ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Carrera

DISEÑO DE COMPONENTES SOFTWARE DE TIEMPO REAL

Para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Pedro Javier Espeso Martínez

Diciembre - 2002

I. TECNOLOGÍA DE PROGRAMACIÓN BASADA EN COMPONENTES 1
I.1. Componentes software 1
I.2. Entornos normalizados de desarrollo de componentes software 4
I.2.1. EJB (Enterprise Java Bean) 4
I.2.2. COM+ (Component Object Model) 5
I.2.3. CORBA (Common Object Request Broker Architecture) 7
I.3. Metodología de diseño de aplicaciones basadas en componentes 9
I.4. Componentes software de tiempo real 11
I.5. Entorno MAST de modelado , diseño y análisis de sistemas de tiempo real 13
I.6. Perfil CBSE_MAST 15
I.7. Objetivos del proyecto 22

II. ESPECIFICACIÓN Y DISEÑO FUNCIONAL DE COMPONENTES 26
II.1. Componentes e interfaces 26
II.2. Especificación de una interfaz 28
II.3. Especificación de componentes 35
II.4. Proceso de automatización de la gestión de componentes 42

III. MODELO DE TIEMPO REAL DE LOS COMPONENTES 44
III.1. Modelo de tiempo real de una aplicación basada en componentes 44
III.2. Descripción de la plataforma WINDOWS NT 45
III.3. Modelo de tiempo real de la plataforma WINDOWS NT 48
III.4. Estimación de los valores de los parámetros del modelo de la plataforma 53
III.4.1. FP_Processor 53
III.4.2. Interference's y Ticker 55
III.5. Código MAST-File del componente NT_Processor_mdl_1 60
III.6. Modelo de tiempo real del componente C_IG_DT3153 64
III.7. Código Mast-File del componente C_IG_DT3153 72

IV. DEMOSTRADOR DE UN SISTEMA DE TIEMPO REAL BASADO EN COMPONENTES 76
IV.1. Control por computador de una maqueta de trenes utilizando visión artificial 76
IV.2. Funcionalidad de la aplicación 80
IV.3. Arquitectura software del demostrador 80
IV.4. Modelo de los componentes lógicos de la aplicación 86
IV.4.1. Modelo MAST de la clase Railway 86
IV.4.2. Modelo MAST de la clase Train 92
IV.5. Modelo de las situaciones de tiempo real de la aplicación 96
IV.5.1. Configuración de la RT_Situation 97
IV.5.2. Declaración de las transacciones 100
IV.6. Análisis de tiempo real de la aplicación 102

V. CONCLUSIONES 104

VI. REFERENCIAS 105

A. ESPECIFICACIÓN DE LOS COMPONENTES 107
A.1. Componente C_IG_DT3153 107
A.2. Componente C_Ada_Vision 108
A.3. Componente C_PCI9111 113

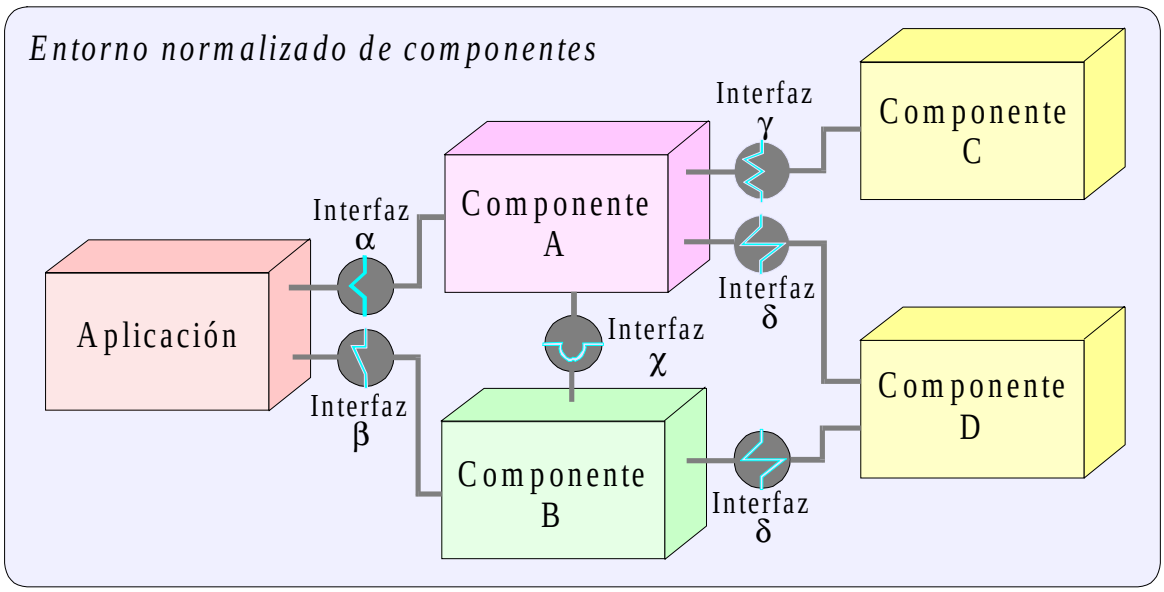
B. DISEÑO Y CABLEADO DE LA TARJETA INTERMEDIA DE RELÉS 116

I. TECNOLOGÍA DE PROGRAMACIÓN BASADA EN COMPONENTES

I.1. Componentes software

El objetivo de la tecnología de componentes software es construir aplicaciones complejas mediante ensamblado de módulos (componentes) que han sido previamente diseñados por otras personas a fin de ser reusados en múltiples aplicaciones [1]. La ingeniería de programación que sigue esta estrategia de diseño se la conoce por el acrónimo CBSE[1] y es actualmente una de las más prometedoras para incrementar la calidad del software, abreviar los tiempos de acceso al mercado y gestionar el contínuo incremento de su complejidad.

La arquitectura software de una aplicación basada en componentes consiste en uno o un número pequeño de componentes específicos de la aplicación (que se diseñan específicamente para ella), que hacen uso de otros muchos componentes prefabricados que se ensamblan entre sí para proporcionar los servicios que se necesitan en la aplicación [2].



En la tecnología de componentes la interfaz constituye el elemento básico de interconectividad. Cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación. Y debe operar correctamente con independencia de los mecanismos internos que utilice para soportar la funcionalidad de la interfaz.

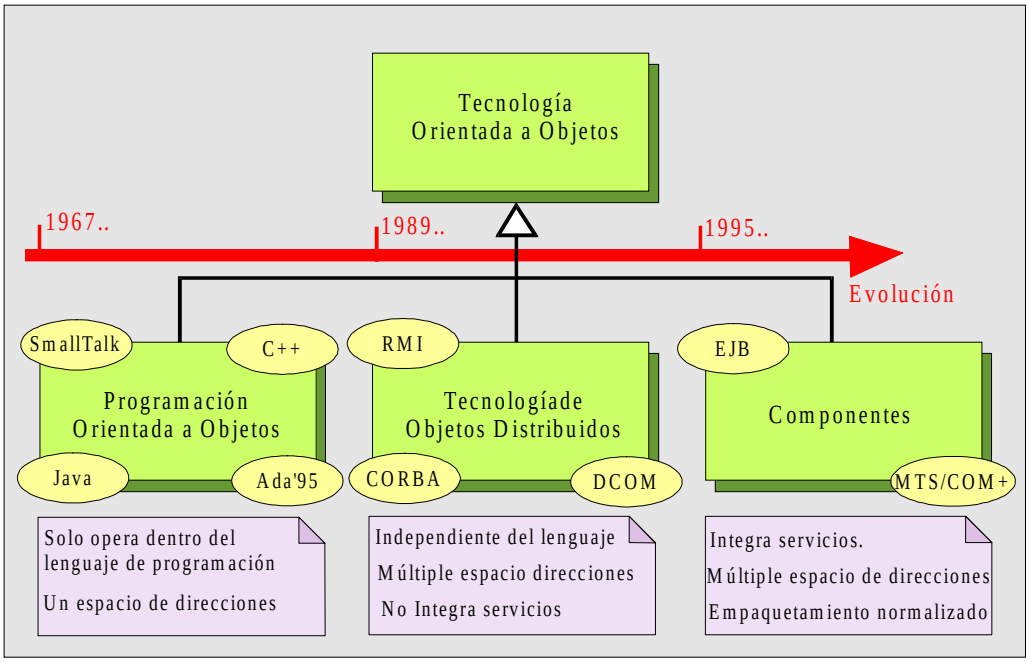
Características muy relevantes de la tecnología de programación basada en componentes son la modularidad, la reusabilidad y componibilidad y en todos ellos coincide con la tecnología orientada a objetos [3] de la que se puede considerar una evolución. Sin embargo, en la tecnología basada en componentes también se requiere robustez ya que los componentes han de

1. Component Based Software Engineering.

operar en entornos mucho mas heterogéneos y diversos. Principios básicos compartidos por las tecnologías CBSE y OO[1] son:

- *Integración de datos y funciones*: un objeto software consiste en una serie de valores (estado) y las funciones que procesan esos datos.
- *Encapsulamiento:* el cliente[2] de un objeto software no tiene conocimiento de como son almacenados los valores en el interior del objeto , ni como se implementan las funciones.
- *Identidad:* cada objeto software tiene una identidad única.
- *Polimorfismo:* las interfaces se describen por separado de la implementación , de modo que un código que requiera una determinada interfaz puede utilizar cualquier componente / objeto que implemente dicha interfaz. Esto permite una gran flexibilidad en el diseño de aplicaciones.

De hecho, el desarrollo de software basado componentes (CBSE) es la evolución natural de la ingeniería software para mejorar la calidad, disminuir los tiempos de desarrollo y gestionar la creciente complejidad de los sistemas.



Los componentes presentan una serie de ventajas frente a las tecnologías orientadas a objetos:

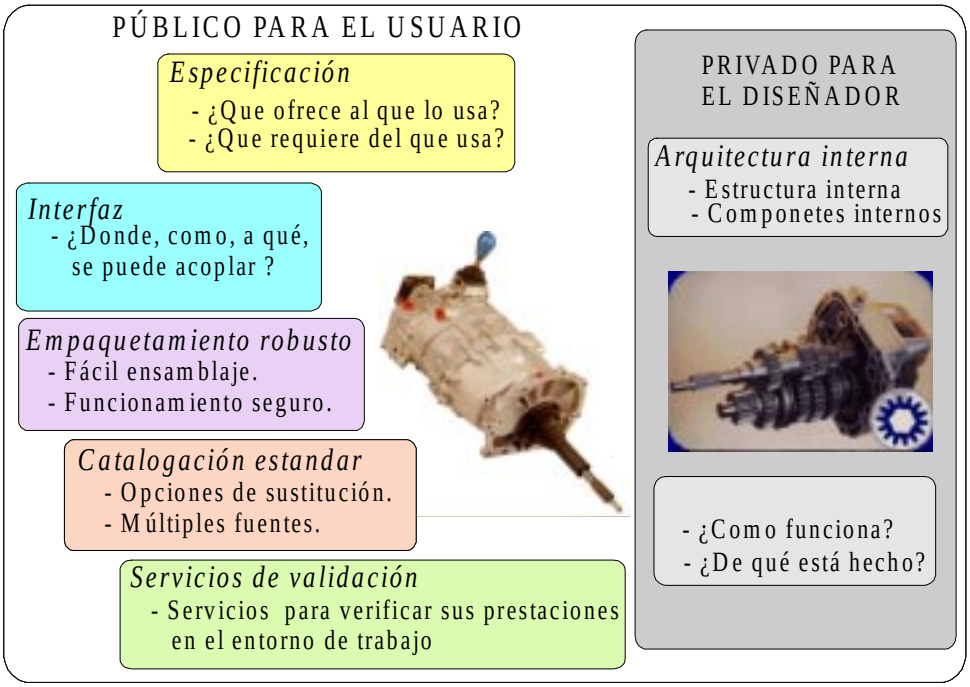
- Componentes desarrollados con diferentes lenguajes de programación son compatibles y pueden ejecutarse dentro de una misma aplicación.
- Los componentes puede tener su propio estado persistente (en una base de datos , en el sistema de ficheros del disco , ...).

1. Object Oriented.
2. En el sentido de usuario o consumidor.

- Las funciones de la interfaz no tiene que estar necesariamente implementadas utilizando técnicas orientadas a objeto.
- Los componentes requieren mecanismos de empaquetamiento que deben ser más robustos que los de los objetos.
- Normalmente (por ejemplo en COM+ y EJB[1]) las instancias de componentes se instalan en un *contenedor* , que les proporciona contexto local. Mientras el componente aporta la simplemente la funcionalidad (*business logic*), es el contenedor quien se encarga de mapear las estructuras de datos desde el espacio lógico hasta el espacio físico de memoria en la máquina sobre la que se está ejecutando.
- Los componentes integran servicios completos , lo que minimiza las interconexiones entre módulos.

La funcionalidad de un componente software se ofrece a través de un conjunto de interfaces. Cada interfaz es una unidad funcional independiente que representa dentro de un dominio de aplicación un conjunto de servicios en los que se puede basar una aplicación sin tener que hacer referencia al componente que los soporta. Por ello, la información completa sobre los servicios que ofrece una interfaz debe formar parte de ella y describir tanto la funcionalidad que ofrece en estados correctos como la que ofrece bajo situación de fallo.

La tecnología de componentes es habitual en otras ingenierías, y su éxito se basa en:



- Una <u>especificación completa</u> hacia el usuario. Tanto de los servicios que ofrece como de los servicios externos que requiere para operar correctamente.

- <u>Múltiples interfaces</u> en los que se especifican los diferentes modos de acoplo que admite, así como los mecanismos de acoplo y sus compatibilidades.

1. Entornos normalizados descritos posteriormente en este mismo texto.

- Un empaquetamiento robusto, de fácil manipulación y que garantice un funcionamiento fiable e independiente de otros componente que operen junto a él.

- Una catalogación estándar que simplifique su localización, permita su sustitución por otros equivalentes y que facilite el desarrollo de múltiples fuentes que garantice el soporte del producto.

- Debe ofrecer mecanismos de validación en el entorno de trabajo del usuario y capacidad de configuración y sintonización de acuerdo con las necesidades de la aplicación.

La arquitectura interna, las peculiaridades de su implementación y la tecnología que lo soporta, sólo debe interesar al diseñador del componente y a los que sean responsables de su mantenimiento y evolución.

Como reflexión final, podemos concluir que la ventaja principal del diseño basado en componentes no es la reutilización del software (que también), sino la posibilidad de sustituir un componente por otro que implemente las interfaces utilizadas en el sistema. De este modo mejoramos la calidad del sistema completo sin necesidad de rediseñarlo.

I.2. Entornos normalizados de desarrollo de componentes software

Para que una arquitectura de componentes pueda operar es necesario disponer de un entorno normalizado que proporcione soporte a los mecanismos con que se comunican las interfaces.

