tiempo de ejecución de cada prueba para cada tamaño, medido en número de productos escalares total a realizar. Como se podía esperar, el tiempo de ejecución es lineal con el número de operaciones y no es posible que cambie simplemente con vectorizar o paralelizar. La conclusión directa de esta gráfica es que hemos mejorado el tiempo que se tarda en realizar una única operación sin modificar el orden del algoritmo. De este modo, los tiempos se mantendrán dentro de unos márgenes aceptables para volúmenes de datos mayores que con el algoritmo original, pero creciendo igualmente de forma lineal.

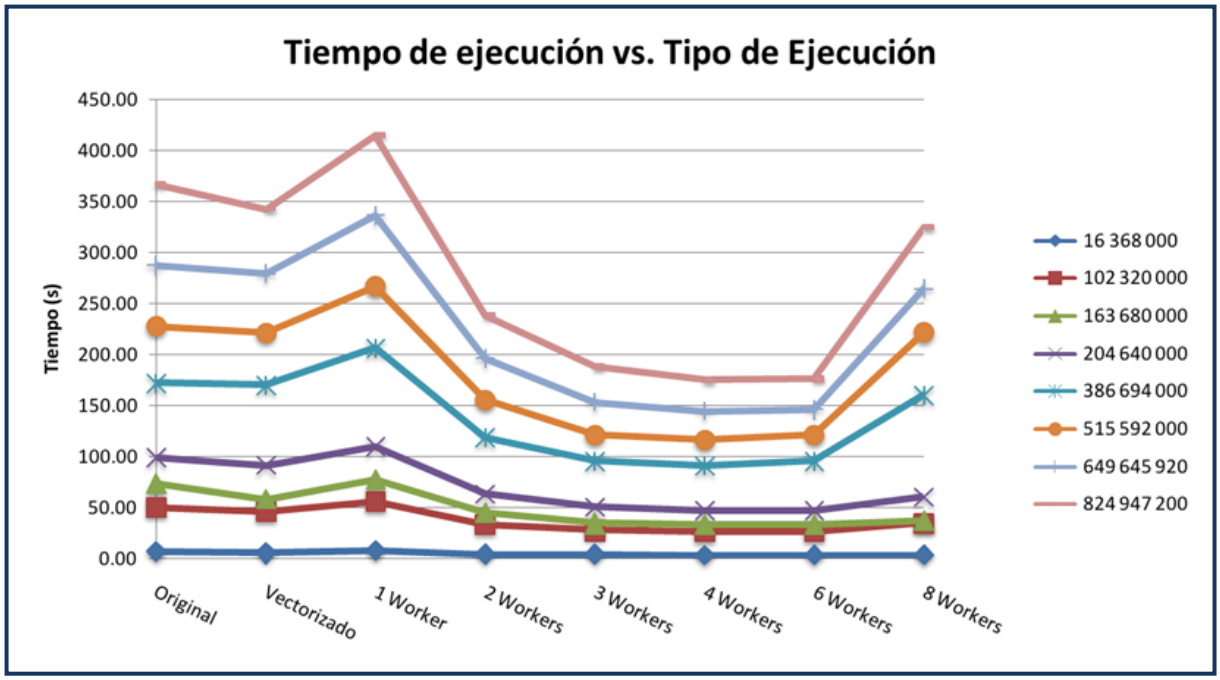


Figura 25 - Gráfica del Tiempo de Ejecución frente al Tipo de Prueba Ejecutada

Algo que se observa de forma más clara en la Figura 25 es otro comportamiento esperado: Los mejores tiempos se consiguen igualando el número de *workers* o procesos en ejecución con el número de núcleos de la máquina. También podemos observar cómo las tareas adicionales de la paralelización ralentizan el algoritmo si realmente no paralelizamos y solo empleamos un *worker*.

