Estudio e implementación del software de control para la centralita electrónica de un coche autónomo

Andrés Caballero Toledo

Resumen– Desde que los sistemas de conducción autónoma son una realidad tecnológica, estos sistemas cuentan con un componente clave: un software de control de la centralita electrónica de conducción autónoma del coche. Esta centralita, presente en todos los coches de conducción autónoma, se encarga de gestionar todas las acciones del coche, procesándolas y haciéndolas llegar a los mecanismos físicos actuadores pertinentes para que el coche responda físicamente. El presente proyecto ha pretendido colaborar con el grupo ADAS-CVC en su Proyecto "Elektra", dedicado a la investigación tecnológica en un marco de desarrollo de un sistema de conducción autónoma, para reimplementar su componente software de control de la centralita electrónica, ahora siguiendo un adecuado proceso de ingeniería del software, proceso que no se había aplicado en su primer desarrollo.

Palabras clave– Elektra, ingeniería del software, sistema de conducción autónoma, software de control de la centralita electrónica, actuadores, sensores, reimplementación

Abstract– Since autonomous driving systems are a technological reality, these systems possess a key component: the autonomous driving electronic control unit software. The control unit, present in every autonomous driving car, is responsible for managing all car actions, processing and making them reach the proper physical actuation mecanisms for the car to respond physically. This project was aimed to collaborate with the ADAS-CVC group in its Project "Elektra", dedicated to the technological investigation within a framework of development of an autonomous driving system, in order to reimplement its electronic control unit component, but now following an appropriate software engineering process, process that wasn't applied in its first development.

Keywords– Elektra, software engineering, autonomous driving system, electronic control unit software, actuators, sensors, reimplementation

1 Introducción

El grupo ADAS *(Advanced Driver Assistance Systems)* del Centro de Visión por Computador de la Universidad Autónoma de Barcelona [1] lleva años participando, entre otros, en un proyecto propio llamado Proyecto "Elektra" [2]. Este proyecto consiste en la investigación en varios campos (principalmente de los sistemas de visión por computador y de algoritmos de control de rutas de desplazamiento), usando para ello el desarrollo de un sistema de conducción autónoma a instalar en un coche eléctrico real. El proyecto se inició hace años, pero, previo inicio del TFG, se encontraba algo parado a causa de una avería en la centralita electrónica de conducción autónoma del coche usado, elemento difícil de sustituir ya que estaba hecha a medida por una empresa que ya no existe. Además, algunos de los componentes del sistema carecieron de procesos formales de desarrollo, por lo que eran difíciles de mantener y seguir desarrollando. Afortunadamente, se consiguió una nueva centralita electrónica de conducción autónoma. Gracias a poseer una nueva centralita, con el presente TFG se ha pretendido colaborar con el Centro de Visión por Computador para replantear parte del Proyecto "Elektra" y ayudar a darle un nuevo impulso mediante dos vías centradas en uno de sus componentes esenciales.

1. La primera y principal, participando en el proceso de reimplementación, aplicando un eficiente proce-

• E-mail de contacto: andres.caballero@e-campus.uab.cat
• Mención realizada: Ingeniería del Software
• Trabajo autorizado por: Lluís Gesa Bote (DCC)
• Curso 2017/18

dimiento de ingeniería del software, del software de control de bajo nivel de la centralita electrónica de conducción autónoma del coche, conocido como ECUSW *(Electronic Control Unit Software)*, cuyo proceso de desarrollo se encontraba en pausa.

2. Y la segunda, proponiendo un nuevo enfoque para formalizar, de manera básica, el estado del ECUSW como miembro del Sistema "Elektra".

Como se verá en el punto 3, los coches de hoy en día poseen varias centralitas electrónicas [3]. Las centralitas son sistemas hardware encargados del control de un subsistema del automóvil. Como el TFG se centra en el software de control de la nueva centralita electrónica de conducción autónoma, a lo largo del documento se referencia como ECU (Electronic Control Unit) y se asume que, cuando se habla de centralita electrónica, se hace referencia a dicha centralita en la infraestructura del Sistema "Elektra".

1.1. Guía del documento

Este documento comienza con una introducción del motivo del TFG. Acto seguido, en la sección de objetivos se contextualiza el Proyecto "Elektra", necesario para comprender los objetivos definidos para el TFG, que se explican seguidamente. A continuación, la explicación del estado del arte define los conceptos e ideas más relevantes del contexto del TFG, y seguidamente metodología y planificación describen la dinámica de trabajo y fases seguida, además de los recursos considerados y la descomposición ordenada del trabajo en actividades. Posteriormente se explican, en orden, las fases de las que ha constado el desarrollo del TFG, describiendo, para cada una, su proceso. Finalmente se explican los resultados obtenidos en cada una de las fases, y se termina con las conclusiones del trabajo y las líneas futuras que deja abiertas.

2 OBJETIVOS

2.1. Proyecto Elektra

El Proyecto "Elektra" del grupo ADAS-CVC tiene como meta final el, mediante un proceso de desarrollo de un sistema de conducción autónoma que pueda ser instalado en un coche eléctrico, fijar una infraestructura de investigación del uso de nuevas tecnologías y componentes que puedan usarse en dichos sistemas para mejorar así el rendimiento de los sistemas actuales, conservando la funcionalidad final esperada.

El Sistema "Elektra" es entonces un sistema de conducción autónoma, compuesto por una serie de componentes y tecnologías, que es capaz de aprender, ver, procesar lo que ve y actuar en consecuencia de manera completamente autónoma y segura, siguiendo las normas de circulación vial sin necesidad de intervención humana. El software de control de la ECU del coche es uno de los componentes clave, y es precisamente éste último el que se ha tratado en el presente TFG.

2.2. Objetivos TFG - ECUSW

El ECUSW es el software de control de la ECU del coche, un componente software del Sistema "Elektra" encargado de procesar las órdenes producidas por un sistema de conducción inteligente y generar el correspondiente estímulo interpretable por los mecanismos físicos de actuación que hagan actuar al coche en consecuencia.

Este componente se encontraba parado a medio desarrollar, por lo que, en líneas generales, el presente TFG ha pretendido rehacer el software hasta el punto funcional en el que estaba, pero mejorado en cuanto a diseño e implementación, esta vez aplicando un adecuado proceso de ingeniería del software para tratar de garantizar un software de calidad con el que sea fácil retomar y continuar su desarrollo cómodamente.

A continuación, pues, se listan brevemente los objetivos del TFG junto a sus prioridades.

- TFG-OBJ-00 – Aplicar un proceso de ingeniería del software para rehacer el componente ECUSW siguiendo los cánones de metodología y calidad propios.
- TFG-OBJ-01 – Conocer cuál es y cómo funciona la infraestructura de entorno y física del ECUSW.
- TFG-OBJ-02 – Aplicar un proceso de estudio del estado de desarrollo del ECUSW.
- TFG-OBJ-03 – Analizar la idea, propósito, objetivos, contexto, entorno operativo, infraestructura, usos y requisitos del componente ECUSW.
- TFG-OBJ-04 – Rediseñar la arquitectura modular del ECUSW y los flujos de ejecución entre sus módulos en base al análisis realizado.
- TFG-OBJ-05 – Reimplementar, en base al análisis y diseño, el ECUSW considerando los requisitos funcionales que lo dejen el punto funcional en que se encontraba.
- TFG-OBJ-06 – Realizar casos de test que comprueben que el nuevo diseño y los casos de uso considerados en la implementación funcionan según se espera.