Existen varios entornos de soporte de componente creados para uso interno de empresas, pero los más conocidos (actualmente comercializados) son:

I.2.1. EJB (Enterprise Java Bean)

Tecnología desarrollada por Sun Microsystems.

Un “Java Bean” es un componente utilizado en un entorno Java que permite agrupar funcionalidades que pueden ser utilizadas por múltiples aplicaciones.

Un “Enterprise Java Bean” también agrupa servicios funcionales utilizables por aplicaciones, sin embargo, a diferencia del anterior el EJB implica la existencia de un entorno de ejecución conocido como “EJB Container”. Así, mientras que un “Java Bean” requiere ser integrado con otros componentes para que sea funcional, un EJB puede ser activado y usado con solo ser incluido en un “EJB Container”.

Algunas ventajas de los EJB’s son:

- **División de trabajo**: el “EJB Container” es el encargado de ofrecer los servicios , de modo que el diseñador del componente se centra exclusivamente en el desarrollo de la funcionalidad principal (*business logic*). La interoperabilidad entre servicios y

componentes se define en las especificaciones correspondientes que forman parte del estándar J2EE[1].

- **Diversos vendedores**: existen diversos vendedores tanto de "EJB Containers" , los cuales son incluidos en un servidor de aplicaciones Java , como de EJB's que resuelven algún tipo de lógica concreta. La compatibilidad está garantizada, y se puede ejecutar cualquier EJB en cualquier "EJB Container", aunque ambos hayan sido desarrollados por distintas empresas proveedoras.

- **Procedimientos remotos**: el diseño de los EJB gira alrededor de la tecnología RMI[2], lo que permite la operación en sistemas distribuidos.

- **Diversos clientes**: un EJB puede interactuar con una gran gama de clientes: Servlets, bases de datos , applets , sistemas ERP[3], ...

Por contra , también presenta algunos inconvenientes:

- **Tiempo de desarrollo elevado**: desarrollar un sistema con EJB's es sumamente complejo ; si bien para muchas empresas puede presentar una solución ideal , para otras puede resultar un solución sobrada debido a su elevado costo (complejidad - tiempo).

- **Conocimiento exhaustivo de Java**: la tecnología EJB es uno de los principales componentes de J2EE y por esta razón depende fuertemente de otras de sus partes: RMI , JNDI[4] , JDBC[5] , ...

I.2.2. COM+ (Component Object Model)

Microsoft Component Object Model [4].

Los lenguajes de programación clásicos fueron diseñados para desarrollar aplicaciones secuenciales compuestas de módulos, todos ellos codificados con un solo lenguaje. Sin embargo, hay situaciones en las que no es práctico restringirse al uso de un único lenguaje. La tecnología COM aborda la solución a este problema proporcionando un sencillo, pero a la vez potente modelo para construir sistemas software a partir de la interacción de objetos (componentes).

COM define un estándar binario (esto implica que es independiente del lenguaje de programación) para objetos y la intercomunicación entre ellos. Toda comunicación se realiza a través de operaciones que son proporcionadas dentro de interfaces. El diseñador invoca las operaciones que necesite directamente, incluso si el objeto destinatario está localizado en otro

1. Java 2 Platform Enterprise Edition.
2. (Remote Method Invocation) Invocación Remota de Procedimientos.
3. (Enterprise Resource Planning) describe una serie de actividades de gestión empresarial soportadas por aplicaciones de tecnología de información.
4. (Java Naming and Directory Interface) conjunto de aplicaciones API que actúan como interfaz de múltiples servicios de directorio e identificación.
5. (Java DataBase Conectivity) estándar que permite conexiones independientes entre bases de datos basadas en API's SQL utilizando Java.

proceso o en otra máquina. De este modo, la combinación de la tecnología COM junto con las técnicas de programación orientada a objeto, nos ofrece una importante simplificación en el proceso de desarrollo de aplicaciones informáticas.

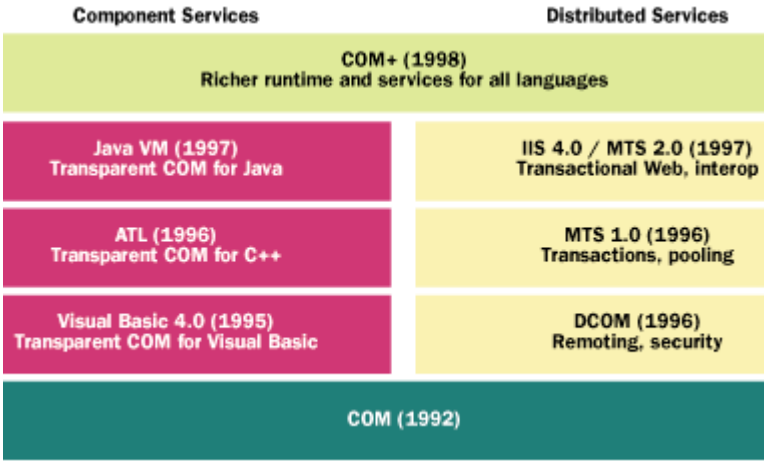
Distintas herramientas han introducido particularidades propias para facilitar el desarrollo de aplicaciones basadas en COM.

Por ejemplo, ATL[1] dispone de un mecanismo basado en scripts para simplificar el registro de componentes.

Visual Basic implementa un acceso sencillo a estructuras de datos a través de controles.

MTS[2] proporciona mecanismos que permiten la escritura de código escalable; esto quiere decir que el desarrollador del componente escribe código asumiendo que un solo cliente accede al componente al tiempo, y es MTS quién se encarga de automatizar el proceso cuando varios clientes acceden simultáneamente. Esta propiedad simplifica enormemente el desarrollo de sitemas distribuidos.

En 1998 aparece la tecnología COM+ como la evolución natural de las tecnologías de componentes y sistemas distribuidos.

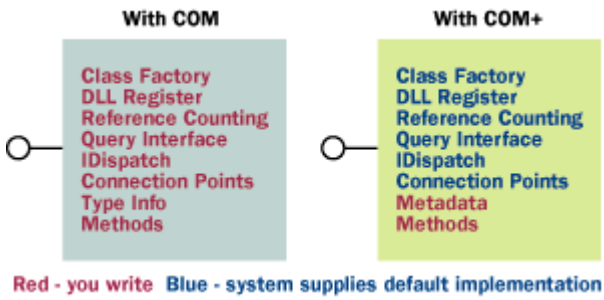


La tecnología COM+ presenta varias ventajas respecto a su antecesora COM.

En primer lugar, simplifica el desarrollo de los componentes ya que genera de forma automática la implementación de diversas interfaces (IUnknown, IDispatch, IConnectionPoint, ...) necesarias en COM para gestionar aspectos relativos al tiempo de vida del componente, contabilidad , referencias , ...

Uno de los inconvenientes de COM era la complejidad de definir nuevas interfaces utilizando IDL[3]. Con COM+, las interfaces se definen usando lenguajes de programación estándar: no se fuerza a aprender un nuevo lenguaje, API[4] o forma de escribir código.

1. (Active Template Library) Biblioteca de plantilas C++ para el uso de componentes COM.
2. Microsoft Transaction Server.
3. Interface Definition Languaje.
4. Application Programming Interface.



COM+ proporciona un modelo más simple y robusto para registrar e instalar componentes. El mecanismo de registro permite reescribir la versión de una clase asociada a un paquete. Esto resuelve una de las cuestiones más molestas de las aplicaciones basadas en componentes. Por ejemplo, supongamos que la aplicación A utiliza la versión 1.0 del componente C y funciona correctamente. La aplicación B instala la versión 2.0 del componente C y también funciona correctamente, pero la aplicación A deja de hacerlo con la nueva versión de C. Con COM+, ambas versiones del componente se pueden usar simultáneamente, la 1.0 para A y la 2.0 para B.

COM+ presenta muchas otras ventajas respecto a COM en aspectos relativos a la gestión interna del componente, liberación de memoria (*garbage collection*), accesos seguros, sistemas distribuidos , ...

Además, COM+ es compatible con COM, de modo que los antiguos componentes COM pueden ser usados junto con componentes COM+.

I.2.3. CORBA (Common Object Request Broker Architecture)

Common Object Request Broker Architecture de OMG[1][5].

CORBA es la solución propuesta por el OMG para cubrir las necesidades de interoperabilidad entre los diversos productos hardware y software que nos ofrece el mercado hoy en día.

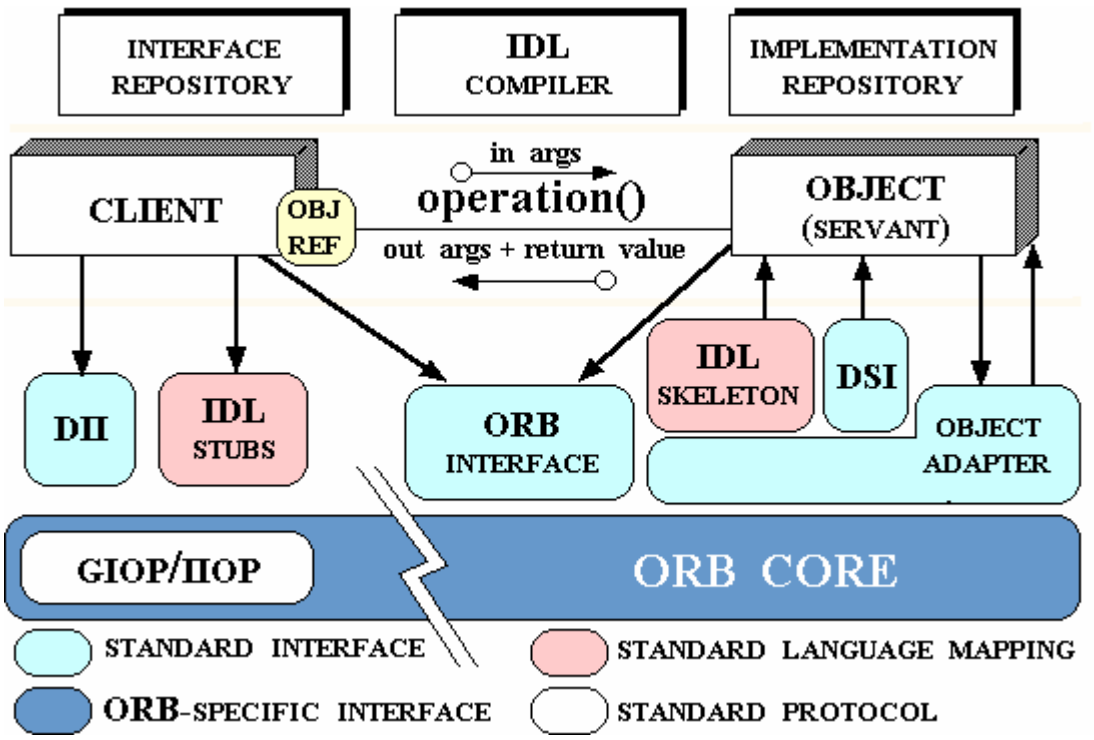
CORBA 1.1 aparece en 1991 y en él se definen el IDL y la API que permiten la interacción de objetos en modo cliente/servidor utilizando una implementación específica de un ORB[2]. En 1994 aparece CORBA 2.0, donde se define la interoperabilidad real entre ORB's de distintos fabricantes.

En la figura de la siguiente página se muestra la arquitectura CORBA - ORB.

La descripción de los distintos elementos que la conforman es la siguiente:

- **Object:** entidad de programación de CORBA compuesta de una identidad, una interface y una implementación (también conocida como *Servant*).

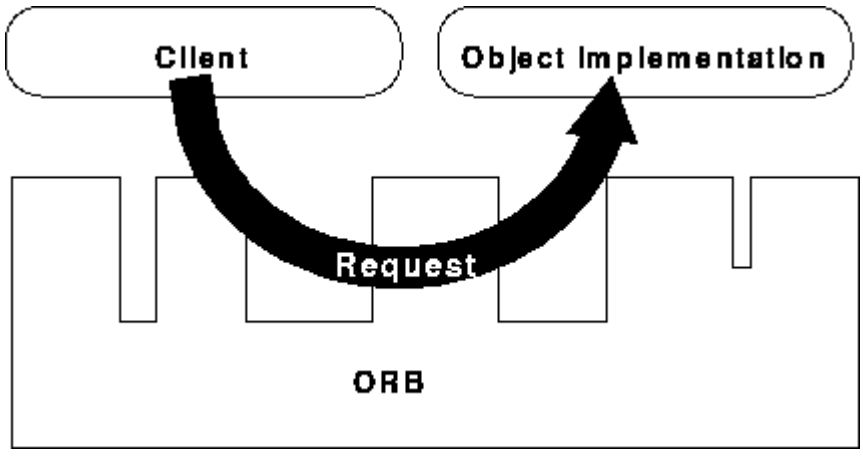
1. Object Management Group
2. Object Request Broker.



- **Servant:** implementación de un *Object* donde se definen las operaciones ofrecidas por una interfaz CORBA - IDL. Puede estar escrita en diversos lenguajes (C, C++, Java, Smalltalk, Ada, ...).

- **Client:** entidad de programación que invoca una operación sobre un *Object*. El acceso a los servicios del *Object* remoto es transparente para el usurio *Client*.

- **Object Request Broker (ORB):** mecanismo que proporciona una comunicación transparente entre el *Client* y el *Object*. Esto hace que las solicitudes de Client sean aparentemente simples procedimientos locales desde su punto de vista. Cuando el *Client* invoca una operación, el *ORB* es el responsable de localizar la implementación del *Object*, trasladarle la petición y devolver los resultados al *Client*. Internamente, un *ORB* puede utilizar diferentes protocolos de comunicación, como por ejemplo *IIOP*[1] o *GIOP*[2] [6].



- **IDL Stubs / Skeletons** : son el punto de conexión entre el *Client* / *Object* (respectivamente) y el *ORB*. La transformación entre la definición del CORBA IDL y el lenguaje de programación utilizado es automática a través del compilador IDL. El uso del compilador reduce el peligro de aparición de inconsistencias entre los *Stubs* del cliente y los *Skeletons* del servidor, e incrementa las posibilidades de optimizar la generación automática del código.

- **Dynamic Invocation Interface (DII):** interface que permite al cliente acceder directamente a los mecanismos subyacentes de peticiones que ofrece un *ORB*. Las aplicaciones utilizan el *DII* para realizar peticiones dinámicamente sin necesidad de utilizar interfaces IDL (*Stubs*). A diferencia de los Stubs, el DII permite al Client efectuar llamadas no bloqueantes[1] o unidireccionales[2].

- **Dynamic Skeleton Interface (DSI):** es el análogo al *DII*. El *DSI* permite a un *ORB* trasladar peticiones a un *Object* que no tiene conocimiento en tiempo de compilación del tipo de *Object* que va a implementar. El *Client* que realiza la petición no tiene porque saber si el *ORB* utiliza IDL *Skeletons* o *DSI*.