Si atendemos al *speedup* obtenido, (Figura 26) observamos que estamos lejos de alcanzar los valores ideales. Un detalle a resaltar es que a pesar de utilizar un procesador de cuatro núcleos, la mejora con tres y cuatro procesos es muy similar. Esto puede deberse a la existencia de algún proceso intensivo en CPU, como el antivirus o el propio proceso base

que se encarga de repartir las tareas entre los workers. También se observa cómo una vez superado el número de procesos que el hardware puede realmente ejecutar en paralelo, la

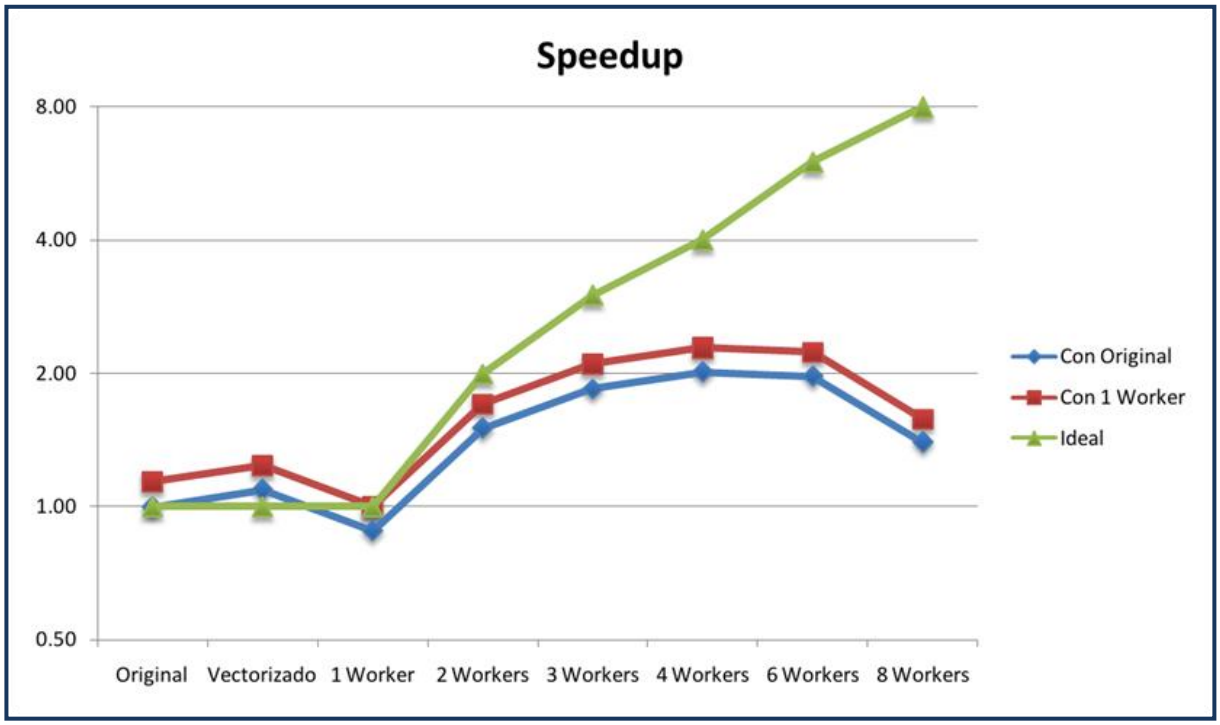


Figura 26 - Gráfica del Speedup

paralelización deja de suponer una ventaja introduciendo cada vez más retardos a causa de los cambios de contexto y el hecho de que parte de los procesos permanezcan inactivos.

Ideas

Varias ideas han quedado en el tintero por falta de tiempo o del material necesario. Por ejemplo, hubiese resultado bastante didáctico tener la oportunidad de ejecutar el algoritmo en un *cluster* para evaluar su escalabilidad. Por otra parte, las tarjetas gráficas abren muchas posibilidades como procesadores vectoriales para el cálculo de la transformada de Fourier, uno de los puntos débiles del algoritmo. Por tanto, hubiese sido interesante realizar pruebas de aceleración del algoritmo con una tarjeta gráfica Nvidia y el soporte integrado en MATLAB para realizar la transformada de Fourier con CUDA, o con unas librerías actualmente en desarrollo que exploran las posibilidades de las gráficas ATI con OpenCL .

5

Resultados y Conclusiones

En este capítulo se describen los resultados obtenidos, su correspondencia con los objetivos marcados y las conclusiones que se derivan de los mismos

Resultados

Como resultado general, podemos decir que se han cumplido, y en algunos aspectos rebasado, tanto los objetivos establecidos por el autor en su propuesta del proyecto, como aquellos establecidos por la organización en su propuesta de prácticas.

Proyecto FCAT

El proyecto FCAT es el desarrollo para el cual se contrató al autor inicialmente. Por tanto, resulta conveniente destacar que los resultados con respecto al mismo han sido positivos, puesto que, tanto el IGN como el cliente, la Comisión Europea, están satisfechos con el producto desarrollado.