Sus prioridades se clasifican del siguiente modo:

- Crítica - Alta: TFG-OBJ-00
- Alta: TFG-OBJ-01, TFG-OBJ-02, TFG-OBJ-03, TFG-OBJ-04, TFG-OBJ-05, TFG-OBJ-06

TFG-OBJ-00, TFG-OBJ-01 son objetivos vivos a lo largo de todo el TFG, mientras que TFG-OBJ-02, TFG-OBJ-03, TFG-OBJ-04, TFG-OBJ-05, TFG-OBJ-06 son objetivos secuenciales, a cumplir en el orden de su nomenclatura según la planificación.

El TFG se enmarca en el ámbito de la mención en ingeniería del software, por lo que todos los objetivos se relacionan con aplicar un correcto proceso de ingeniería del software en el marco de redefinición del ECUSW. A pesar de que los objetivos de análisis (TFG-OBJ-03) y diseño (TFG-OBJ-04) contemplaban todos los casos de uso y requisitos funcionales contemplados para la versión final, los

objetivos de implementación (TFG-OBJ-05) y test (TFG-OBJ-06) pretendían reimplementar y testear sólo aquellos casos de uso y requisitos funcionales que ya se encontraban en la versión antigua (ECUSW en su estado pre-TFG).

3 Estado del arte

Desde 1987, cuando se acabó de desarrollar el primer prototipo de automóvil autónomo, el software de control de la centralita electrónica de conducción autónoma del coche ha tenido un papel fundamental en este tipo de sistemas.

Todas las grandes empresas de automoción con prototipos propios (Mercedes-Benz, Tesla, Google, Nvidia, etc.) han desarrollado sus propios sistemas de conducción autónoma, con su respectivo software de control de la centralita electrónica de conducción autónoma. Sin embargo, estos prototipos son de carácter privado, por lo que no están disponibles para ser consultados. Afortunadamente, debido a la importancia de fijar una arquitectura software común para el software de control de las centralitas electrónicas, en 2002 se fundó la asociación AUTOSAR (Automotive Open System Architecture) [4]. AUTOSAR es una asociación integrada por grandes empresas del sector (BMW Group, Ford, Volkswagen, etc.) cuyo principal objetivo es crear y establecer una arquitectura software estandarizada para software de control de centralitas electrónicas de cualquier automóvil. Un automóvil puede tener varias centralitas electrónicas (motor, transmisión, climatización, etc.); si el coche es autónomo, una de ellas se encarga de controlar la gestión de las acciones de autonomía, como es en el caso del Sistema "Elektra" y su ECU. Los estándares que promulgan están a disposición pública, por lo que son una buena referencia para cualquier organización interesada en desarrollar un software de control de la centralita electrónica de conducción autónoma de un coche.

Por otro lado, el Proyecto "Elektra" ya contaba con diversas versiones del software de control de la ECU del coche usado en la investigación. Primero contó con un software exclusivo para la anterior ECU estropeada (obsoleto ya; no tenía ningún tipo de documentación), después se obtuvo una aplicación para tablet de comunicación vía bluetooth con la nueva ECU (aplicación orientada a test funcional de integración), y, por último, la versión antigua del ECUSW para la ECU estropeada, a medio desarrollar sin seguir ningún tipo de proceso de ingeniería del software.

La idea básica general del estándar AUTOSAR y, principalmente, las tres versiones disponibles definen un punto clave en el objetivo principal del TFG de desarrollar nuevamente el ECUSW, ya que facilitan el análisis y diseño de la nueva versión del ECUSW, ayudando a captar la idea del software y a conocer qué se espera en cuanto a módulos básicos para la arquitectura, estados de ejecución, y funcionalidades principales.

4 Metodología

Una metodología Waterfall [5] con una aproximación a la metodología ágil SCRUM [6] fue la elegida para ejecutar todas las fases del proceso de ingeniería del software. SCRUM se enfoca a grupos, pero se puede adaptar al trabajo individual. Tiene tres elementos característicos: iteraciones de trabajo cortas, roles y elementos de seguimiento. La adaptación individual se ha aplicado así:

- Iteraciones: se han fijado sprints de trabajo semanales relacionados con las actividades correspondientes al periodo planificado.
- Roles: como sólo hay una persona, quien liderara el trabajo y la planificación (Scrum Master) y quien lleve a cabo el trabajo (Team) debían ser la misma persona; el representante del cliente (el Proyecto "Elektra") ha sido el tutor del TFG (Product Owner).
- Elementos: el Product Backlog lo han integrado los documentos SRS y de Visión; el Sprint Backlog ha sido el contenido gestionado semanalmente pensando qué era lo siguiente a hacer (Sprint Planning); Daily Sprint Meeting ha sido una evaluación personal para llevar un control diario del trabajo realizado y a realizar.

SCRUM se ha aplicado sobre un ciclo de desarrollo Waterfall correspondiente a ejecutar las fases de desarrollo de manera secuencial y ordenada.

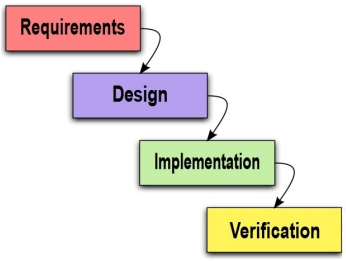


Fig. 1: Fases del ciclo de desarrollo

Se asumió que durante el desarrollo del proyecto se detectarían necesidades de cambios en detalles de los requisitos y/o diseño planteados inicialmente. Mediante un parcial solapamiento de fases contiguas, este ciclo se ha adaptado ligeramente para que contemplara poder actualizar los resultados de la fase anterior para poder seguir trabajando en la presente y así asumir mejor las necesidades analizadas para la versión final del software.

Se eligió esta metodología porque la adaptación SCRUM y Waterfall proporcionan un control a corto plazo y secuencial del avance que se va realizando, permitiendo controlar mejor cómo se van satisfaciendo las actividades y detectar antes necesidades de variación en la planificación o las necesidades del software, además de ayudar a mantener un ritmo de trabajo constante y ordenado.

5 Planificación

Para el proceso de redefinición del ECUSW, se planificaron el conjunto de fases estipuladas por la ingeniería del software, su orden, y se determinaron las actividades y subactividades a realizar.

5.1. Actividades Globales

Las actividades globales han sido aquellas comunes a todas las fases, vivas a lo largo de todo el proceso de desarrollo y enfocadas a satisfacer todos los objetivos del TFG.

- TFG-ACT-00 – Ingeniería del Software: Aplicar un proceso de ingeniería del software, con sus principios, fases, metodologías y buenas prácticas.
- TFG-ACT-01 – Documentación: Cumplimentar toda la documentación asociada al proceso de ingeniería del software y los documentos formales de seguimiento del TFG.
- TFG-ACT-02 – Microcontrolador: Obtener un conocimiento básico del funcionamiento del microcontrolador donde se carga el ECUSW.
- TFG-ACT-03 – Planificación: Definir objetivos del TFG, actividades a realizar en cada fase del desarrollo y metodología de trabajo a seguir.

5.2. Fases y Actividades Específicas

Por otro lado se estipularon las fases (todas satisfacen el objetivo TFG-OBJ-00) y, para cada una, su actividad relacionada y el conjunto de subactividades que mejor servían para cumplir con el objetivo intrínseco de la fase.