- **Object Adapter:** es el encargado de asociar los distintos *Servants* con el *ORB*. Pueden ser especializados para ciertos estilos de *Servants*, como por ejemplo OODB[3] Object Adapters para objetos persistentes o Library Object Adapters para objetos locales (no remotos).

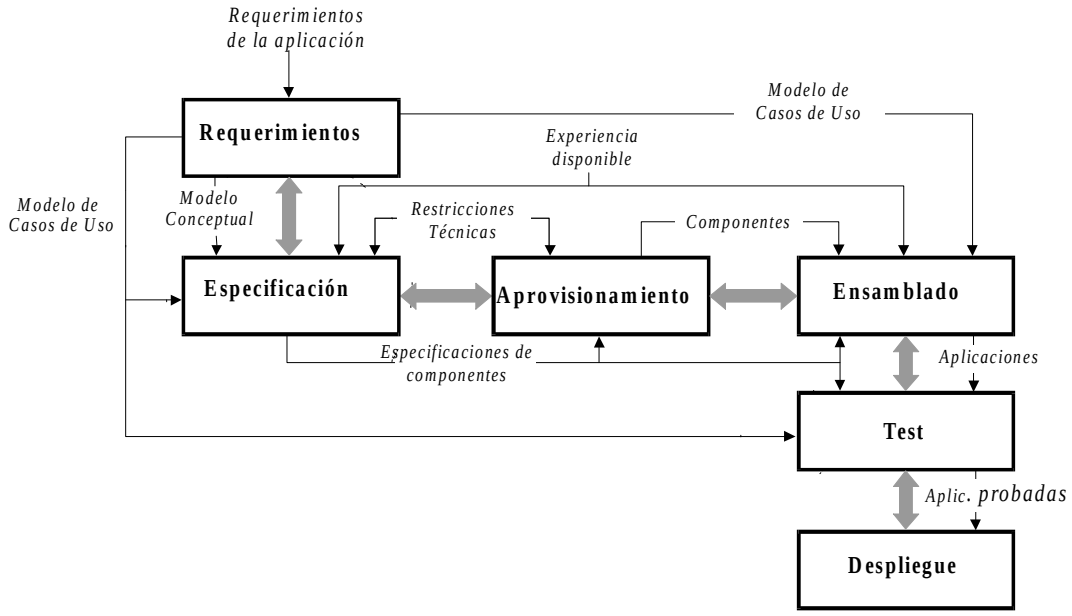
I.3. Metodología de diseño de aplicaciones basadas en componentes

El aspecto central del proceso de diseño de aplicaciones basadas en componentes es la gestión de la funcionalidad a través de las especificaciones de las interfaces. Tras una evaluación de algunos de los procesos de diseño basado en componentes [3][8][9][10], se ha adoptado el proceso RUP[4] propuesto por por Cheesman [8]. En la figura se muestra un esquema a alto nivel de este proceso. Los bloques representan conjuntos de actividades que dan lugar a resultados tangibles, las flechas gruesas representan su secuenciación y las flechas finas representan el flujo de elementos generados que transfieren información entre ellas. Si comparamos este diagrama con los propuestos en la metodología RUP para la metodología orientada a objetos, se comprueba que las actividades iniciales y finales de definición de requerimientos, prueba y despliegue coinciden, mientras que difiere en que las fases centrales del proceso RUP (análisis, diseño e implementación de objetos) se han reemplazado por otras que representan la especificación, aprovisionamiento y ensamblado de componentes.

Los conjuntos de actividades que intervienen en el proceso son:

- **Requerimientos**: obtener una idea clara y concreta de los requerimientos que exige la aplicación. Se genera un modelo conceptual general y un modelo de casos de uso.

1. Internet Inter-ORB Protocol.
2. General Inter-ORB Protocol.
1. Operaciones de envío y recepción distintas.
2. Sólo envío.
3. Object Oriented DataBases.
4. Rational Unified Process.



- **Especificación**: consta de tres etapas. Primero, la *identificación de componentes* donde se genera un conjunto inicial de interfaces y especificaciones de componentes conectados entre sí en una primera aproximación de la arquitectura. Segundo, la *interacción entre componentes* donde se decide como los componentes van a trabajar de modo coordinado para conseguir la funcionalidad deseada. Tercero, la *especificación de componentes* donde se definen las especificaciones concretas de cada uno de los componentes utilizados. Al final se han generado las especificaciones de los componentes.

- **Aprovisionamiento**: a partir de las especificaciones se implementa la funcionalidad de los componentes. Se generan los componentes completos.

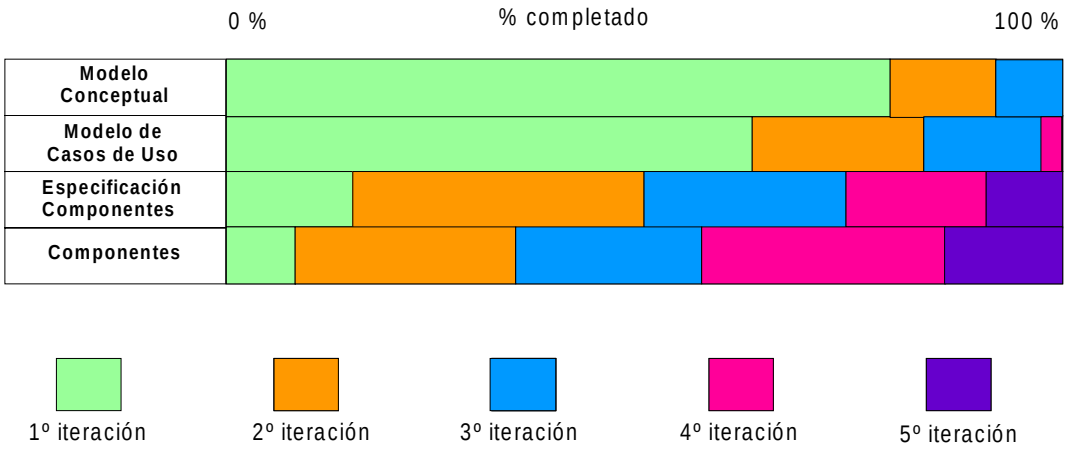
- **Ensamblado**: se conectan los componentes unos con otros a través de sus interfaces para lograr la funcionalidad requerida. Se generan las aplicaciones.

- **Test**: se comprueba el correcto funcionamiento de las aplicaciones. Se generan las aplicaciones probadas.

- **Despliegue**: se instalan las aplicaciones probadas en el entorno de trabajo donde van a llevar a cabo su funcionamiento.

La secuenciación de las actividades está representada por flechas bidireccionales. Esto quiere decir que el proceso de desarrollo es iterativo, y en un momento dado se puede volver a alguna etapa anterior para realizar modificaciones. De hecho, lo habitual es necesitar varias iteraciones para llegar a las versiones finales de los resultados generados.

Los dos criterios básicos por lo que este proceso se ha tomado como base de nuestro trabajo son su carácter neutro respecto del proceso de diseño lo que facilita las extensiones y su formulación basada en UML (Unified Modeling Languaje) que lo hace compatible con las



herramientas CASE (Computer Aided Software Engeneering) que ya utilizábamos (ROSE de Rational).

I.4. Componentes software de tiempo real

Entendemos por sistemas de tiempo real aquellos en los que el correcto funcionamiento del sistema requiere el cumplimiento de plazos temporales [11]. Habitualmente los sistemas de tiempo real se utilizan para controlar o interactuar con un sistema físico, y los requerimientos temporales vienen impuestos por el entorno. Como consecuencia de esto, el correcto funcionamiento de estos sistemas depende no sólo de los resultados conseguidos a través de la computación de los datos, sino también de los instantes temporales en que se obtienen.

Aunque la tecnología de componentes es ya una realidad en muchos campos, como multimedia, interfaces gráficas, etc., plantea problemas para su aplicación en otros dominios como es el caso de los sistemas de tiempo real.

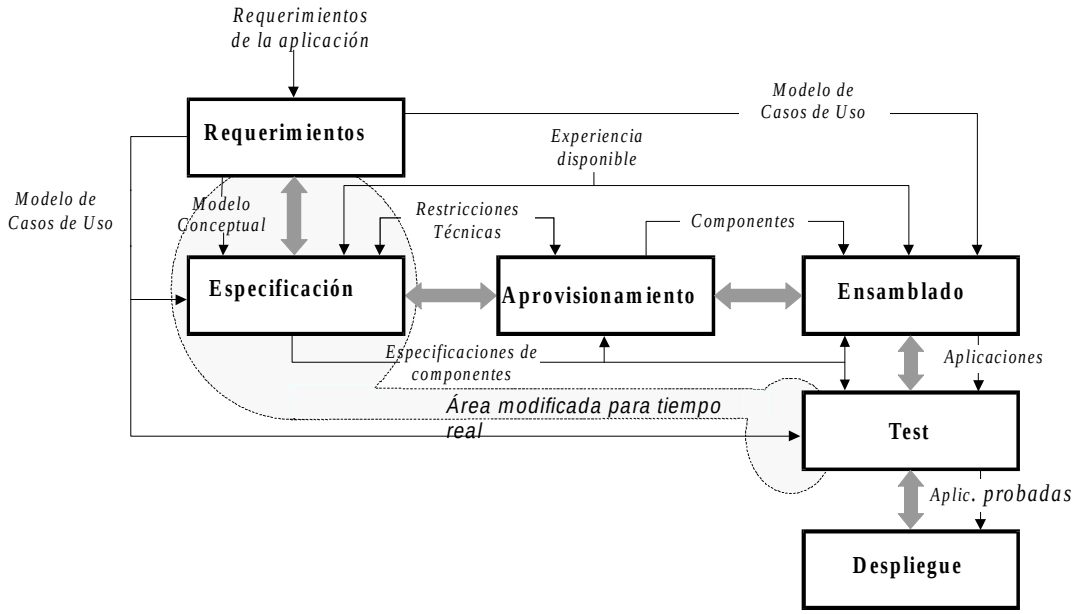
Sin embargo, la necesidad de la tecnología de componentes en sistemas de tiempo real está siendo manifestada por la industria de automatización, potencia y equipamiento eléctrico, aviación, automóvil, etc... Empresas líderes en estos campos como ABB han implantado la tecnología de componentes en sus productos, basándola en soluciones y especificaciones propias [12], lo cual le ha reportado ventajas en cuanto al mantenimiento y a la gestión de la evolución de sus productos, pero ha supuesto muchos problemas a sus clientes que encuentran dificultades de compatibilidad de estos productos con aplicaciones desarrolladas por terceros.

Cuando aplicamos CBSE en el desarrollo de sistemas de tiempo real uno de los problemas fundamentales es la reusabilidad de los componentes. El diseño de los componentes es mucho más complejo ya que, en aplicaciones de tiempo real , han de colaborar unos con otros para cumplir las restricciones temporales. El uso de un determinado sistema operativo o una base de datos concreta no es siempre posible en sistemas de tiempo real, ya que muchos de estos elementos han sido diseñados para optimizar su rendimiento, pero no para garantizar su predicibilidad temporal. Componentes comerciales como EJB ,CORBA o COM no pueden ser utilizados en sistemas de tiempo real precisamente por este motivo. En líneas generales podemos afirmar que el incremento de flexibilidad conlleva un decremento en la predicibilidad

temporal, de modo que es impensable (actualmente) obtener la flexibilidad que nos ofrecen los entornos comerciales mencionados en un modelo para sistemas de tiempo real.

La propuesta que hacemos es modificar el proceso RUP[1] para el diseño de componentes de tiempo real agregando a las especificaciones de los requerimientos de las aplicaciones y a las descripción de los componentes nuevas secciones que especifican y describen el comportamiento de tiempo real requerido y ofertado. Así, disponiendo de la descripción de cada una de los componentes que constituyen una aplicación, junto con un modelo complementario que describe las capacidades de procesamiento de las plataformas hardware/software en que se ejecutan, se puede construir un modelo de tiempo real de la aplicación completa, que es utilizado como base de operación de las herramientas de diseño que ayudan a la configuración óptima de los componentes y de las herramientas de análisis que permiten garantizar la planificabilidad de la aplicación, o en caso negativo mediante análisis de holguras localizar las causas que le impiden satisfacer los requerimientos temporales.

La extensión que se propone solo afecta a tres de los bloques de trabajo del proceso de desarrollo:



- En el bloque de definición de los requerimientos, se ha realizado la extensión necesaria para que se puedan formular los requerimientos de tiempo real de la aplicación. Estos requerimientos se definen de forma independiente para cada modos de operación del sistema, y por cada uno de ellos, se formulan a través de las declaraciones de los conjuntos de transacciones que concurren en él. La declaración de cada transacción de tiempo real incluye la descripción de sus características de disparo, los conjuntos de actividades que conlleva su ejecución y los requerimientos de tiempo real que se imponen a lo largo de ella.

1. Descrito en el apartado I.3.

- En el bloque de actividades relativas a la especificación de los componentes la extensión incluye los procedimientos de descripción del comportamiento temporal del componente. Esto se realiza mediante la especificación de la secuencia de actividades que conlleva la ejecución de cada una de los procedimientos ofrecidos por sus interfaces, y por cada actividad, se describe la cantidad procesado que requiere así como los recursos compartidos que puede requerir y que pueden dar lugar a suspensiones de su ejecución.

- Por último, en el bloque de actividades de test se incluyen las actividades que generan el modelo de tiempo real de la aplicación a partir de los modelos de los componentes que se han utilizado para su ensamblaje y de los modelos de las plataformas hardware/software en que se despliega la aplicación. Así mismo se incluyen, los procesos de análisis del modelo construido, bien para la toma de decisión sobre la configuración óptima de los componentes o bien para verificar su planificabilidad.

La idea básica de la extensión que se propone es que el diseño de sistemas de tiempo real basados en componentes no se fundamenta en una estrategia de diseño de componentes con unas prestaciones temporales preestablecidas e independientes de su uso (lo cual no se sabe hacer salvo para situaciones muy laxas), sino en construir componentes que llevan asociado un modelo de comportamiento temporal, que permite modelar y analizar el comportamiento de tiempo real de las aplicaciones que hagan uso de ellos. El método que se propone, se basa en la potencia de modelado y de análisis de que se dispone, y no en la propuesta de nuevas en estrategias de diseño.

El desarrollo de componentes estándar de tiempo real que puedan funcionar sobre distintas plataformas hardware es complicado debido a que los componentes tienen diferentes características temporales sobre diferentes plataformas. Cada componente ha de ser adaptado y verificado de nuevo para cada plataforma hardware sobre la que se pretende instalar , especialmente si el sistema es crítico (tiempo real estricto). Lo que se propone es generar por separado modelos independientes de las plataformas y de los componentes que nos permitan analizar el funcionamiento del sistema completo.