Proyecto GASIP

En una fecha comprendida dentro del periodo en el que el autor participó en el proyecto FCAT, surgió la necesidad de entregar a la CE una versión reducida de la herramienta GASIP. Dado que ya se habían implementado todos los algoritmos de cálculo, y para acelerar el desarrollo de la interfaz gráfica, se decidió usar el *framework* desarrollado por el autor en este proyecto. Debido a esto, no solo se redujo el tiempo y coste de desarrollo, sino que además se añadieron nuevas funcionalidades a la aplicación sin coste adicional. Tanto el cliente como la propia organización han quedado satisfechos y la utilidad de FACIL, fuera del proyecto original, probada. La aplicación será puesta a disposición del público general en la página web de la Comisión Europea en un futuro cercano.

FACIL

Este *framework* que ha ido creciendo a lo largo de este proyecto final de carrera ha resultado ser una de las piezas clave, tanto desde el punto de vista académico como en vistas a la productividad. A lo largo de su desarrollo ha sido necesario profundizar en diversos aspectos del diseño de interfaces gráficas en MATLAB, programación orientada a objetos, aspectos poco documentados del lenguaje, o interacción con Java, por poner algunos ejemplos. Por supuesto, ha sido obligatorio mantener los estándares de calidad de la organización en materia de documentación, modularidad, usabilidad y corrección del código. Actualmente, FACIL está en camino de convertirse en el estándar para el desarrollo de interfaces gráficas en el IGN y hay al menos dos equipos de trabajo interesados en aplicarlo a sus proyectos.

Optimizaciones

A pesar del poco tiempo disponible para esta tarea debido a los plazos de entrega y la priorización de tareas, el trabajo realizado en la optimización del cálculo de la correlación cruzada entre códigos PRN ha dado sus frutos. La versión optimizada del algoritmo, tras superar una serie de pruebas de control, será probablemente incluida en la versión de FCAT que será entregada al cliente. Además, gracias al estudio realizado sobre las aplicaciones de la paralelización en MATLAB y a su documentación e implementación práctica, se ha abierto la puerta para que otros desarrolladores del IGN, en especial personal científico no familiarizado con algunos de estos conceptos, puedan examinar el trabajo realizado y aplicarlo a sus propias aplicaciones.

Conclusiones personales

El desarrollo de este proyecto ha estado plagado de buenos y malos momentos, de orgullo y de decepción, de momentos de trabajo relajado (los menos) y de carrera contra el reloj (los más). Un detalle interesante a remarcar es que el doble contexto, académico y profesional, del proyecto ha motivado la necesidad de buscar constantemente el compromiso entre viabilidad en términos de costes e interés académico, con el objeto de mantener el alcance del mismo dentro de unos márgenes aceptables para todas las partes implicadas y cumplir los plazos de entrega.

Aprendizaje

La realización de este proyecto ha conllevado un aprendizaje continuo, profundizando en gran medida en el diseño de interfaces en MATLAB, sus estructuras de datos y la programación orientada a objetos en este entorno. Por otra parte, también ha supuesto toda una experiencia en lo que respecta a las vicisitudes de la gestión de proyectos, documentación y el trabajo en equipo en una empresa real. Además, la naturaleza del proyecto ha llevado al autor a aprender sobre los diferentes sistemas de navegación vía satélite y su funcionamiento.

Dificultades surgidas

Como cualquier proyecto de ingeniería que se precie, el presente no ha estado exento de ciertas dificultades. Algunas de ellas derivadas del trabajo en empresa, como la exigencia de los plazos de entrega o el desacuerdo con otros miembros del equipo.

Otra dificultad importante a señalar estriba en el hecho de haber desarrollado una aplicación basada en un *framework* desarrollado al mismo tiempo. Por un lado, ha sido necesario mantener los módulos correspondientes a FACIL lo más depurados posible para evitar que un error o una regresión se manifestasen en las tres interfaces en desarrollo (GASIP y las dos herramientas de FCAT). Además, a la hora de incorporar una característica nueva, ha sido preciso analizar si ésta debía formar parte del *framework* o, por el contrario, debía implementarse de forma específica para la aplicación en cuestión.

Por último pero no menos importante, cabe nombrar el hecho de trabajar en un entorno excesivamente ligado, bajo la opinión del autor, al software propietario. Esto ha causado algunos problemas de disponibilidad de licencias y elección del sistema operativo, además

de limitaciones en las bibliotecas externas empleadas para el desarrollo, que debían estar licenciadas bajo una licencia BSD o, en general, una licencia compatible con el desarrollo de software privativo.