1. Pre-Análisis (TFG-ACT-04): Estudiar la infraestructura del Sistema "Elektra". Incluye subactividades: determinar propósito, abasto, estado del arte, stakeholders, usuarios y restricciones (TFG-ACT-04-01); observar el entorno operativo y físico del sistema y la contribución del ECUSW en él, infraestructura y problemas a satisfacer con la realización del TFG (TFG-ACT-04-02). Satisface objetivos: TFG-OBJ-01, TFG-OBJ-02.
2. Análisis (TFG-ACT-05): Tratar el pre-análisis desde la perspectiva del ECUSW para obtener una descripción general del componente y determinar sus aspectos funcionales. Incluye subactividades: obtener perspectiva, funcionalidades principales, relación con entorno y casos de uso (TFG-ACT-05-01); determinar requisitos funcionales (TFG-ACT-05-02); determinar requisitos no-funcionales (TFG-ACT-05-03). Satisface objetivos: TFG-OBJ-01, TFG-OBJ-02, TFG-OBJ-03.
3. Diseño (TFG-ACT-06): Reidear la arquitectura modular y los flujos de ejecución de la lógica interna del software a diversos niveles de abstracción. Incluye subactividades: determinar propiedades deseables de diseño según la finalidad (TFG-ACT-06-01); determinar la nueva división modular que mejor se adapta al entorno operativo (TFG-ACT-06-02); determinar los estados del sistema, dependencias modulares y flujos de ejecución (TFG-ACT-06-03). Satisface objetivos: TFG-OBJ-04.
4. Implementación (TFG-ACT-07): Reimplementar el software. Incluye subactividades: adaptar el código al nuevo diseño, añadiendo la lógica necesaria para seguir cumpliendo con los requisitos factibles (TFG-ACT-07-01); dotar al código de mayor calidad siguiendo buenas prácticas de programación (TFG-ACT-07-02). Satisface objetivos: TFG-OBJ-05.
5. Test (TFG-ACT-08): Comprobar el correcto funcionamiento del nuevo diseño y de los requisitos funcionales contemplados. Incluye subactividades: elaborar casos de test de diseño y funcionales (requisitos implementados) (TFG-ACT-08-01); ejecutar los tests, registrar resultado y corregir errores (TFG-ACT-08-02); probar el ECUSW en la infraestructura real (TFG-ACT-08-03). Satisface objetivos: TFG-OBJ-06.

La planificación de fechas inicial (diagrama de Gantt disponible en el punto A.1 del apéndice) varió ligeramente al trabajar con más productividad de lo estimado a partir del inicio de la fase de implementación, lo que permitió adelantar fechas, conservando mismo orden y dependencias. Sobre el calendario, la planificación de fechas actualizada se estableció así:

Fase	**Fechas**
Pre-análisis	10/09/17 - 15/09/17
Análisis	17/09/17 - 30/10/17
Diseño	01/10/17 - 30/10/17
Implementación	01/11/17 - 15/12/17
Test	20/11/17 - 10/01/17

5.3. Herramientas

En cuanto a herramientas usadas para facilitar el proceso de desarrollo, las más importantes han sido:

- Latex [7]: Sistema de composición de textos que hace uso del lenguaje TEX, lenguaje que permite separar contenido de formato. Usado para la documentación.
- Trello [8]: Herramienta de organización de proyectos que permite la creación y registro de tareas. Usada para gestionar la planificación acorde a la metodología.
- Github [9]: Herramienta de control de versiones donde se ha alojado el repositorio usado para guardar la documentación y los avances del código fuente.
- Code Composer Studio [10]: Entorno de Desarrollo Integrado de Texas Industries, fabricante del microcontrolador (ver el punto A.4 del apéndice) usado en la ECU, necesario para trabajar con el código (C/C++) y cargarlo en el microcontrolador.
- PlantUML [11]: Herramienta para el diseño de diagramas UML. Usada para generar los diagramas del diseño.

6 Desarrollo del TFG

En este punto se explica de manera concisa en qué ha consistido principalmente cada fase del proceso de ingeniería del software.

6.1. Pre-Análisis

Esta fase fue clave para asentar las bases del proyecto. Mediante un proceso de estudio consistente en consultar el estado del Sistema "Elektra" y del ECUSW (funcionalidades que satisfacía hasta el momento) y obtener feedback (principalmente del tutor del TFG) que permitiera generar una idea de la infraestructura, entorno operativo y funcionalidades principales, se pretendía reunir información

que contextualizara el ECUSW y ayudara a entender su funcionamiento individual y en el Sistema "Elektra".

La aportación más importante de este estudio es que ayudó a obtener una visión general del Sistema "Elektra", pudiendo recopilar el conjunto de objetivos del proyecto que facilitaran el entender el funcionamiento del sistema, así como el objetivo del ECUSW en su infraestructura. Los objetivos y saber qué tecnologías y qué componentes se debían usar facilitó el contextualizar el ECUSW, saber cuál era su aportación, su lugar en la infraestructura y su relación con los demás componentes.

6.1.1. ECUSW en el Sistema Elektra

El funcionamiento del Sistema "Elektra" es el siguiente: una serie de cámaras y sensores físicos conectados a un software de conducción inteligente (sistema de visión por computador) generan información que envían al mismo. El software de conducción inteligente, que se ejecuta en un PC central, procesa la información para generar, con el conocimiento que posee, acciones con las que el coche debe reaccionar ante una determinada situación. Estas acciones se envían, mediante una tecnología llamada CAN BUS *(Controller Area Network Bus)*, al microcontrolador de la ECU, que es donde se ejecuta el ECUSW.

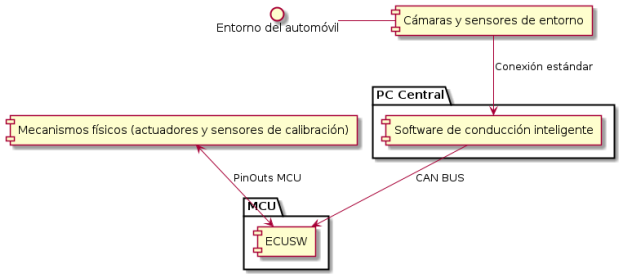


Fig. 2: Diagrama de componentes del Sistema "Elektra"

El ECUSW se encarga de contrastar de qué tipo son las acciones que recibe y convertirlas a señales eléctricas que se envían a través de los PinOuts del microcontrolador, conectados a mecanismos físicos de actuación que son capaces de comprenderlas y mover la pieza del coche a la que estén asociados según indique la señal. Además, existen sensores asociados a los actuadores que pueden enviar información del estado del actuador al ECUSW a través de los mismos PinOuts.

6.2. Análisis

Una vez estudiado el Sistema "Elektra" y contextualizado el ECUSW, se debía empezar a trabajar específicamente en el ECUSW. En base al pre-análisis y a consultar los recursos expuestos en el punto 3 fue posible recopilar información clave de cara a determinar su funcionamiento y funcionalidades básicas. En primer lugar se conoció la necesidad de usar dos tecnologías:

- CAN BUS [12]: Protocolo para comunicar diversos dispositivos a través de un único bus mediante mensajes identificados y con estructura común
- MCU-TIVA [13]: Microcontrolador que puede comunicarse con su entorno a través de un mecanismo de pines mediante señales eléctricas (PinOuts)

Y, en segundo lugar, se obtuvieron el conjunto de funciones básicas a realizar por el software:

- Controlar movimientos principales: Acelerar, frenar, girar, establecer velocidad
- Señalización: Controlar iluminación
- Inicio y configuración: Arrancar sistema, configurar sistema
- Lógica de control y restricciones: Arrancar sistema automático, arrancar lógica automática de control y error, ceder control manual

Este conjunto de funciones básicas establecieron los fundamentos de los casos de uso del sistema, cuya correspondencia es prácticamente directa. Y a raíz de estos casos de uso se pudieron extraer todos los requisitos del ECUSW. Estos requisitos describen íntegramente las funcionalidades que el ECUSW debe poseer para su versión final, así como los atributos de calidad y restricciones.