I.5. Entorno MAST de modelado , diseño y análisis de sistemas de tiempo real

El entorno MAST[1] está siendo desarrollado en la actualidad por el grupo CTR[2] de la Universidad de Cantabria , y su metodología ya ha sido aplicada en diferentes entornos [13][14][15]. Su principal objetivo es simplificar la aplicación de técnicas estándar y bien conocidas de análisis de sistemas de tiempo real, proporcionando al diseñador un conjunto de herramientas para aplicar técnicas de análisis de planificabilidad, de asignación óptima de prioridades o de cálculos de holguras, etc., sin necesidad de que tenga que conocer los detalles algorítmicos de los métodos. Actualmente, MAST cubre sistemas monoprocesadores, sistemas multiprocesadores, y sistemas distribuidos basados sobre diferentes estrategias de planificación, incluyendo planificación expulsoras y no expulsoras, manejadores de interrupciones, servidores esporádicos, escrutinios periódicos, etc.

1. Modeling and Analysis Suite for real-Time aplications.
2. Computadores y Tiempo Real.

Características relevantes de la metodología MAST que la hace especialmente idónea para modelar sistemas basados en componentes , son:

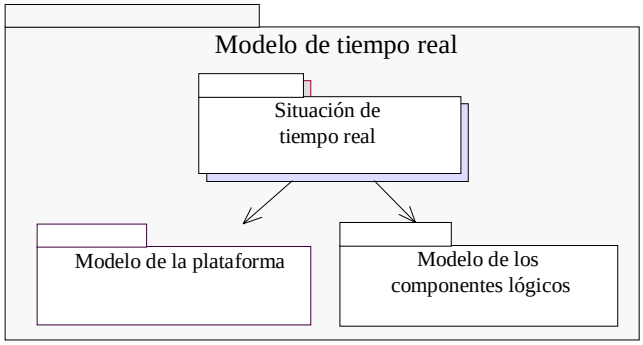
- Se basa en el modelado independiente de la plataforma (procesadores, redes de comunicación, sistema operativo, drivers, etc.), de los componentes lógicos que se utilizan (requerimientos de procesado, de recursos compartidos, de otros componentes, etc.) y del propio sistema que se construye (partición, despliegue, situaciones de tiempo real, carga de trabajo (workload) y requerimientos temporales).

- El modelo se construye con elementos que modelan los aspectos de tiempo real (temporización, concurrencia, sincronización, etc.) de los elementos lógicos (componentes), y en consecuencia da lugar a un modelo que tiene la misma estructura que el código.

- Permite el modelado independiente de cada componente lógico de alto nivel, a través de un modelo de tiempo real genérico, que se instancia en un modelo analizable cuando se completa con el modelo de los componentes de los que depende y se asignan valores a los parámetros definidos en el modelo.

- Soporta implícitamente la distribución de componentes, de forma que en función de que un componente se encuentre asignado o no al mismo procesador que otro del que requiere servicios, se incorpora o no el modelo de la comunicación a la invocación del servicio.

El comportamiento de tiempo real de un sistema se descompone en tres secciones independientes pero complementarias que en conjunto describen un modelo que proporciona toda la información estructural y cuantitativa que se necesita para ser procesado por las herramientas de diseño y análisis de que se dispone.



Modelo de la Plataforma: Modela los recursos hardware/software que constituyen la plataforma en que ejecuta la aplicación y que es independiente de ella. Modela los procesadores (la capacidad de procesado, el planificador, el timer del sistema, ...), las redes de comunicación: (la capacidad de transferencia, el modo de transmisión, los planificadores de mensajes, los drivers de comunicación, ...) y la configuración (las conexiones entre los procesadores a través de las redes de comunicación)

Modelo de los componentes lógicos: Modela el comportamiento de tiempo real de los componentes software con los que se construye la aplicación. El modelo de un componente software describe:

- La cantidad de procesado que requiere la ejecución de las operaciones.
- Los componentes lógicos de los que requiere servicios.
- Las tareas o threads que crea para implementar sus operaciones.
- Los mecanismos de sincronización que requiere sus operaciones.
- Los parámetros de planificación definidos en el código de sus operaciones.
- Los estados internos relevantes a efecto de requerimientos de tiempo real.

Modelo de las situaciones de tiempo real: Una "Situación de Tiempo Real" representa un modo de operación del sistema junto con un modelo de la carga de trabajo que tiene que ejecutar y constituye el ámbito en que operara una herramienta de análisis. El análisis de un sistema de tiempo real consiste en la identificación de las situaciones de tiempo real que el sistema puede alcanzar y en el análisis independiente de cada una de ellas. Aunque los modelos de las situaciones de tiempo real son planteados independientemente comparten el modelo de la plataforma y el modelo de muchos de los componentes software que se utilizan en ellas.

El entorno MAST ofrece un conjunto de componentes primitivos que se utilizan en el modelado de un sistema [13]. Estos componentes son simples y muy próximos a los componentes hardware y software que intervienen en la respuesta temporal de la aplicación, y su uso directo en aplicaciones complejas en las que se necesitan cientos o miles de ellos para construir el modelo puede ser difícil de validar. Para evitar este problema se han definido diferentes perfiles de modelado que proporcionan componentes de mas alto nivel que facilitan la formulación del modelo de tiempo real dentro de ciertos entornos de programación específicos. Actualmente se disponen de los siguientes perfiles:

_ **UML_MAST:** Perfil para el diseño de modelo de aplicaciones de tiempo real basadas en metodologías orientadas a objetos y desarrolladas mediante herramientas UML[14].

_ **ADA_MAST:** Perfil para el diseño de modelos de aplicaciones de tiempo real y distribuidas desarrolladas utilizando ADA´95 con las extensiones establecidas en su apéndices D (Real-Time System) y E (Distributed Systems) [15].

_ **CBSE_MAST:** Perfil para el diseño de modelos de tiempo real de componentes software que facilitan el diseño del modelo de aplicaciones basadas en ellos [17].

El perfil que ha sido utilizado en este proyecto es este último y será descrito con mayor detalle en la próxima sección.

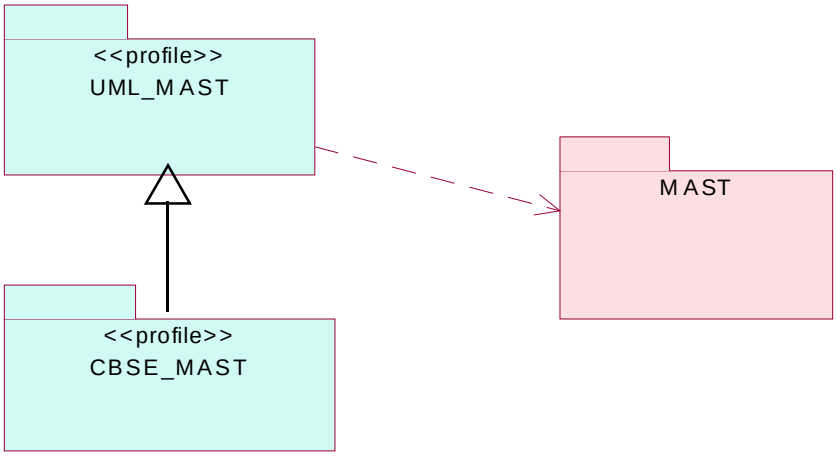
I.6. Perfil CBSE_MAST

CBSE_MAST es un perfil definido en el entorno MAST a fin de simplificar la formulación del modelo de tiempo real de un sistema y de una aplicación construida por agregación de componentes software tal como se considera en la metodología CBSE (Component Based Software Engineering).

Con el perfil CBSE_MAST, los componentes (hardware y/o software) que se tienen catalogados tienen asociado su modelo de tiempo real y cuando construimos el sistema componiendo algunos de ellos, el modelo de tiempo real del sistema completo se podrá a su vez construir componiendo los modelos de tiempo real de los componentes utilizados.

El perfil CBSE_MAST ha sido formulado para ser aplicado a sistemas descritos mediante UML y proporciona recursos y normas para construir una nueva vista complementaria del modelo UML del sistema, que denominamos MAST_RT_View, que constituye un modelo de su comportamiento de tiempo real.

CBSE_MAST es un perfil especializado del perfil UML_MAST y heredada de él todos los componentes de modelado de mediano y bajo nivel. De hecho CBSE_MAST es una extensión del perfil MAST_UML en el que se incluyen los elementos contenedores y de modularización que se necesitan para modelar eficientemente los diseños de sistemas desarrollados usando la metodología CBSE.

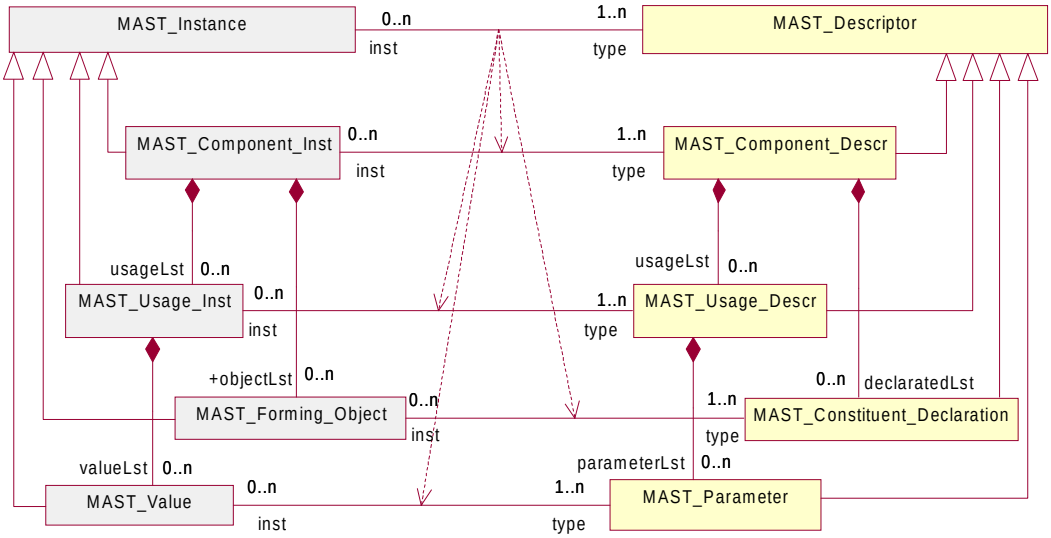


CBSE_MAST queda definido por un metamodelo CBSE_MAST_Metamodel formulado en UML [17]. Este metamodelo es a su vez una extensión del metamodelo UML_MAST_Metamodel y solo describe los nuevos aspectos establecidos en él respecto de este.

El aspecto central de la metodología CBSE es la modularización de subsistemas para su reusabilidad y esto requiere que el método de modelado disponga de una mayor precisión de los conceptos de descriptor de un componente MAST como clase que define un patrón de modelado de un subsistema parametrizable disponible en la biblioteca de componentes y el de instancia de componente MAST, como objeto que representa el modelo de tiempo real de un componente concreto que participa en la ejecución de la aplicación que se analiza y que influye en el comportamiento temporal.

Un **MAST_Descriptor** es un ente de modelado que constituye un descriptor genérico y posiblemente parametrizable que define la información que se necesita para describir el modelo de tiempo real de algún tipo de recurso o servicio del sistema. El descriptor proporciona la información semántica y cuantitativa que es común a todos los entes que puedan resultar de su instanciación.

Un **MAST_Instance** es un ente que constituye el modelo de tiempo real concreto y final de una instancia individual de un recurso o servicio del sistema. En el modelo de un sistema, cada objeto del tipo MAST_Instance que se declare debe tener una clase del tipo MAST_Descriptor que describa su naturaleza y semántica, y del que se deriva al establecer valores o instancias concretas a todos los atributos, parámetros o referencias que tenga declarados.



Un elemento **MAST_Component_Descr** es un MAST_Descriptor que describe una clase de elementos de modelado que describen un tipo de recurso hardware o software del sistema. Un MAST_Component_Descr tiene una triple función:

1) Constituye un elemento contenedor que declara:

_ Los atributos, parámetros o símbolos que deben ser establecidos para declarar una instancia del modelo. De cada uno de ellos se especifica su tipo y optativamente el valor por defecto que se les asignará.

_ Declara los componentes externos a él (y sus tipos) que deben ser referenciados ya que el modelo del componente hace uso de los modelos que representan.

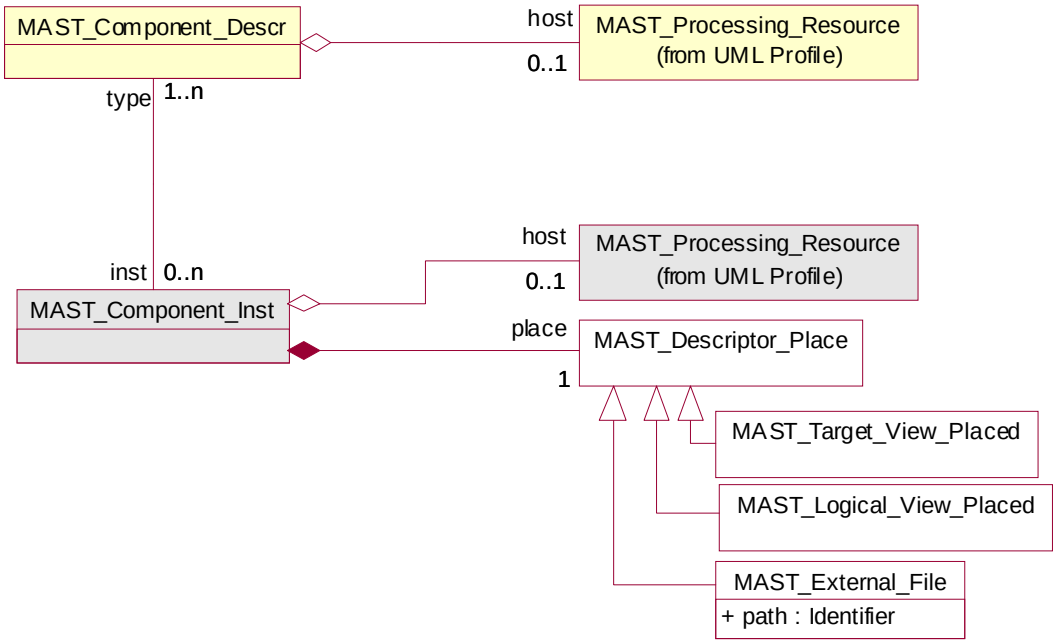
_ Declara los componentes (y sus tipos) que se instancian, por cada instancia suya que se declare en el modelo.

_ Declara los modelos de los usos del componente que son requeridos en la ejecución del sistema. Estos usos se describen individualmente como modelos de operaciones o bien agrupados por dominios de funcionalidad en interfaces.

2) Define un ámbito de visibilidad, para nombres de símbolos, atributos y componentes. Cualquier atributo, símbolo o componente que se defina en un componente es visible y utilizables en la definición de cualquier componente incluido dentro de él (salvo que se oculte por una declaración mas próxima que haga uso del mismo identificador).

3) Cada MAST_Component_Descr tiene una referencia denominada “host” a un componente del tipo predefinido en UML "MAST_Processing_Resource" y hace referencia al modelo del processing resource en el que estará instalado el componente.