Interfaces gráficas en MATLAB

Sobre el diseño de interfaces gráficas en MATLAB hay mucho que decir. Quizá, en parte por las exigencias del propio desarrollo, no fue posible profundizar lo bastante en este tema al inicio del mismo. El diseño de interfaces gráficas en MATLAB sufre de dos grandes problemas: por un lado la implementación de los controles no es demasiado buena, son lentos y contienen diversos errores (alguno de los cuales ha sido necesario parchear durante este desarrollo y otros están documentados como insalvables); por otra parte, su existencia separada e incompatible con la jerarquía de objetos, descartando la herencia como una posibilidad, ha obligado al autor a implementar por sí mismo un *framework* (al margen de FACIL) que le aportase las posibilidades del trabajo con objetos que cualquier lenguaje de programación moderno ofrece.

Personalmente, y de tener que volver a realizar un desarrollo similar en el que sea indispensable implementar las funciones de cálculo usando MATLAB, el autor sería de la opinión de encaminar el proyecto hacia una de las dos siguientes alternativas:

- Desarrollar las funciones de cálculo en MATLAB, empaquetarlas en clases Java o ficheros MEX (código C) mediante el soporte incluido en el propio entorno y producir la interfaz gráfica de la aplicación mediante Java, Qt o alguna otra biblioteca gráfica para C++/C#.
- Desarrollar la interfaz gráfica en Java, posteriormente incluir la ruta de las clases en un script de MATLAB que se encargase de llamar a la interfaz y gestionar la interacción con las funciones de cálculo.

Metodología seguida

Sobre la metodología seguida, se puede decir que en general ha estado guiada por la directora del proyecto, si bien en muchas ocasiones el autor ha tenido carta blanca para organizar su trabajo de la manera que creyese más conveniente. Como suele suceder en proyectos de esta naturaleza, quizás haber dedicado algo más de tiempo a las fases de análisis y diseño en algunas iteraciones del proyecto hubiese ahorrado algunos problemas de implementación y posteriores cambios o correcciones. Por otra parte, el autor ve como

algo muy positivo el intento inicial de aplicar Scrum al proyecto, y la posterior adaptación y simplificación cuando se vio que no era factible, manteniendo de todos modos la forma de trabajo de las metodologías ágiles.

Inquietudes despertadas

Sin lugar a duda, MATLAB no es el mejor entorno ni el mejor lenguaje de programación para interfaces gráficas; sin embargo, ha demostrado ser una herramienta efectiva, no solo para el cálculo sino también para el trabajo con estructuras de datos dinámicas y métodos complejos de referencia e indización. Por todo esto, el autor no descarta emplear esta herramienta en un futuro (o explorar su alternativa libre, Octave, en caso de existir problemas de licencias) para la implementación de bibliotecas de cálculo o modelos de datos.

Por otra parte, el apartado de las optimizaciones, al que lamentablemente no ha podido dedicarse mucho tiempo, ha sembrado interés en un alumno que ya en su momento disfrutó con la asignatura Programación Paralela. Por ello no se descarta el acercamiento a la paralelización mediante tarjetas gráficas, de ser posible, en un proyecto futuro o a título personal.

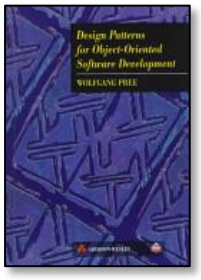
Como cierre a estas conclusiones, cabe decir, que si bien ya figuraba entre las ideas del autor la posibilidad de trabajar en el mundo de la investigación, las experiencias del último año de carrera, primero en el Instituto de Biocomputación y Física de Sistemas Complejos, y ahora en el IGN, han reforzado esta idea. Para el autor es importante que el futuro puesto que vaya a desempeñar, no solo requiera dedicación, rigor o eficiencia, sino que también sea un puesto en el que la creatividad, la iniciativa y las ideas sean valoradas positivamente.

Bibliografía

Libros

[L1] Design patterns for object-oriented software development

Wolfgang Pree, 1995

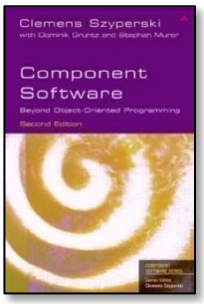


[L2] Component-Based Software Eng.: 9th International Symp., Västerås, Sweden, June 29-July 1, 2006 : Proc.