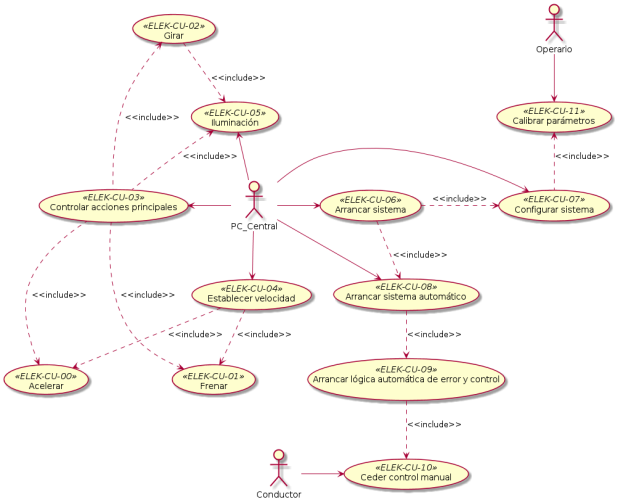


Fig. 3: Diagrama de casos de uso del ECUSW

6.2.1. Requisitos Funcionales

A continuación se muestran una serie de requisitos funcionales que representan, cada uno, un conjunto de requisitos mucho más específicos, pero que sirven para captar la idea de las funcionalidades consideradas.

ID	Descripción
RF-00	El ECUSW debe poder configurar los valores de aceleración, frenado, giro, velocidad e iluminación y accionar los actuadores conectados, respectivamente, al acelerador, frenos, volante, acelerador y emisores de luz en consecuencia.
RF-01	El ECUSW debe poder recibir valores de los sensores de calibración que proporcionan información sobre el estado de los actuadores.

RF-02	El ECUSW debe poder inicializar todos los actuadores, sensores, estados de ejecución del sistema y controladores de actuación del sistema, y fijar sus valores de inicio.
RF-03	El ECUSW debe poder manejar los estados de ejecución (del sistema) y los controladores de actuación del sistema (grupos de control de actuadores según contexto de acción).
RF-04	El ECUSW debe poder asegurar el correcto funcionamiento de la ejecución mediante la monitorización de las capacidades automatizadas de actuación usando rutinas automáticas.
RF-05	El ECUSW debe poder activar un procedimiento de emergencia en caso de que las rutinas automáticas detecten anomalías de funcionamiento.
RF-06	El ECUSW debe poder detener el automatismo y ceder el control manual del automóvil a un conductor en caso de indicárselo.

6.2.2. Requisitos no-funcionales

Y, a continuación, los requisitos no-funcionales considerados. Se han agrupado también en requisitos más abstractos para poder citarlos de manera concisa.

ID	Descripción
RNF-00	El ECUSW debe poder funcionar respetando las restricciones de rendimiento impuestas por los recursos de que dispone el microcontrolador y el uso de CAN BUS.
RNF-01	El ECUSW debe respetar unos requisitos de diseño. Principalmente: considerar propiedades hardware de subsistemas, separar lógica del funcionamiento hardware y usar patrones de diseño para adaptar arquitectura a infraestructura.
RNF-02	El ECUSW debe cumplir con una serie de capacidades de seguridad que garanticen la privacidad, uso y propiedad del código al grupo ADAS-CVC.
RNF-03	El ECUSW debe cumplir con los siguientes atributos de calidad software: disponibilidad, correctitud, flexibilidad, mantenibilidad, portabilidad, fiabilidad, robustez, modularidad y testeabilidad.

6.3. Diseño

Posterior al análisis, la etapa de diseño puede considerarse la más importante del TFG. Se considera así porque el ECUSW, antes de iniciar el proceso de ingeniería del software para rehacerlo, no siguió ningún proceso de ingeniería del software durante su desarrollo anterior, por lo que no constaba de ningún tipo de documentación que describiera el sistema ni sus funcionalidades, ni tampoco de ningún tipo de diagramas que describieran su arquitectura. Además, no se había estudiado apropiadamente su infraestructura ni entorno operativo, por lo que la adaptación del diseño a estos dos factores y a los requisitos no era la más acertada.

Por lo tanto, careciendo de todos estos recursos desde un principio, era imprescindible trabajar concienzudamente el nuevo diseño para que tuviera en total consideración los detalles del análisis para que las mejoras en el diseño fueran tangibles. Consecuentemente, la implementación mejoraría también al cambiar o añadir lógica que adecuara el código al nuevo diseño, siendo en un futuro un código mucho más fácil de entender, mantener y continuar implementando en base a las funcionalidades analizadas. La mejora del ECUSW, al no contemplar la reimplementación un incremento de las funcionalidades que ya poseía, recaía directamente en mejorar el diseño.

Acorde a lo establecido por un adecuado proceso de ingeniería del software, se ha usado UML (Unified Modeling Language) [14] como lenguaje gráfico para diseñar.

El proceso seguido para trabajar un nuevo diseño ha consistido en empezar por generar un diseño básico, un diagrama de clases de la nueva estructura, únicamente en base al análisis realizado. Una vez obtenido una primera versión del diseño, se debía repasar el código antiguo para generar su diagrama de clases y tratar de detectar todas buenas decisiones de diseño previas. Entonces, se consideraron las buenas decisiones de diseño previas para acabar de perfeccionar la versión final del nuevo diagrama de clases del diseño. Hecho esto, se generaron el resto de diagramas considerados clave para describir el software, dejando, con todo, la mejor referencia posible para la reimplementación del código.

6.3.1. Primera Versión

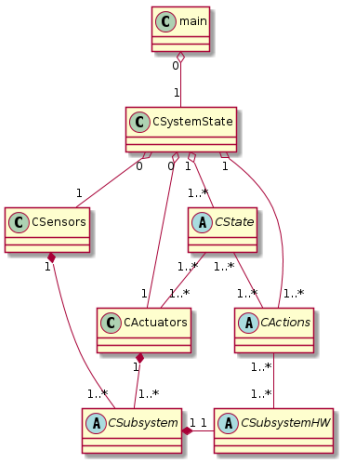


Fig. 4: Primera versión del diagrama de clases del ECUSW

El primer diagrama de clases básico obtenido únicamente en base al análisis fue el que se muestra en la figura 4. Como se realizó en base al análisis, este diagrama cumple con la idea principal de la arquitectura básica esperada para el software de control de la ECU de un coche, contemplando los módulos que se ha considerado que describen adecuadamente la infraestructura de un sistema de este tipo.

El diagrama de clases en su versión final puede encontrarse en el punto A.2 del apéndice.

6.3.2. Principios

Hecha la primera versión, en el código antiguo se detectó que el software no cumplía adecuadamente con los principios software deseables "Creador", "Alta cohesión", "Bajo acoplamiento" y "Polimorfismo". La inicialización era desestructurada, algunas clases tenían demasiadas responsabilidades y dependencias innecesarias, y no se aprovechaba el polimorfismo. El nuevo diseño contemplaba solventar esa serie de problemas.

6.3.3. Módulos Principales

En base a la primera versión y al diseño antiguo, se determinaron los siguientes módulos para la nueva arquitectura:

- Estados: Representan los estados de ejecución. En el antiguo estaba, pero faltaban estados por considerar y, los considerados, salvo el operativo, estaban a medias de implementar.
- Sistema: Representa el gestor de la ejecución encargado de la inicialización, gestión de input y la comunicación entre módulos. En el antiguo había un gestor semejante, pero, a pesar de pretender ser un gestor principal, no mediaba entre los módulos del sistema, por lo que había mucho acoplamiento entre módulos.
- Gestores de subsistemas: Representan a los encargados de conocer a los actuadores y sensores. En el antiguo no existía. Había algo que pretendía ser un nexo gestor de sensores y actuadores, pero no gestionaba nada (dando lugar a confusiones) sino que más bien representaban dos clases que ahora pertenecerían al nuevo módulo acciones.
- Subsistemas y Subsistemas Hardware): Representan los actuadores y sensores individuales, conteniendo sus propiedades hardware. En el antiguo no se hacía esta distinción separada, por lo que se considera un módulo nuevo, ya que no se separaba su lógica de funcionamiento de manejo a nivel hardware.
- Acciones: Representan los encargados de controlar configuración hardware tanto del microcontrolador como de la lógica de funcionamiento de acciones (la lógica debe saber manejar el hardware sin conocerlo directamente). En el antiguo no habían acciones como módulo, sino una especie de clases contenedoras de actuadores que no gestionaban hardware del microcontrolador ni lógica, sino directamente actuadores enteros (de los que dependían), y tenían demasiadas responsabilidades.
- CAN BUS: Representa la comunicación del software de conducción inteligente con el ECUSW. Sin cambios, ya que la implementación del procesado del contenido de un mensaje CAN no estaba aún implementada. Para superar este contratiempo se dispone de una consola que permite enviarle manualmente al software en ejecución comandos que representan el contenido de los mensajes CAN (acción a realizar) una vez se finalice el módulo.