Un elemento **MAST_Component_Inst** es una instancia concreta de un tipo de ente de modelado descrito mediante un MAST_Component_Descr. Contiene la información cuantitativa completa y final que define el modelo de tiempo real del ente hardware o software que modela.



Cuando se declara en un modelo un MAST_Component_Inst:

- Quedan declarados recursivamente a la vez todos las instancias de componentes que estaban declarados en la MAST_Component_Descr como elementos agregados.
- Se le tiene que asignar a cada parámetro, atributo o símbolo un valor concreto que sea resoluble dentro del modelo. Si en el correspondiente MAST_Component_Descr el parámetro, atributo o símbolo tenía declarado valor por defecto, puede omitirse de la declaración del MAST_Component_Inst si el valor que se le asigna es el declarado como por defecto.
- Se tiene que asignar a cada elemento declarado por referencia en el correspondiente MAST_Component_Descr, la referencia de un MAST_Component_Inst que esté declarado en el modelo.
- Debe asignarse a la referencia “host” el componentes de tipo MAST_Processing_ Resource en que se instancia el componente (siempre que no se le asigne el valor por defecto). La asignación de la referencia host sigue las siguientes reglas:

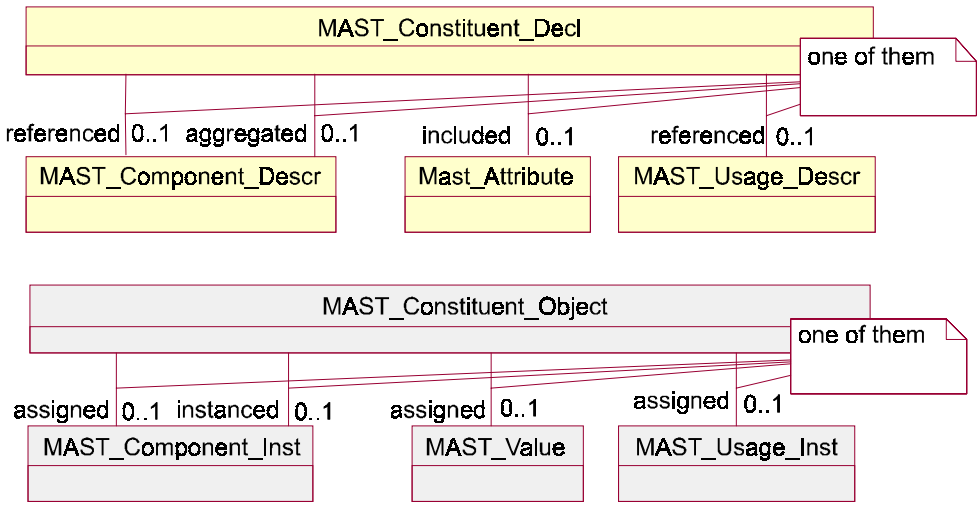
  _ Cuando una MAST_Component_Inst se declara explícitamente en un modelo siempre debe asignarse la referencia de la instancia MAST_Processing_Resource en la que se instale.

  _ Cuando una MAST_Component_Inst se declara implícitamente como agregada dentro de otro, su referencia “host” tiene como valor por defecto el mismo objeto

(del tipo MAST_Processing_Resource) referenciado en la referencia “host” del componente en el que se declara.

Cuando un componente es del tipo MAST_Processing_Resource la referencia “host” se referencia a sí mismo.

- Debe asignarse al atributo “place” el valor que permite localizar la descripción del MAST_Component_Descr del que es instancia. Las posibles ubicaciones son la Target_View o la Logical_View del propio componente, o en un fichero externo, al que se puede acceder a través del “path” que se declara.



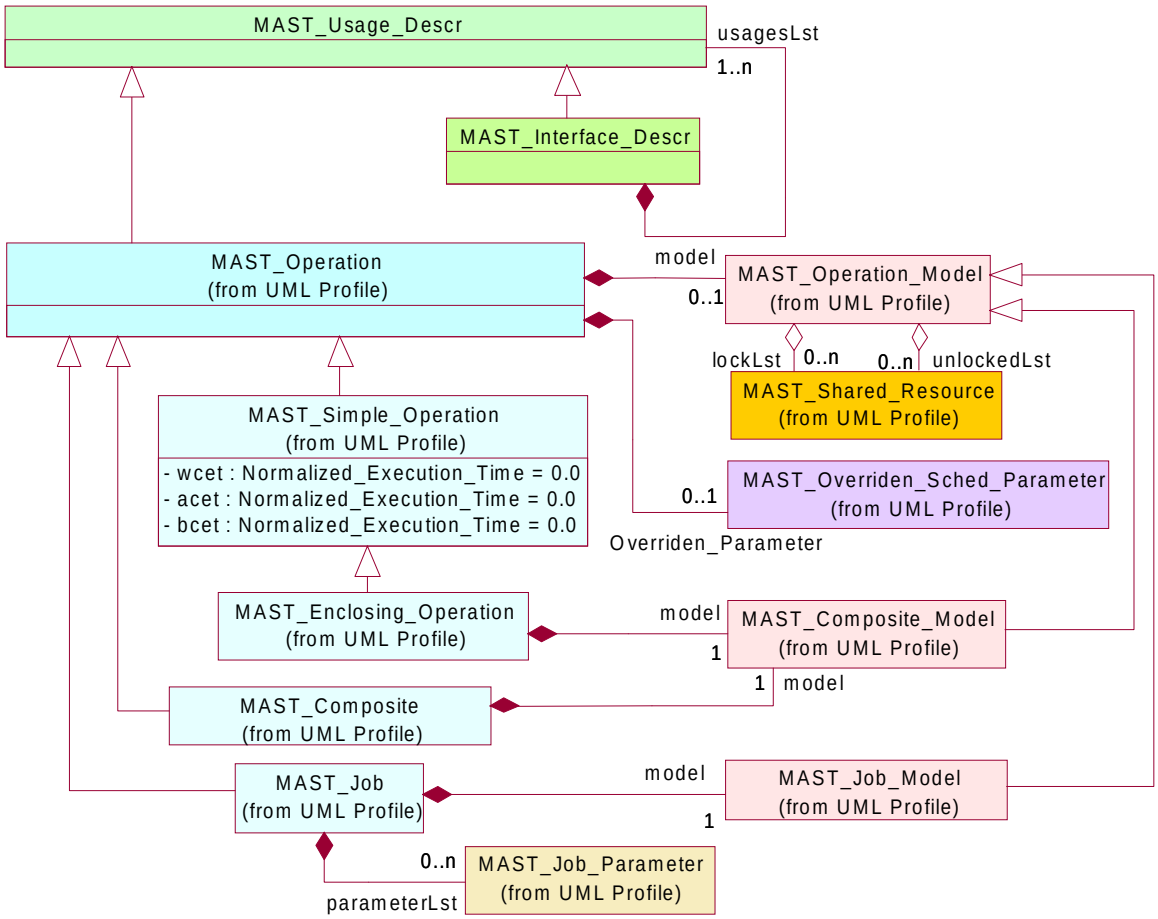
Una **MAST_Constituent_Decl** es una referencia a algún ente (componente, tipo de uso o atributo) que se declaran en un MAST_Component_Decl como elemento utilizado internamente para construir el modelo de tiempo real del componente. Existen tres tipos de declaraciones:

- Aggregated MAST_Component_Descr que declara el tipo de un componente incluido en él que será instanciado por cada instancia del componente que se describe.
- Referenced MAST_Component_Descr que declara un componente externo a él, que necesita conocerse, ya que su modelo es parte del modelo del componente que se describe. La instanciación del componente referenciado no es inducida por la instanciación del componente que la referencia.
- Referenced MAST_Usage_Descr que representa un tipo de uso incluido en un componente diferente y cuyo modelo se necesita conocer para construir el modelo del componente que se describe.
- Included MAST_Atribute que representa un tipo de datos cuyos valores necesitan conocerse para construir el modelo del componente que se describe.

Un **MAST_Constituent_Object** es cada una de las instancias concretar que tienen que existir para que esté completo el modelo de la instancia MAST_Component_Inst que se declara. En correspondencia con las MAST_Instance_Declaration que se definen en cada declaración de un componente, en la instancia del mismo se puede tener que especificar:

- Instanced MAST_Component_Inst que representa la instancia que crea como consecuencia de la declaración aggregated MAST_Component_Descr establecida en el componente que describe al componente que se instancia. Esta declaración solo se hará en la instancia si en el descriptor existián elementos abstractos que se necesitan concretar o se desean asignar valores diferentes a los establecidos como por defecto.
- Assigned MAST_Component_Inst que representa la referencia a un componente que se encuentra instanciado en el modelo y que se declara en correspondencia a la declaración referenced MAST_Component_Descr establecida en el componente que describe al componente que se instancia.
- Assigned MAST_Value que representa el valor que se asigna al atributo declarado por el included MAST_Attribute attribute establecido en el componente que describe al componente que se instancia.
- Assigned MAST_Usage_Inst que representa la referencia a un MAST_Usage_Inst que se obtiene por referenciar un MAST_Usage_Inst de un MAST_Component_Inst y asignar un nuevo valor concreto, si fuese preciso, al parámetro del uso.

Si un componente referenciado, agregado o un atributo tiene definidos valores por defecto, se le pueden definir o no nuevos valores concretos cuando se realiza su instanciación. Si se asumen todos los valores por defectos y con ellos el componente o uso queda completamente definidos, no es necesario declarar el elemento instanciado en la instancia que se describe.



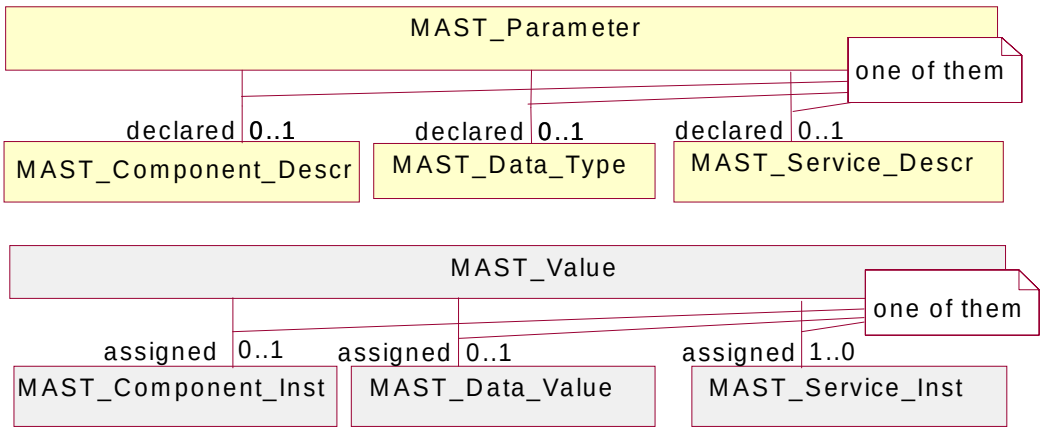
Un **MAST_Usage_Descr** describe el modelo de tiempo real de una forma de uso de un componente. Corresponde, por ejemplo, al modelo de la ejecución del código software que se realiza en respuesta a la invocación de un procedimiento lógico o al modelo de un uso de un recurso hardware que se realiza en background (por ejemplo atención al timer) o en respuesta de una interrupción o acceso a algún estado interno.

Un MAST_Usage_Descr se puede declarar individualmente en el componente como MAST_Operation o agrupado por su dominio de utilización en elementos contenedores del tipo MAST_Interface.

Una **MAST_Inteface** es un conjunto de tipos de usos asociados por pertenecer a un mismo dominio funcional. Una MAST_Interface puede incluir bien un conjunto de MAST_Operations o de otras MAST_Interfaces más simples. Es meramente un elemento contenedor, que permite organizar los servicios y sobre todo proporcionar un ámbito específico de nombres y de parámetros. Los nombres de parámetros y de operaciones dentro de una interfaces tienen que ser únicos, sin embargo parámetros u operaciones con el mismo nombre pero incluidos en diferentes interfaces son diferentes.

Un **MAST_Operation** es un tipo de uso simple que corresponde a la ejecución de un procedimiento de un sistema con unos datos y unas condiciones de ejecución específicas y es un concepto definido en el perfil MAST_UML.

Aunque en el diagrama de clases se muestra la estructura del metamodelo que describe MAST_Operation, y el conjunto de clases que se derivan de ellas, todas ellas pertenecen al UML_MAST profile y están descritas en su documentación.



Un MAST_Usage_Descr puede tener definidos un conjunto de parámetros **MAST_Parameter** que son utilizados en la descripción del modelo de la forma de uso. Estos parámetros permiten especificar el recurso o forma de uso concreto que se utiliza en algún punto del modelo de una operación, como consecuencia de que el procedimiento que se modela se ejecuta con diferentes datos o recursos. Los MAST_Parameter pueden utilizarse para declarar:

- Cada uno de los MAST_Component_Descr que son referenciados para formular el modelo de la forma de uso.
- Otros MAST_Usage_Descr a los que son referenciados para describir el modelo.

- MAST_Data_Type que referenecia datos de tipo escalar o estructurado, que son utilizados para definir el modelo de la forma de uso.

En la declaración de un MAST_Parameter se pueden definir valores por defecto.

La declaración de las MAST_Interface_Inst está implícita en la declaración del MAST_Component_Inst en que están declaradas (y por tanto no existen declaraciones explicitas de las MAST_Interfaces_inst). En consecuencia, cuando se declara una MAST_Component_Inst hay que asignar a todos los MAST_Parameter de las MAST_interfaces declaradas en él, un MAST_Value que es un valor concreto (o instance) con el que se define completamente el modelo de la interfaz, lo cual constituye su instancia. Si el valor MAST_Value que se debe asignar al Mast_Parameter es el establecido por defecto en el MAST_Interface_Descr, la asignación no necesita ser explicitada.

I.7. Objetivos del proyecto

A lo largo de los tres últimos años, he participado en dos proyectos de investigación[1] desarrollados por la empresa Equipos Nucleares en colaboración con el grupo Computadores y Tiempo Real (CTR), relativos a la aplicación de la visión artificial a la automatización de los procesos industriales y al desarrollo de controladores de sistemas robotizados. En ellos, he tenido que llevar a cabo la adaptación de diferentes librerías comerciales (relativas a visión artificial (digitalización de imágenes y procesado digital de imágenes), acceso remoto a bases de datos Informix, driver de control de tarjetas de I/O analógicas y digitales) al entorno en que se desarrollaban los proyectos, que eran plataformas Windows NT y aplicaciones de tiempo real desarrolladas con tecnología Ada.

En este proyecto fin de carrera hay dos **objetivos principales** que vertebran el desarrollo del proyecto y la exposición del mismo en la memoria:

1) Diseño e implementación de tres componentes de tiempo real útiles en diseño de controladores industriales: Se han desarrollado un conjunto de interfaces correspondientes a tres dominios de implementación, y se han desarrollado tres componentes que las implementan.