Ian Gorton, George T. Heineman, Ivica Crnkovic et al., 2006

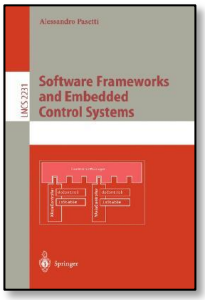
[L3] Component software: beyond object-oriented programming

Clemens Szyperski, Dominik Gruntz, Stephan Murer, 2002



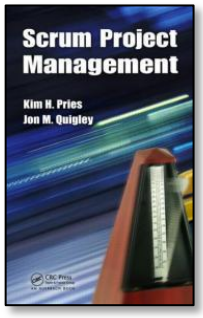
[L4] Software frameworks and embedded control systems

Alessandro Pasetti, 2001



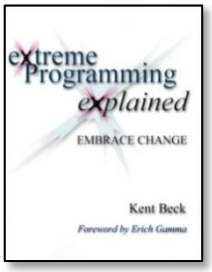
[L5] Scrum Project Management

Kim H. Pries, Jon M. Quigley, 2010



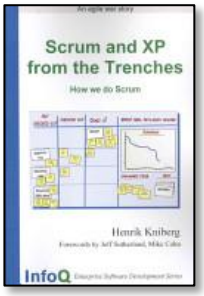
[L6] Extreme Programming Explained: embrace change

Kent Beck, 2001



[L7] Scrum and XP from the Trenches

Henrik Kniberg, 2007

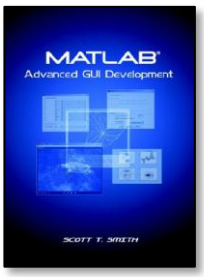


[L8] Instrumentación virtual: adquisición, procesado y análisis de señales

Antoni Mànuel, 2001

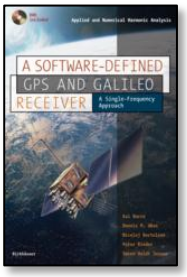
[L9] MATLAB: advanced GUI development

Scott T. Smith, 2006



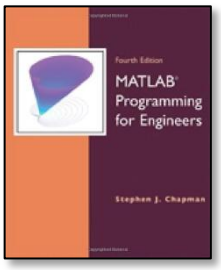
[L10] A software-defined GPS and Galileo receiver: a single-frequency approach

Kai Borre, Dennis M. Akos, Nicolaj Bertelsen et al., 2007



[L11] MATLAB programming for engineers

Stephen J. Chapman, 2008



Artículos

[A1] **GNSS Interoperability: Achieving a Global System of Systems or "Does Everything Have to Be the Same?"**
Günter Hein, 2006

[A2] **RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)**
Douglas Crockford, 2006

[A3] **Extensible Markup Language (XML) 1.0 (Fifth Edition)**
W3C, 2008

[A4] **An Analysis of L1-C/A Cross Correlation & Acquisition Effort in Weak Signal Environments**
Sana Ullah Qaisar, Andrew G Dempster, 2007

Otros Documentos

[D1] **Invitation to Tender n° ENTR/53/PP/ENT/SAT/10/4994**
Comisión Europea, 2010

[D2] **Código fuente de los scripts incluidos en la instalación de MATLAB 2007a**
Mathworks, 2007

[D3] **Código fuente de los scripts incluidos en la instalación de MATLAB 2010b**
Mathworks, 2010

Páginas Web

[W1] **Institute of Geodesy and Navigation**
http://ifen.bauv.unibw-muenchen.de/

[W2] **Introducing JSON**
http://www.json.org/

[W3] **Mathworks MATLAB**
http://www.mathworks.com/products/MATLAB/

[W4] **Oracle Java**
http://www.oracle.com/technetwork/java/

[W5] **Netbeans IDE**
http://netbeans.org/

[W6] **Perl**
http://www.perl.org/

[W7] **3D Studio Max**
http://www.autodesk.es/adsk/servlet/pc/index?siteID=455755&id=14626995

[W8] **Undocumented MATLAB: Continuous slider callback**
http://undocumentedMATLAB.com/blog/continuous-slider-callback/

[W9] **Qt Reference Documentation: Signals & Slots**

http://doc.qt.nokia.com/4.7/signalsandslots.html

[W10] **The GPS System**

http://www.kowoma.de/en/gps/

[W11] **IPT_ATI_PROJECT**

http://sites.google.com/site/iptatiproject/Home/fft