6.3.4. Patrones

A raíz de los principios y durante la determinación de las finalidades de los módulos concretados, se resolvió que el uso de dos patrones mejoraría cualitativamente el diseño.

- Mediator: Definir un objeto central que coordine relaciones entre módulos para evitar fuertes acoples entre ellos. Debido a la necesidad de comunicación acciones-subsistemas hardware y estados-subsistemas-acciones se determinó la condición del módulo Sistema como gestor de ejecución para mediar entre todas esas comunicaciones.
- Template Method: Definir un esqueleto de algoritmo en una clase base, dejando a sus derivadas la implementación de aspectos de funcionamiento. Las condiciones de que un subsistema pudiera ser de varios tipos, el subsistema hardware que contiene pudiera ser de varios tipos, y las acciones pudieran ser de varios tipos, encajaba con el patrón.

6.3.5. Flujos de Ejecución

Conociendo los módulos del nuevo diseño y los patrones a aplicar, se elaboraron las principales relaciones modulares contempladas en la ejecución para los casos de uso considerados en la reimplementación, para lo que se elaboraron unos diagramas de secuencia que permitieran su visualización y comprensión.

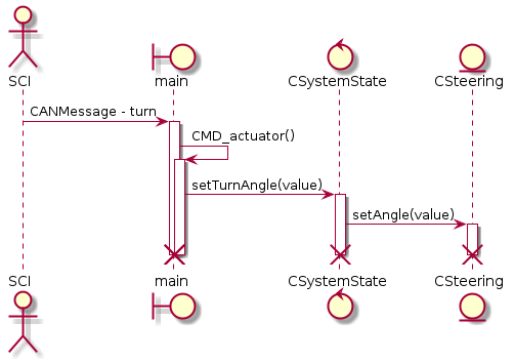


Fig. 5: Ejemplo de diagrama de secuencia del ECUSW

6.3.6. Estados

También se consideró necesario elaborar un diagrama de estados que representara el flujo de ejecución de estados considerado para el software. De este modo se facilita la comprensión del funcionamiento general del sistema, a pesar de que el estado trabajado en el TFG y que más tiempo de ejecución consume es el "operativo" (movimiento físico y control de acciones).

El diagrama de estados del ECUSW puede encontrarse en el punto A.3 del apéndice.

6.4. Implementación

6.4.1. Reimplementación Funcional

Tal y como se ha explicado en el punto 2.2, la reimplementación consideraba sólo aquellos casos de uso y requisi-

tos funcionales que ya se encontraban en la versión antigua. Algunos de los casos de uso considerados estaban completamente implementados; otros, parcialmente. A continuación se especifica qué casos de uso y respectivos requisitos funcionales ya constaban en la versión antigua y, por tanto, se han reimplementado adaptándose al nuevo diseño:

- Totalmente: RF-00, RF-01
- Parcialmente: RF-02, RF-03
- Sin implementar: RF-04, RF-05, RF-06

De manera sucinta, ello quiere decir que el ECUSW ya está preparado para realizar las acciones principales y recopilar datos de estado de los actuadores pudiendo activar todos los actuadores y sensores disponibles respectivamente. También significa que el ECUSW puede inicializarse completamente (salvando los estados que aún no están implementados) y que puede manejar todas las acciones disponibles, pero no puede controlar el flujo de ejecución de estados (por ahora, asume que siempre está en operativo). Por otro lado, el ECUSW aún no puede ni monitorizar las capacidades automatizadas del automóvil, ni activar procedimientos de emergencia, ni ceder el control manual de la conducción.

6.4.2. Adecuación al Nuevo Diseño

El aspecto clave de la reimplementación era adaptar el código al nuevo diseño modificando y/o añadiendo toda la lógica necesaria para cumplir él, conservando las funcionalidades que ya estaban implementadas. Los principales cambios en la lógica necesarios para cumplir con el nuevo diseño se explican a continuación, citando qué han aportado para mejorar la implementación del ECUSW.

1. **Secuencia de inicialización:** Para cumplir con el principio "Creador", los módulos deben hacerse cargo de inicializar los objetos que son responsabilidad suya, de manera que: Estados (no inicializa nada), Sistema (Estados, Gestores de Subsistemas, Acciones), Gestor de Sensores (Sensores), Gestor de Actuadores (Actuadores), Subsistemas - Actuadores y Sensores (Subsistemas Hardware), Acciones (Configuración Hardware del microcontrolador - cada acción la relativa a los PinOuts que gestiona).

   De esta manera se mejora en la creación, aumenta la cohesión y disminuye el acoplamiento, cuyos estados previos eran deficientes a causa de repartir por los módulos las inicializaciones de objetos que no les correspondían según su responsabilidad.

2. **Patrones Mediator y Template Method:** Dada la cualidad del módulo Sistema como gestor principal conocedor de Estados, Gestores de Subsistemas y Acciones, y la necesidad de desacoplar estos módulos, un patrón mediator era ideal para otorgar a Sistemas la responsabilidad de mediar las comunicaciones en los siguientes casos:

   - Acciones reciben el hardware de los subsistemas que deben manejar para realizar una acción
   - Estados deben recibir los subsistemas y acciones necesarias para dejar el sistema en un estado concreto según describa su lógica (aún sin implementar debido a la falta del flujo de ejecución de estados)

   Por otro lado, dada la cualidad de que los Subsistemas pueden ser Actuadores o Sensores, de que los Subsistemas Hardware pueden ser Actuadores-Tiva y Sensores-Tiva, y de que las Acciones pueden ser Frenar, Girar, Velocidad, Iluminación e Información-Sensores, el patrón Template Method servía adecuadamente para generar unas clases abstractas con la lógica principal a rellenar en cada clase derivada según la funcionalidad concreta que debía realizar en el contexto de su módulo.

   De este modo mejoran notablemente cohesión y acoplamiento, dotando de responsabilidades adecuadas a cada módulo.

3. **Iluminación e Información-Sensores:** La responsabilidad otorgada a las clases Iluminación e Información-Sensores en el módulo Acciones antes estaba desordenada por el código, controlando acciones de iluminación y obtención de información de sensores e inicializando el hardware del microcontrolador encargado de esas acciones en otros módulos no relacionados con la responsabilidad.

   De este modo, se mejora en cohesión, ya que dichas responsabilidades pasan a las clases a las que pertenece.

4. **Separar lógica - hardware:** Uno de los principales problemas que se encontraron fue que el ECUSW era demasiado estático en cuanto al microcontrolador donde se ejecutaba. El diseño anterior no contemplaba que el ECUSW pudiera ejecutarse en microcontroladores distintos, por lo que sus Subsistemas (por entonces distintos) siempre asumían el mismo tipo de propiedades hardware. Con los nuevos módulos Subsistemas, Subsistemas Hardware y Acciones, la lógica de control de Subsistemas, gestionada por Acciones, sólo necesita de los Subsistemas Hardware de los Subsistemas a controlar. De este modo, Acciones sólo necesita recibir Subsistemas Hardware para cualquier acción a realizar, independientemente de su tipo concreto, gracias al polimorfismo de los Subsistemas Hardware.