- Dominio **Adquisición de señales analógicas y digitales**: Se han diseñado un conjunto de interfaces destinadas a la adquisición de señales analógicas y digitales a través de tarjetas de IO de propósito general.

  _ Interface **I_Adq_Configuration**: Conjunto de operaciones y constantes de gestión y configuración de la tarjeta de adquisición.

  _ Interface**I_Analog_Adq**: Conjunto de operaciones de adquisición y generación de tensiones analógicas a través de convertidores A/D y D/A.

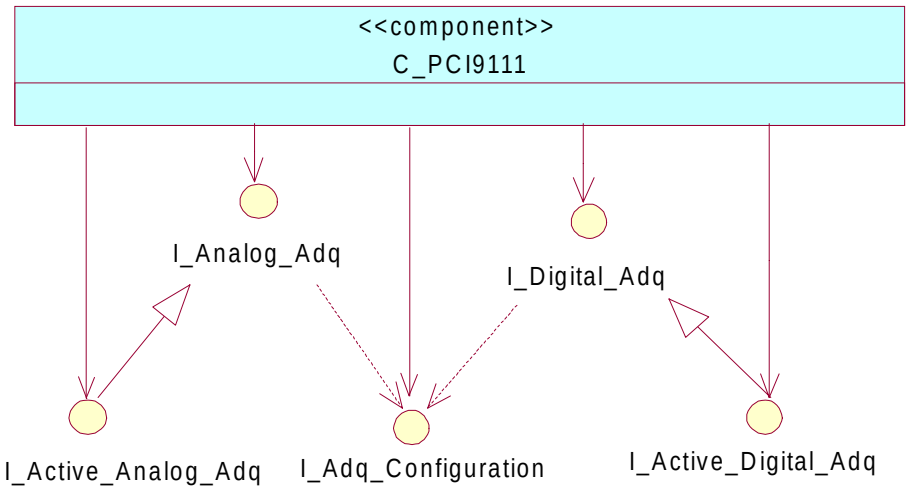
1. "Entorno para la programación de tiempo real de controladores industriales y aplicación a la automatización del proceso de fabricación de generadores de vapor en cámara limpia". Fondos FEDER, 1999-2001.
   "Sistema robótico teleoperado (SRT)". Financiado por Equipos Nucleares S.A.ENDESA y REN, 1999-2000.

_ Interface**I_Adq_Analog_Active**: Interfaz activa que añade a la interfaz básica I_Analog_Adq un conjunto de operaciones basadas en tareas de background que temporizan la adquisición o establecimiento de tensiones analógicas, las procesan y gestionan los resultados.

_ Interface**I_Digital_Adq**: Conjunto de operaciones de lectura y escritura en líneas y grupos de líneas de entrada y de salida digitales.

_ Interface**I_Active_Digital_Adq**: Interfaz activa que añade a la interfaz básica I_Digital_Adq un conjunto de operaciones basadas en tareas de background que leen y escriben las líneas digitales, procesan la información y gestionan los resultados.

_ Componente **C_PCI9111**: Componente basado en la tarjeta de adquisición PCI-9111DG de la casa ADLINK, con 1 canal de salida analógico de 12 bits de resolución, 16 canales de entrada analógicos multiplexados de 12 bits de resolución, 16 bits digitales de entrada, 16 bits digitales de salida y 4 bits digitales de entrada / salida. Este componente implementa las 5 interfaces antes citadas.

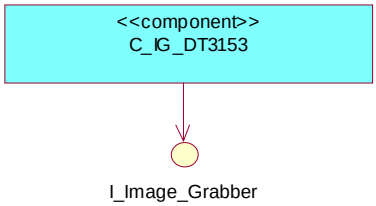


- Dominio: **Adquisición y digitalización de imágenes de vídeo**: Conjuntos de recursos para la configuración de la tarjeta de digitalización de imágenes de vídeo, captura de imágenes, y transferencia de imágenes en vivo a ventanas de la workstation.

  _ Interface:**I_Image_Grabber**: Ofrece la capacidad de controlar y configurar la tarjeta, capturar y digitalizar imágenes individuales, convertirlas en otros formatos y activar o inhibir la transferencia continua de imágenes hacia ventanas de la aplicación

  _ Componente: **C_IG_DT3153**: Contiene los recursos software de gestión de la tarjeta DT3153 de Data Translation que es una tarjeta digitalizadora de imágenes de

vídeo en color con capacidad de transferencia de imágenes a memoria de la workstation. Este componente implementa la interface I_Image_Grabber .



- Dominio **Procesado digital de imágenes.** Corresponde a diferentes recursos para la gestión, procesado digital, análisis, caracterización estadística etc. de imagenes almacenadas en el ordenador. Para este dominio se han definido 8 interfaces que ofrecen cada una de ellas un tipo de servicios específico:

  _ Interface**I_Img_Managing**: Ofrece el conjunto básico de operaciones de gestión y almacenamiento de imágenes. Es una interfaz que ofrece los recursos básicos sobre las estructuras de datos asociadas a una imagen, así como los procedimientos de almacenamiento y recuperación de la misma. Es utilizada por las restantes interfaces de visión.

  _ Interface**Img_Processing**: Ofrece el conjunto de operaciones que permiten el procesado de las imágenes por parte del usuario. Ofrece procedimientos de acceso a píxel y de transformación de la imágenes en función de la posición de los pixels.

  _ Interface**I_Img_Logic_Processing**: Ofrece el conjunto de operaciones de procesado lógico (bit a bit) de imágenes de grises.

  _ Interface**I_Img_Arith_Processing**: Ofrece el conjunto de operaciones de procesado aritmético de imágenes de grises.

  _ Interface**I_Img_Transforming**: Ofrece el conjunto de operaciones de transformaciones de imágenes de un modo global. Se ofrecen transformaciones geométricas, lineales, morfológicas y filtrados.

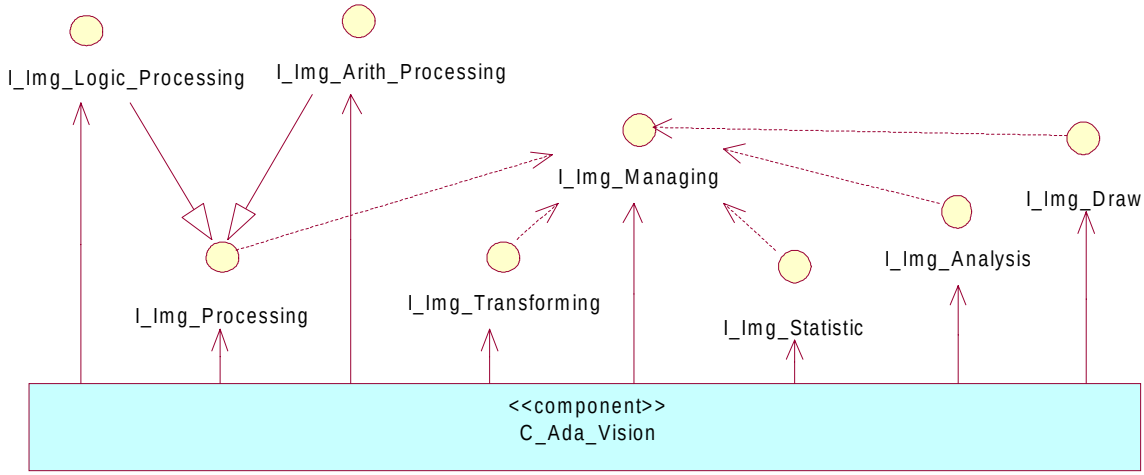
  _ Interface**I_Img_Statistic**: Ofrece el conjunto de operaciones de cálculo de valores estadísticos sobre una imagen de grises.

  _ Interface**I_Img_Analysis**: Ofrece el conjunto de operaciones que realizan análisis complejos sobre imágenes para obtener resultados concretos. Permite identificar, localizar y caracterizar patrones presentes en la imagen.

  _ Interface**I_Img_Draw**: Ofrece de operaciones que permite dibujar e insertar distintos elementos en una imagen.

  _ Component **C_Ada_Vision**: Este componente ofrece los recursos relativos a gestión, procesado de imágenes y análisis de imágenes. Su funcionalidad corresponde a las librerias IPL [18] y Open_CV [19] de código abierto

proporcionadas por Intel y que han sido diseñadas específicamente para que sean muy eficientes sobre la arquitectura de los procesadores con tecnología MMX.



2) Desarrollo de una aplicacion de tiempo real ejemplo y su análisis de planificabilidad de peor caso: Se ha desarrollado como ejemplo una aplicación sencilla que hace uso de algunas de las interfaces que son implementadas por los tres tipos de componentes desarrollados y que consiste en el control de una maqueta de trenes de juguete, a través de la detección de la posición de los trenes mediante visión artificial. Esta aplicación desarrollada tiene la finalidad de validar los componentes, tanto en lo relativo a su especificación, a su interconectividad y a sus modelos de tiempo real, y sobre todo permite obtener experiencia sobre el ciclo de desarrollo de aplicaciones basadas en componentes.

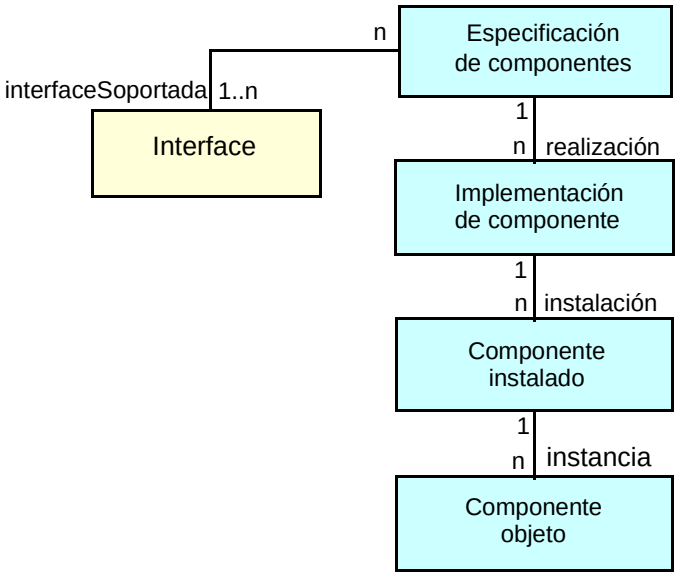
El proyecto fin de carrera ha tenido otros objetivos indirectos, que se enmarcan dentro de las líneas de investigación que desarrolla el Grupo de Computadores y Tiempo Real, y cuya importancia transciende mas allá del propio proyecto. Estos objetivos han sido:

- Se ha establecido estructuras de datos y estrategias para representar componentes de tiempo real en la herramienta UML CASE ROSE de Rational.
- Se proponen unos formatos para codificar interfaces y componentes implementados como paquetes Ada'95.
- Se han analizado los procesos de transformación de los códigos ADA que representan los modelos funcionales de los componentes en las diferentes fases del desarrollo de la aplicación, lo cual representa una información muy valiosa para el desarrollo de futuras herramientas de generación automática de código a partir de los modelos UML.
- Se han analizado los procesos de construcción de los modelos de tiempo real de una aplicación a partir de su MAST_RT_View y del los modelos de tiempo real de los componentes que se utilizan. Información que será la base, o al menos una experiencia relevante a fin de desarrollar en el futuro herramientas de generación automática del modelo de tiempo real MAST de las aplicaciones.
- Se ha desarrollado modelos de tiempo real de una plataforma contruida por una workstation operando bajo sistema operativo Windows NT Server 4.0, y diferentes tarjetas de IO (Grabber DT 3153 y IO card PCI-9111DG).

II. ESPECIFICACIÓN Y DISEÑO FUNCIONAL DE COMPONENTES

II.1. Componentes e interfaces

La palabra componente representa muchos conceptos diferentes dentro de CBSE y es importante fijar el significado que le damos en este proyecto. En la siguiente figura, se muestran diferentes formas en las que se puede tratar un componente.



La especificación de un componente es un descriptor del comportamiento de un conjunto de objetos componentes y define una unidad de implementación. El comportamiento de la especificación de un componente se describe mediante un conjunto de interfaces.

Una interface define los diferentes aspectos del comportamiento de un componente relativos a un mismo dominio que se ofrecen como una unidad o servicio que puede ser ofrecido de forma intercambiable por diferentes componentes.

Una implementación de un componente es una realización de una especificación de componente que puede ser utilizado en el desarrollo de una aplicación.

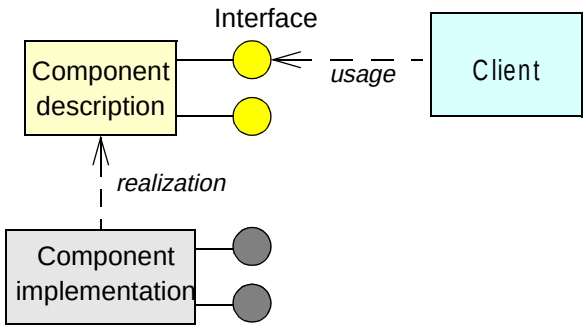
Un componente instalado o desplegado es una copia de una implementción de un componente que está registrado en algún entorno de ejecución. Esto capacita al entorno de ejecución para identificarlo y usarlo cuando se genere una instancia del componente o se invoque alguna de sus operaciones.

Un componente objeto es una instancia ejecutable de un componente instalado. Es un concepto de ejecución. Es un objeto con una única identidad y con su propia instancia de datos y estado. Un componente instalado puede tener muchos (o solo un) componentes objetos instanciados, cada uno con su propia identificación.

En este proyecto, se utilizan los siguientes conceptos:

- Descripción de componente o de interface: son modelos independientes UML que describen su funcionalidad. En el modelo de un componente aparecen agregados los modelos de las interfaces que implementa.
- Implementación de un componente: es un un paquete Ada compilable, que implementa a su vez la descripción de un componente y las de las interfaces que ofrece el componente.
- Componente objeto: es una librería instanciada en una aplicación Ada del paquete Ada que describe al componente.

Es importante tener en consideración los dos tipos de "contrato" que implica el manejo de componentes:



- Contrato de uso: Es el contrato que se establece entre la especificación del componente y la aplicación cliente que lo utiliza. Esto se realiza a través de las interfaces que definen de por sí y con independencia del componente que las implementa y ofrece, las operaciones y el modelo de información del servicio que ofrece. Cuando una aplicación usa una interface, su funcionalidad no puede quedar afectada por la naturaleza del componente que implementa la interface que se usa.
- Contrato de instalación: Es el contrato que se establece entre la implementación del componente y el entorno en el que puede operar. Debe establecer que entorno de implementación y que otros componentes o interfaces deben estar implementados en él para que el componente pueda ofrecer su funcionalidad.

Aunque estos dos tipos de contratos deben estar detalladamente explicitados en la descripción de un componente, deben ser independientes entre si. La aplicación que usa un componente no debe depender, ni necesita conocer el contrato de realización, y la instalación de un componente debe ser independiente de que aplicación va a utilizarla.