5. **Polimorfismo:** El polimorfismo supone una ventaja abismal en el caso de la separación lógica-hardware: otorga a cualquier clase del módulo Acciones la posibilidad de controlar un Subsistema Hardware sin necesidad de saber su tipo concreto, haciéndolo a través de un puntero de tipo clase base. Ello significa que si se cambia el microcontrolador a uno semejante (debe conservar los PinOuts, pines y puertos ya que son elementos clave en la infraestructura del sistema "Elektra"), aunque el manejo de sus propiedades

hardware (pines y puertos) sea distinto, se podrá manejar el Subsistema Hardware concreto a través de un puntero clase base (Subsistema Hardware) que apunte a un objeto de clase derivada que sí tenga definida la lógica de manejo hardware del Subsistema concreto.

De este modo se mejora tanto en cohesión, como en acoplamiento como en propiedades polimórficas, y el software pasa a ser mucho más flexible en términos del microcontrolador hardware donde se ejecute.

6. **Relocalización y calidad:** La reimplementación requería de las siguientes reestructuraciones asociadas a todos los cambios anteriores: (1) Añadir las clases necesarias presentes en el nuevo diseño, (2) reorganizar variables y funciones para que pertenezcan a las clases de quien son responsabilidad, (3) renombrar variables y funciones para autodocumentar código adecuadamente, (4) alterar lógica de las clases ya existentes que se han considerado para el nuevo diseño, (5) uso de los mecanismos "virtual" para clases abstractas y abstractas puras, (6) centralizar la gestión de sensores y actuadores en sus propios gestores, (7) creación de los estados aún no contemplados, (8) reorganización de variables globales y enums.

6.5. Test

Debido a la notable importancia del diseño en todo el proceso de reimplementación del ECUSW, se dedicó un especial esfuerzo a testear tanto el funcionamiento del nuevo diseño como el funcionamiento de las funcionalidades que se describían en los requisitos funcionales de los casos de uso implementados.

Generalmente, se ha optado por automatizar todo el test cuya automatización mejoraba la ejecución de los tests, resultando en realizarse más rápidamente y pudiendo igualmente comprobar el output de manera sencilla. Todos los casos de test de diseño se han automatizado. Sólo se ha optado por test manual en los casos de test de los casos de uso relacionados con las acciones físicas del coche, ya que su carácter manual otorgaba un control presencial del test que permitía que, al ejecutar los tests en la infraestructura real, se pudiera saber y ver exactamente qué se estaba haciendo, más que simplemente ejecutar una secuencia automática. Así, se introducían los comandos por consola (consola de comunicación UART con el ECUSW en ejecución) en los casos de testeo de requisitos y se observaba manualmente qué sucedía y si se ajustaba a lo esperado, además de, si se consideraba necesario una vez pasada la secuencia del test planificada, alterar ligeramente la secuencia para comprobar secuencias alternativas para el mismo objetivo de test.

En cuanto a lo testeado en test de diseño y test de requisitos, los testeos integraron las siguientes comprobaciones:

- Diseño: Módulos Acciones, Subsistemas Hardware, Subsistemas, Gestores de Subsistemas, Estados (sólo los existentes, cuya lógica aún no estaba acabada de implementar), Sistema, Input manual
- Requisitos: RF-00, RF-01, RF-02 (parcialmente), RF-03 (parcialmente)

Respecto a los tests de diseño, todos se han superado satisfactoriamente, certificando que el nuevo diseño se ha conseguido implementar adecuadamente, manteniendo el funcionamiento de los flujos de ejecución pero con una estructura nueva. Estos tests generalmente han consistido en comprobar que todas las clases de los módulos fueran accesibles según su especificación, que se inicializaran correctamente y que, si dependían de otras clases, se pudiera acceder a ellas.

En cuanto a los tests de requisitos, éstos han consistido en realizar los tests de las funcionalidades implementadas, tests que no se habían realizado en la versión antigua. Para ello, estos tests de requisitos, debido a su carácter manual, se han realizado en la infraestructura real del coche. En general, el funcionamiento del software es satisfactorio y todos los requisitos funcionales implementados o parcialmente implementados funcionan adecuadamente.

7 Resultados

Los resultados obtenidos están relacionados con conocimientos que se han adquirido a lo largo de la mención de Ingeniería del Software: análisis, diseño, implementación y test. Éstos han sido el fruto esperado de cada una de las fases de desarrollo del TFG y de sus respectivas actividades. Los resultados obtenidos son los que se esperaban en cada fase del ciclo de desarrollo, y son consecuencia directa de haber aplicado un adecuado proceso de ingeniería del software que define un marco de trabajo formal y estandarizado, y que determina qué resultados se espera obtener en cada fase. Un resultado complementario ha sido que haber aplicado un correcto proceso de ingeniería del software en un proyecto real ha servido para comprobar cómo funcionan los conocimientos adquiridos en la mención de Ingeniería del Software en una realidad de trabajo.

En el contexto del presente TFG, todas las fases contemplaban cumplir comúnmente el objetivo TFG-OBJ-00. Según la fase, los resultados obtenidos han sido:

- Pre-Análisis: Obtenidos un documento de planificación (objetivos TFG, metodología, recursos y actividades) y un documento de visión (contexto del Sistema "Elektra", estado del arte, entorno, infraestructura y función del ECUSW). Todo el estudio realizado y su información, reunida en ambos documentos, satisfacen los objetivos: TFG-OBJ-01, TFG-OBJ-02. Se ha obtenido lo esperado de una fase de pre-análisis: contextualizar y comprender el software.
- Análisis: Obtenido un documento de especificación de requisitos (descripción general del ECUSW (perspectiva, funciones, casos de uso y entorno operativo), requisitos funcionales y requisitos no-funcionales). El análisis del ECUSW, cuya información forma el documento, satisface el objetivo TFG-OBJ-03. Se ha obtenido lo esperado en la fase de análisis: conocer el entorno operativo, las funciones, casos de uso y requisitos del software.

- Diseño: Obtenido un documento de diseño (propiedades deseables, principios y patrones del software, arquitectura modular y flujos de ejecución). El diseño del ECUSW, plasmado en el documento, satisface el objetivo TFG-OBJ-04. Se ha obtenido lo esperado en la fase de diseño: construir la arquitectura modular y sus flujos de ejecución, dándoles sentido en base a principios, patrones y buenas prácticas.

- Implementación: Obtenido el código fuente reimplementado del ECUSW, cuyos principales cambios se han comentado en el punto 6.4. El código fuente reimplementado hasta el estado funcional de la versión antigua satisface el objetivo TFG-OBJ-05. Se ha obtenido lo esperado en la fase de implementación: una reimplementación del código fuente en base al nuevo diseño, cumpliendo con los casos de uso y requisitos que se encontraban ya implementados en la versión antigua.

- Test: Obtenidos el código fuente del test de inspección automatizado del diseño y el documento de test (definición de los casos de test de diseño de cada módulo y de los casos de test de requisitos de los casos de uso (y respectivos requisitos) implementados). El código y documento de test obtenidos satisfacen el objetivo TFG-OBJ-06. Se ha obtenido lo esperado en la fase de test: un test de inspección automatizado que corrobore el buen funcionamiento del nuevo diseño y comprobar que las funcionalidades que ya estaban implementadas funcionaban adecuadamente en la infraestructura real.