Desde este punto de vista, la especificación del componente y la especificación de las interfaces son conceptos independientes que juegan un papel central en la tecnología CBSE. La especificación del componente es básicamente el contrato de instalación, que se formula principalmente pensando en el realizador, probador y ensamblador del componente. Define al componente como una unidad de realización, define su encapsulación y determina su granularidad y la posibilidad de reemplazabilidad que tiene. Por el contrario, la especificación de las interfaz representa el contrato con el componente cliente, y define los servicios que el

componente usuario puede esperar del componente que la implementa. En la siguiente tabla, se resumen las diferencias entre las descripción de un componente y de una interfaz.

Especificación de Interfaz	Especificación de Componente
Una lista de operaciones	Una lista de interfaces soportadas
Define el modelo abstracto de información lógica que se necesita para usarla	Define el modelo de información que establece las relaciones entre las diferentes interfaces.
Representa el contrato con el cliente.	Representa el contrato con el realizador, instalador, probador y ensamblador.
Especifica como las operaciones afecta o dependen del modelo de información	Define la implementación y la unidad de ejecución.
Describe solo los efectos locales al modelo de información del servicio que oferta.	Especifica como se implementan las operaciones que ofrece en función de las operaciones que requiere de las interfaces del que el componente depende.

En las siguientes secciones de este capítulo se describen los recursos y los formatos que se proponen para formular las especificaciones y las implementaciones de los componentes y de las interfaces.

II.2. Especificación de una interfaz

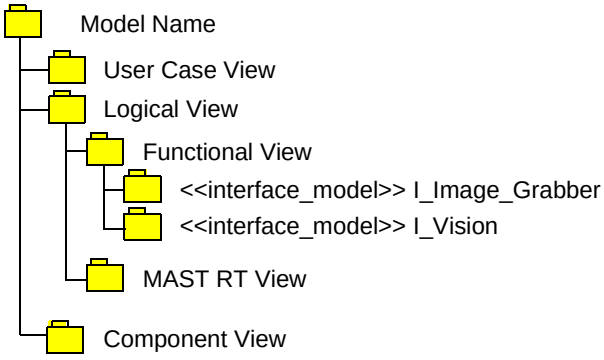
La interfaz representa el conjunto de operaciones y modelo de información que se requiere para definir completamente el servicio que se espera del componente que las oferte y lo que el cliente debe espera de su uso.

La especificación de una interfaz contiene:

- El identificador de la declaración de la interface al que se hace referencia en su instanciación dentro de un componente.
- El modelo de información, esto es, los atributos, tipos, asociaciones, etc. definidos unos en función de otros hasta proporcionar un modelo completo y cerrado.
- La descripción de las operaciones a través de los que el usuario puede acceder al servicio proporcionado por la interfaz. Esto supone definir la especificación completa de cada operación (identificador, parámetros, precondiciones y postcondiciones, etc).
- Conjunto de invariantes y restricciones del modelo de información de la interfaz.
- Otras interfaces con las que tiene establecidas relaciones de dependencia o de herencia.

Una interfaz se describe con independencia de cualquier componente. En UML la representamos, como una clase con estereotipo <<interface_descr>> y su modelo de información se describe dentro de un paquete que contiene todos los elementos UML que se utilizan para su descripción.

El modelo de una interfaz es siempre un paquete con el estereotipo <<interface_model>> que tiene el nombre de la interfaz, y que se encuentra situado dentro del paquete Functional_View , que a su vez esta definido en Logical View del modelo.



El paquete de la interfaz puede estar incluido dentro del modelo que describe un componente, pero mas frecuentemente se encuentra descrita en un fichero (modelo) independiente que describe solo a la interfaz o también a otras interfaces relacionadas.

En la descripción UML de una interfaz se incluye:

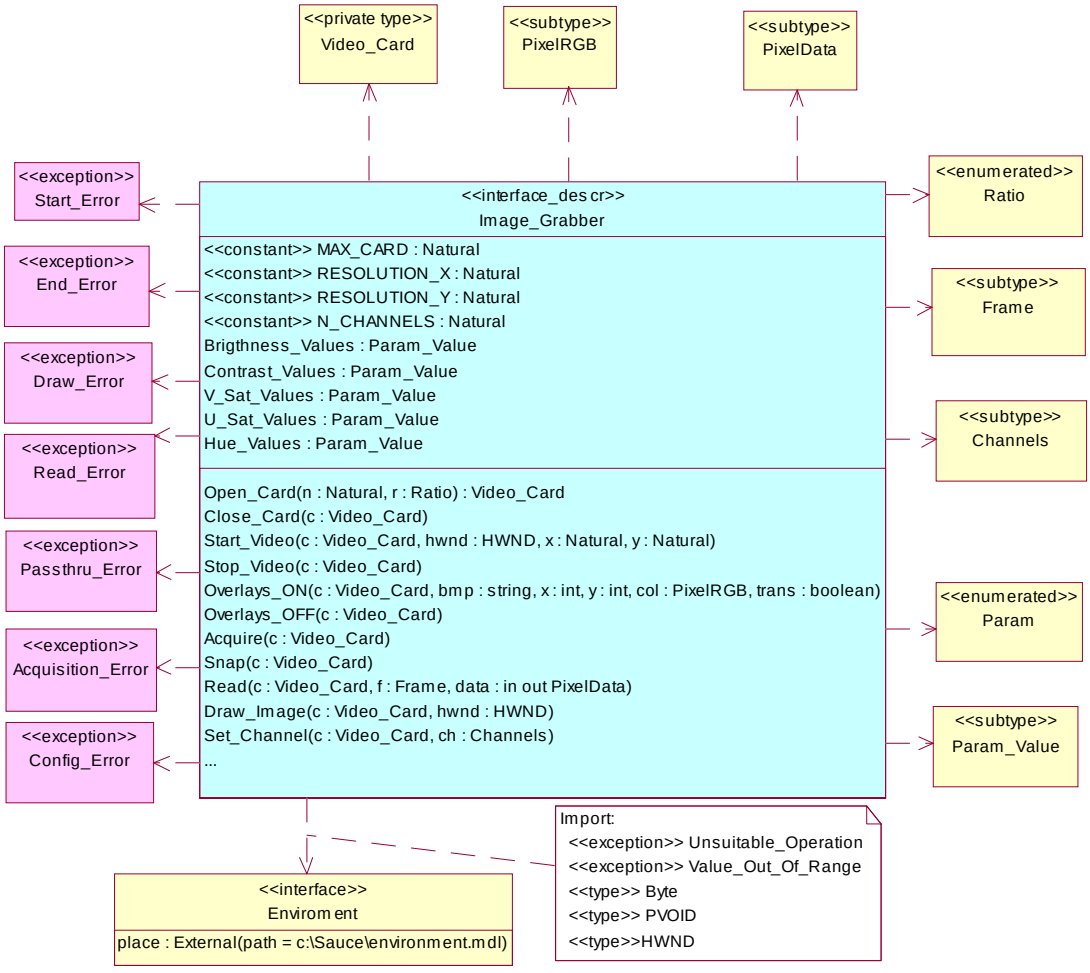
- Identificador de la interfaz: Corresponde con el nombre de la clase con estereotipo <<interface_descr>> en que se declara la interfaz.
- Modelo de informacion de la interfaz: Corresponde con atributos, clases asociadas (por agregación o por dependencia), diagramas de estado, etc. que se incluyen en el paquete de descripción de la interfaz y que proporciona toda la información necesaria para que el usuario pueda utilizar la interfaz de forma autocontenida.
- Declaración de las operaciones de la interfaz: Se formulan como operaciones declaradas en la clase de descripción de la interfaz. Cada operación se declara con su descripción completa UML, esto es, con su identificador, su conjunto de parámetros, y en su caso valor de retorno. Así mismo, en formato de texto se incluyen el conjunto de pre-condiciones (condiciones relativas a la interfaz que deben cumplirse antes de ser invocada una operación a fin de que los resultados sea predecibles) y el conjunto de post-condiciones (condiciones que se satisfacen en la interfaz despues de haber sido ejecutada la operación, si se satisfacian las pre-condiciones).
- Declaración de invariantes y restricciones en la interfaz: Se formulan mediante texto asociado a la clase que declara la interfaz.
- Otras interfaces con la que tiene establecida relaciones de dependencia: Se formulan como clases que representan instancias de interfaces, en las que solo se describe su identificador y el path donde se encuentra su declaración si esta está en un fichero diferente al del modelo.
- Acceso al fichero (de extensión ".aci"): que contiene la declaración de la parte pública del paquete Ada en que se describe la interfaz, mediante lenguaje Ada. El código de

este fichero contiene la información necesaria para que el programador de la aplicación pueda programar las sentencias de acceso a los servicios de la interface, sin embargo no podrá compilar con ella su aplicación, ya que:

_ Falta la parte privada y la declaración de los tipos privados, que serán función de su implementación dentro de un componente.

_ Pueden faltar constantes y tipos propios del dominio que deben ser establecidos también en la implementación.

En el siguiente diagrama de clases, se muestra una vista de la declaración de la interfaz Image_Grabber. En esta vista, algunas de las clases que forman parte del modelo de información de la interfaz no están explícitas, por lo que para completar el modelo, es necesario recurrir tambien a otras vista, complementarias y a la documentación y diagramas de estado y de actividad asociadas con cada clase.



La interfaz Image_Grabber que se propone como ejemplo ofrece los recursos para digitalizar y adquirir imágenes de vídeo. Es independiente de cualquier tarjeta hardware y ofrece servicios para inicializar y liberar los recursos que en cada caso se requiera, configurar

el proceso de digitalización de imágenes, transferir imágenes en vivo a una ventana de la aplicación, transferir una imagen digitalizada a memoria, etc.

El modelo de información necesario para hacer uso de esta interfaz, se describe constantes y atributos declarados en la clase y los tipos de datos y excepciones declarados en las clases asociadas.

En la declaración de la interfaz Image_Grabber, se indica que esta interfaz tiene una dependencia de la interfaz Environment, de la que importa dos tipos de excepciones de mayor ámbito y el tipo relativo a los manejadores de ventanas Win32.

En el siguiente código, se muestra el fichero tipo ".aci" que formula la declaración de la interface desde un punto de vista Ada. Se obtiene a partir de la especificación abstracta UML y consiste en un fichero no compilable de declaraciones con la siguiente estructura:

- Dependencias externas: interfaces necesarias de "Componentes externos".
- Constantes y subtipos: las constantes no inicializadas y los subtipos no definidos.
- Variables de estado: no inicializadas.
- Operaciones.
- Excepciones locales.
- Parte privada: no implementada.

```
--*****************************************************************************
--*                                                                           *
--* Declaration of Abstract MAST-Component Interface (ACI)                    *
--* ACI Name:  Image_Grabber                                                  *
--*                                                                           *
--*                                                                           *
--* Documentation:                                                            *
--*     Conjunto de operaciones de gestión y manejo de la tarjeta de          *
--*     adquisición de imágenes de vídeo.                                     *
--*                                                                           *
--*****************************************************************************
--=============================================================================
-- External dependency
with Environment;
--=============================================================================
package Image_Grabber is

--=============================================================================
-- Declaration of exported constant, types and procedures                   =
--=============================================================================

-----------------------------------------------------------------------------
-- Constantes y tipos
-----------------------------------------------------------------------------

-- Documentation
--     Número máximo de tarjetas que pueden estar registradas
--     simultánemente en un computador.
MAX_CARD: constant Natural;

-- Documentation
```

```
--     Número máximo (pixels) de columnas de las imagenes adquiridas.
RESOLUTION_X: constant Natural;

-- Documentation
--     Número máximo (pixels) de columnas de las imagenes adquiridas.
RESOLUTION_Y: constant Natural;

-- Documentation
--     Número de canales de entrada de video de la tarjeta.
N_CHANNELS: constant Natural;


-- Documentation:
--     Identificador de la tarjeta de adquisición.
type Video_Card is private;

-- Documentation:
--     Escalado del tamaño de las imagenes que se
--     capturaran en la tarjeta.
type Ratio;

-- Documentation:
--     Pixel RGB en formato: R-G-B.
type PixelRGB;

-- Documentation:
--     Píxels almacenados en las posiciones x,y.
type PixelData;

-- Documentation:
--     Canales de entrada de video de la tarjeta.
type Channels;

-- Documentation:
--     Parametros de la imagen.
type Param;

-- Documentation:
--     Valores maximo , minimo y nominal (por defecto)
--     de un parametro.
type Param_Value;

-------------------------------------------------------------------------
-- Variables de estado
-------------------------------------------------------------------------

-- Documentation
--     Valor del brillo.
Brigthness_Values: constant Param_Value;

-- Documentation
--     Valor del contraste.
Contrast_Values: constant Param_Value;

-- Documentation
--     Valor de la saturacion V.
V_Sat_Values: constant Param_Value;

-- Documentation
--     Valor de la saturacion U.
U_Sat_Values: constant Param_Value;

-- Documentation
```