8 Conclusiones y Líneas Futuras

Tal y como se ha comentado a lo largo del documento, este TFG ha consistido en realizar un correcto proceso de ingeniería del software, con sus respectivos cánones, fases y resultados esperados, con el objetivo de rehacer un software hasta el punto funcional en que se encontraba, software cuyo desarrollo se encontraba parado en el momento y que no había sido desarrollado siguiendo un proceso de ingeniería del software, por lo que no se habían aprovechado los recursos que éste ofrece.

En perspectiva, pues, podría considerarse que se ha desarrollado un software desde cero siguiendo un adecuado proceso de ingeniería del software, pero poseyendo una base funcional como referencia del punto hasta el que se debía llegar en el nuevo desarrollo. Así, se ha conservado el funcionamiento de la versión antigua del software, pero habiendo ahora generado una importante base documental que describe íntegramente el análisis, diseño, implementación y test realizados desde cero siguiendo un correcto proceso de ingeniería del software que ha proporcionado principios, metodologías y buenas prácticas que han garantizado gestionar adecuadamente el desarrollo y conseguir hacer el software más completo, entendible, mantenible y fácil de continuar desarrollando.

Por lo tanto, el coger un proyecto con su desarrollo parado, sin bases sólidas que lo describieran, y rehacerlo, aplicando desde cero un adecuado proceso de ingeniería del software, era lo que el software necesitaba para poder impulsar el continuar con su desarrollo. Tener que retomar un proyecto difícil de entender y en el que no queda ninguno de sus miembros originales es un motivo que presupone factible el abandonar el proyecto, ya que la inversión de tiempo para entenderlo puede no compensar los resultados esperados. Así, llegar a conseguir rehacerlo y generar sus bases documentales son dos aspectos que han incrementado la convicción del alumno de lo importante que es que todo desarrollo de un proyecto software siga un buen proceso de ingeniería del software para, no sólo servir como base para actuales y futuros miembros del proyecto, sino también para garantizar que el desarrollo sigue un desarrollo bien definido que facilita la correcta evolución del proyecto y la obtención de los resultados esperados.

En cuanto a los objetivos del TFG, se han satisfecho todos. Al conseguir aplicar correctamente el proceso de ingeniería del software y rehacer el software hasta el punto funcional en que se encontraba, se han cumplido todos los objetivos, ya que estaban estrechamente relacionados con aplicar el proceso de ingeniería del software y obtener todos los resultados que se esperan de cada una de las fases.

De cara al futuro, el software todavía es muy explotable. A partir de la base generada por el presente TFG, el desarrollo del ECUSW puede seguirse por diversos caminos hasta darse por completado.

- El módulo CAN BUS está desarrollado, pero aún no hay comunicación real entre el Sistema de Conducción Inteligente y el ECUSW, la cual se simula manualmente. El ECUSW es capaz de recibir mensajes CAN, pero como la nueva ECU aún no tiene el conector CAN, no se ha podido cubrir ni testear.

- Los estados de ejecución aún deben trabajarse: se han añadido los que no se habían considerado, pero, aún así, los que ya existían no están acabados de implementar (al parecer no se sabía bien qué debía hacer cada uno, cosa que ya se proporciona, ya que se investigó en la etapa análisis del presente TFG) y el flujo de control de estados no está implementado (ahora se asume todo el rato el estado "operativo").

- Considerando el análisis obtenido, aún quedan funcionalidades por implementar y corregir las que se han probado con comportamiento erróneo en la fase de test.

Agradecimientos

Me gustaría agradecerle a mi tutor, Lluís Gesa Bote, todo el apoyo que me ha ofrecido durante todo el TFG, estando siempre disponible para ayudarme y siendo, para mí, un ejemplo a seguir. También me gustaría agradecérselo a toda mi familia, que me ha apoyado y animado incondicionalmente a lo largo de toda la carrera.

Me gustaría también dedicarle el TFG a mi madre por todos los sacrificios que, a lo largo de mi vida, ha hecho y sigue haciendo para que yo haya podido estudiar.

9 BIBLIOGRAFÍA

REFERENCIAS

[1] Advanced Driving Assistance Systems - CVC, UAB [Online]. Disponible: http://adas.cvc.uab.es/

[2] Elektra Autonomous Vehicle developed by CVC, UAB [Online]. Disponible: http://adas.cvc.uab.es/elektra/

[3] Electronic Control Unit [Online]. Disponible: https://en.wikipedia.org/wiki/Electronic_ control_ unit

[4] AUTOSAR, enabling continuous innovations [Online]. Disponible: https://www.autosar.org/

[5] Waterfall Model [Online]. Disponible: https://en.wikipedia.org/wiki/Waterfall_ model

[6] Scrum en pocas palabras [Online]. Disponible: https://cristinaramosvega.com/scrum-pocas-palabras/

[7] The Latex Project [Online]. Disponible: https://www.latex-project.org/

[8] Trello [Online]. Disponible: https://trello.com/

[9] Github [Online]. Disponible: https://github.com/

[10] Code Composer Studio, Integrated Development Environment [Online]. Disponible: http://www.ti.com/tool/CCSTUDIO

[11] PlantUML [Online]. Disponible: http://plantuml.com/

[12] Automotive CAN BUS System Explained [Online]. Disponible: https://www.linkedin.com/pulse/automotive-can-bus-system-explained-kiril-mucevski

[13] ARM Cortex-M4F Based MCU TM4C123G LaunchPad Evaluation Kit [Online]. Disponible: http://www.ti.com/tool/EK-TM4C123GXL

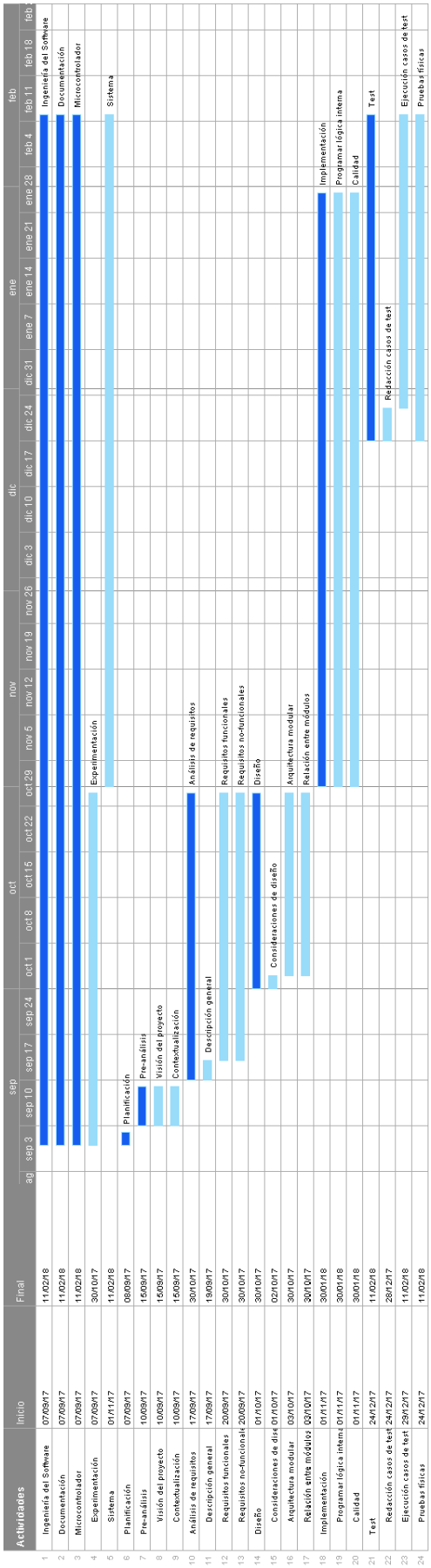
[14] Unified Modeling Language [Online]. Disponible: http://www.uml.org/

[15] ”Metodologies i recursos per a la gestió i desenvolupament de projectes”(2017, septiembre), documento de la asignatura común ”Gestión de proyectos”, Escuela de Ingeniería, Universidad Autónoma de Barcelona, 2016.

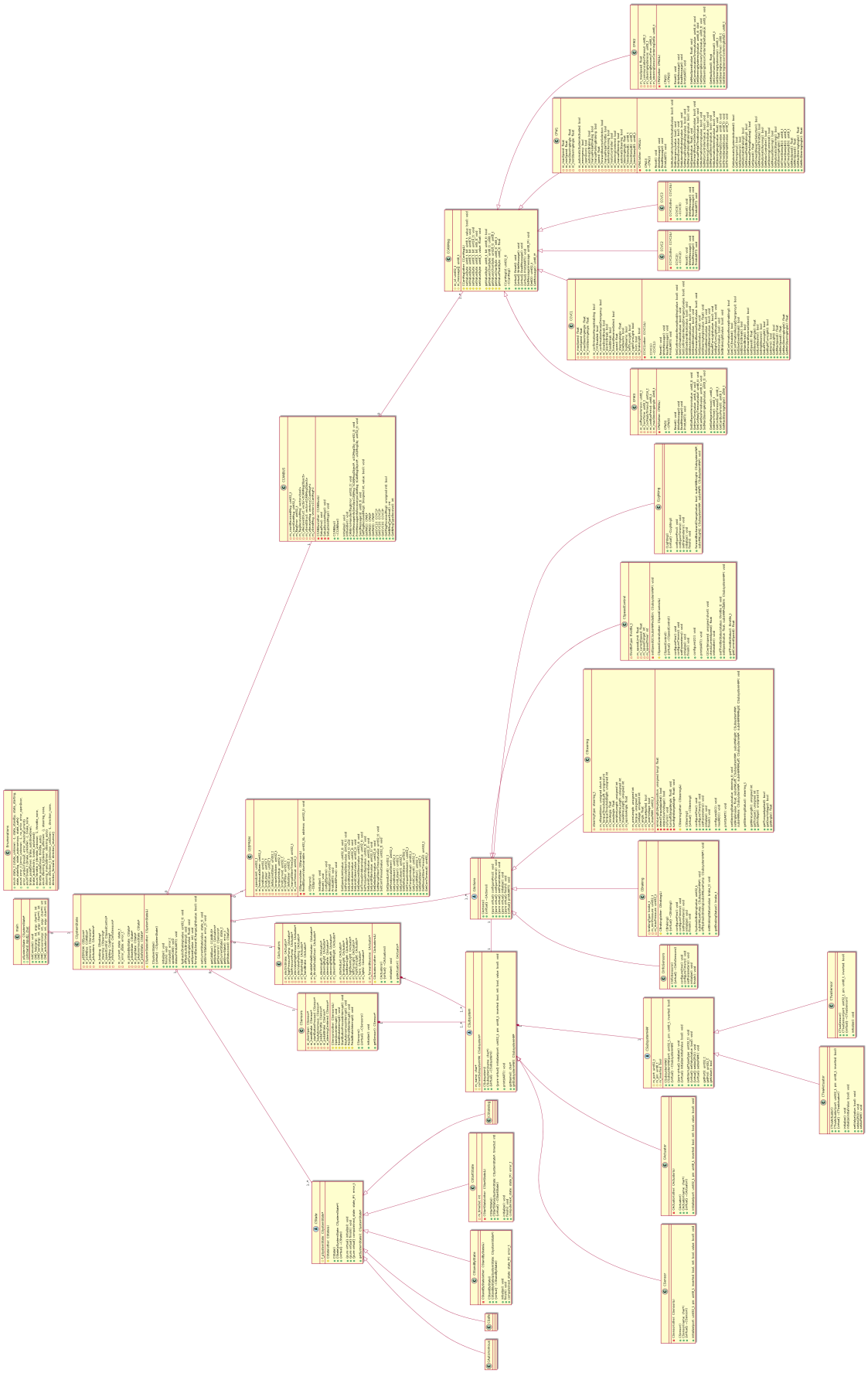
[16] ”Planificación de proyectos”(2017, septiembre), presentación de la asignatura común ”Gestión de proyectos”, Escuela de Ingeniería, Universidad Autónoma de Barcelona, 2016.

APÉNDICE

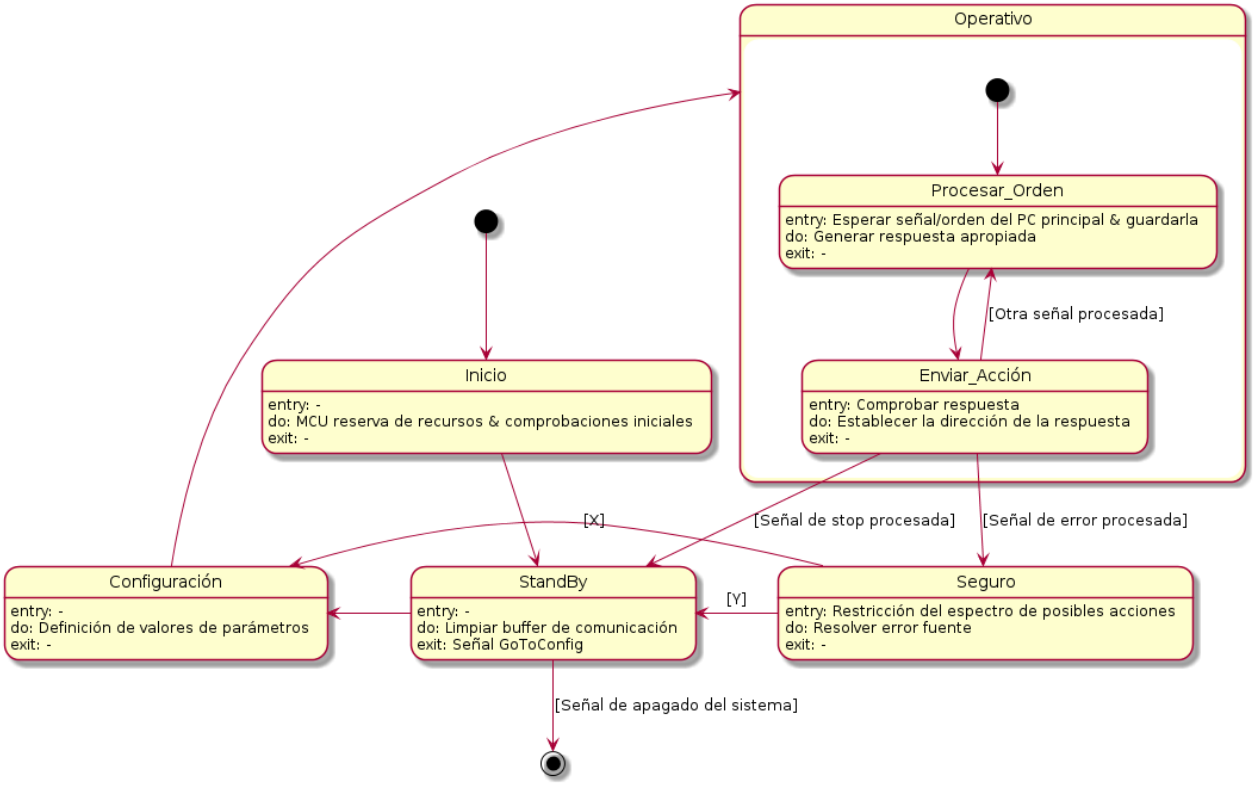
A.1. Diagrama de Gantt inicial



A.2. Versión final del diagrama de clases del ECUSW



A.3. Diagrama de estados del ECUSW



A.4. Microcontrolador TIVA