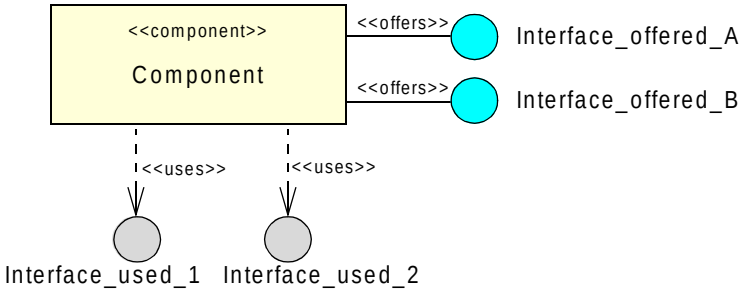
```
--     Valor de la intensidad.
Hue_Values: constant Param_Value;

-------------------------------------------------------------------------
-- Operaciones
-------------------------------------------------------------------------

-- Documentation:
--     Inicializa la tarjeta y reserva la memoria necesaria para dos
--     buffersde almacenamiento: uno para imágenes del vídeo en vivo y otro
--     para capturade imágenes estáticas. También se inicializan los
--     valores del estado de losparametros de vídeo para cada uno de los
--     canales (que pueden ser de distintovalor en un momento dado).
--     Eleva la excepcion Start_Error si se produce un error.
function Open_Card(n:Natural;r:Ratio) return Video_Card;

-- Documentation:
--     Libera los recursos asociados a la tarjeta.
--     Eleva la excepcion End_Error si se produce un error.
procedure Close_Card(C: in out Video_Card);

-- Documentation:
--     Comienza a mostrar el vídeo en vivo en la ventana hwnd.
--     La imagen aparece escalada con dimensiones x por y pixels.
--     La resolución máxima es RESOLUTION_X por RESOLUTION_Y
--     pixels.
--     Se eleva la excepción Passthru_Error si se produce un error
procedure Star_Video(C: in out Video_Card;
                     hwnd:I_Environment.HWND;
                     x:natural;
                     y:natural);

-- Documentation:
--     Se detiene el video en vivo.
--     Se eleva la excepción Passthru_Error si se produce un error
procedure Stop_Video(C: in out Video_Card);

-- Documentation:
--     Muestra una imagen superpuesta sobre el video en vivo
--     bmp es el nombre del fichero donde se encuentra la imagen
--     x e y son las dimensiones de la imagen de video que se
--     desea solapar; han de ser iguales o menores que los valores
--     de los parametros de la funcion VideoON
--     col indica el color de los pixels de la imagen que
--     desaparecen para permitir observar el video en vivo
--     Si translucent=true , la imagen superpuesta es toda
--     ella translucida.
--     Se eleva la excepción Passthru_Error si se produce un error
procedure Overlays_ON(C: in out Video_Card;
                      bmp:in string;
                      x:in integer;
                      y:in integer;
                      col:in PixelRGB;
                      trans:in boolean);
-- Documentation:
--     Finaliza el overlay
--     Es necesario invocar la funcion con el video en vivo ya detenido
--     Se eleva la excepción Passthru_Error si se produce un error
procedure Overlays_OFF(C: in out Video_Card);

-- Documentation:
--     Captura una imagen y la almacena en el buffer de imagen
--     estática con la resolución máxima permitida.
```

```
--     Se requiere que cuando se invoque no esté el video en vivo.
--     Se eleva la excepcion Acquisition_Error si se produce un error
procedure Acquire(C: in out Video_Card);

-- Documentation:
--     Copia en el buffer de la imagen estática la última imagen
--     almacenada en el buffer de vídeo en vivo.
--     Se requiere que cuando se invoque el vídeo esté en vivo.
--     Se eleva la exception Acquisition_Error si se produce un error.
procedure Snap(C: in out Video_Card);

-- Documentation:
--     Transfiere a la memoria principal la imagen capturada
--     anteriormente con Acquire o Snap.
--     Transfiere el rectángulo determinado por los índices de data.
--     Se eleva la excepción Read_Error si se produce un error.
procedure Read(C: in out Video_Card;
               data:in out PixelData);

-- Documentation:
--     Muestra en una ventana la imagen capturada en el buffer de
--     la imagen estática. Es necesario haber capturado previamente
--     una imagen con Acquire o Snap.
--     Se eleva la excepción Draw_Error si se produce un error.
procedure Draw_Image(C: in out Video_Card;
                     hwnd:I_Environment.HWND);

-- Documentation:
--     Selecciona el canal de entrada de vídeo.
--     Se eleva la interrupción Config_Error si se produce un error.
procedure Set_Channel(C: in out Video_Card;
                      ch:Channels);

-- Documentation:
--     Modifica un parámetro de las imágenes capturadas o mostradas
--     en vídeo en vivo en el futuro.
--     Se eleva la excepción Config_Error si se produce un error.
procedure Set_Parameter(C: in out Video_Card;
                      p:Param;
                      value:integer;
                      ch:Channels);

-- Documentation:
--     Devuelve el valor actual de un parámetro de las imágenes
--     capturadas o mostradas en vídeo en vivo en el futuro.
--     Se eleva la excepción Config_Error si se produce un error.
function Get_Parameter(C:Video_Card;
                      p:Param;
                      ch:Channels)return integer;

--=========================================================================
-- Declaration of exported exceptions                                     =
--=========================================================================

Start_Error:exception;
End_Error:exception;
Draw_Error:exception;
Read_Error:exception;
Passthru_Error:exception;
Acquisition_Error:exception;
Config_Error:exception;

--=========================================================================
```

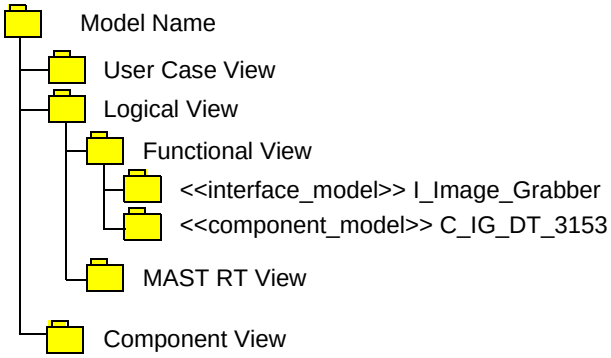
```
   -- Private interface declaration                                          =
   --=========================================================================
   private

   end Image_Grabber;
   --=========================================================================
```

II.3. Especificación de componentes

La especificación de un componente está directamente relacionada con la descripción de los dos contratos que definen su arquitectura. Básicamente debe describir el contrato de uso a través de la declaración de las interfaces que "ofrece" y debe describir el contrato de realización a través de la declaración de las interfaces que "usa".



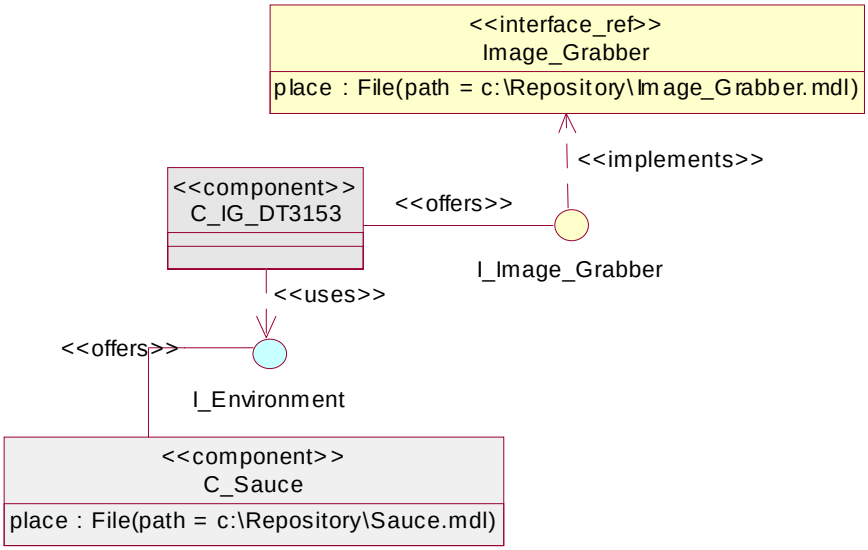
En UML la información que describe un componente se incluye en un paquete con estereotipo <<component_model>> que tiene el nombre de la interfaz, y que se encuentra situado dentro del paquete Functional_View , que a su vez esta definido en Logical View del modelo. Un componente puede estar declarado independientemente en un modelo(fichero con extensión .mdl), o también puede estar varios de ellos agrupados y definidos conjuntamente dentro de un mismo modelo.



Un componente se declara en UML mediante una clase con estereotipo <<component>>, que incluye:

- Nombre del tipo de componente: coincide con el identificador de la clase <<component>>.
- Conjunto de interfaces ofrecidas: Se declaran como atributos o como clases agregadas de con estereotipo <<interface>>. Estas interfaces son instancias de interfaces descritas en el propio modelo o en modelos independientes. En la descripción de cada interface se debe establecer el atributo "place" que describe el modelo en el que se encuentra el descriptor, este toma, por defecto, el valor "model" que hace referencia a que se encuentra descrita en el propio modelo del componente y en el caso de que se encuentre en un modelo externo, toma como valor el path del fichero en que se encuentra. En la declaración de una interface ofrecida, también se explicitan valores iniciales de atributos y tipos que quedaron indefinidos en la declaración de la interface.
- Conjunto de interfaces usadas: Se declaran como instancias de clases asociadas por dependencia. Estas interfaces, deben ser instancias de interfaz declaradas en componentes concretos.
- Restricciones entre interfaces: Restricciones de especificación que se establecen entre las diferentes interfaces del componente como consecuencia de que son todas ellas ofrecidas conjuntamente por el mismo componente.
- Restricciones de acceso a componentes: Restricciones de especificación que limitan las formas de uso con que las diferentes implementaciones pueden acceder a los componentes que se usan para implementar su funcionalidad. Estas restricciones se formulan, bien como texto agregado al componente o a las asociaciones de dependencia.

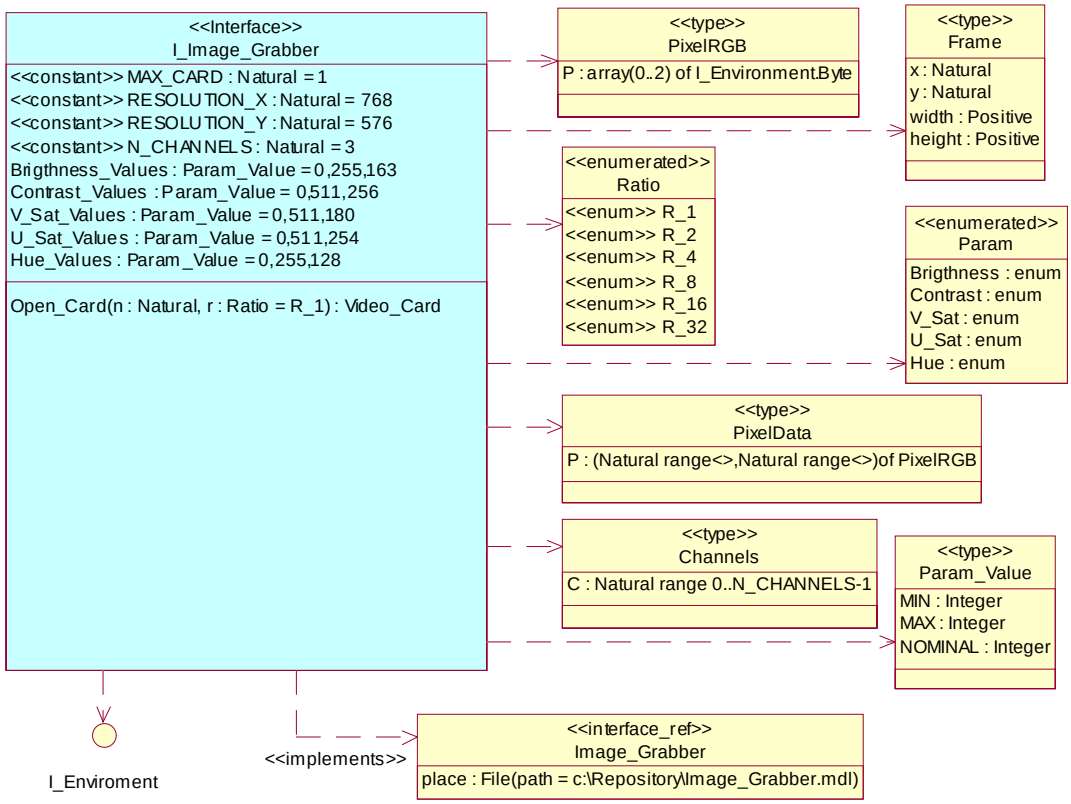
En el diagrama de clases siguiente se muestra la descripción del componente C_IG_DT_3153.



La interface concreta <<interface>>I_Image_Grabber que ofrece, resulta de instanciar la interface abstracta <<interface_descr>>Image_Grabber. Como se muestra en el siguiente

diagrama de clases UML que describe la instancia, esta resulta de establecer los siguientes elementos que en la clase abstarcta estaban indefinidos:

- Se ha establecido que la interfaz del tipo <<interface_descr>>Environment que usa la interfaz concreta <<interface>>I_Image_Grabber del componente C_IG_DT3153 es la interfaz concreta <<interface>>I_Environment ofrecida por el componente C_Sauce que está descrito en el fichero c:\Repository\Sauce.mdl.
- Se ha establecido los valores de la constantes MAX_CARD, RESOLUTION_X, RESOLUTION_Y y N_CHANNELS.
- Se han definido de forma concreta de los tipos: PixelRGB, Frame, Ratio, Param, PixelData, Channels, Param_Value.
- Se han establecido valores iniciales de las variables: Brigthness_Values, Contrast_Values, V_Sat_Values, U_Sat_Values y Hue_Values.
- Se ha concretado el valor por defecto del parámetro r de la operación Open_Card.



En el suguiente listado se muestra el código Ada95 que corresponde a la descripción UML del componente C_IG_DT3153 establecido.

```
--*****************************************************************************
--*                                                                           *
--* Declaration of a MAST-Componet (CMP)                                      *
--* CMP Name:  C_IG_DT3153                                                    *
--*                                                                           *
--*                                                                           *
--* Documentation:                                                            *
--*     Componente de gestión de la tarjeta de adquisición de video DT3153.   *
```

```
--*                                                                          *
--* Implementa la Interface:                                                 *
--*     I_Image_Grabber                                                      *
--*                                                                          *
--****************************************************************************
--****************************************************************************
    -- External dependency
    with C_Sauce;
--****************************************************************************
    package C_IG_DT3153 is

--****************************************************************************
-- Declaration of exported interfaces                                        *
--****************************************************************************

    --************************************************************************
    --*                                                                      *
    --* Declaration of Concrete MAST-Componet Interface (CCI)                *
    --* CCI Name:  I_Image_Grabber                                           *
    --*                                                                      *
    --*                                                                      *
    --* Documentation:                                                       *
    --*     Conjunto de operaciones de gestión y manejo de la tarjeta de     *
    --*     adquisición de imágenes de vídeo.                                *
    --*                                                                      *
    --************************************************************************
    --======================================================================
    -- External dependency
    --with I_Environment;   -- =>  C_SAUCE.I_Enviroment
    --======================================================================
    package I_Image_Grabber is

    --======================================================================
    -- Declaration of exported constant, types and procedures              =
    --======================================================================

    ------------------------------------------------------------------------
    -- Constantes y tipos
    ------------------------------------------------------------------------

    -- Documentation
    --     Número máximo de tarjetas que pueden estar registradas
    --     simultánemente en un computador.
    MAX_CARD: constant Natural:= 1;

    -- Documentation
    --     Número máximo (pixels) de columnas de las imagenes adquiridas.
    RESOLUTION_X: constant Natural:= 768;

    -- Documentation
    --     Número máximo (pixels) de columnas de las imagenes adquiridas.
    RESOLUTION_Y: constant Natural:= 576;

    -- Documentation
    --     Número de canales de entrada de video de la tarjeta.
    N_CHANNELS: constant Natural:= 3;

    -- Documentation:
    --     Identificador de la tarjeta de adquisición.
    type Video_Card is private;

    -- Documentation:
    --     Escalado del tamaño de las imagenes que se
```

```
--     capturaran en la tarjeta.
--     R_1 corresponde a la resolucion máxima
--     R_n corresponde a escalar la resolucion máxima
--     dividiendo por n
type Ratio is (R_1,R_2,R_4,R_8,R_16,R_32);

-- Documentation:
--     Pixel RGB en formato: R-G-B.
type PixelRGB is array(0..2)of C_Sauce.I_Environment.byte;

-- Documentation:
--     Píxels almacenados en las posiciones x,y.
type PixelData is array(natural range<>,netural range<>)of PixelRGB;

-- Documentation:
--     Canales de entrada de video de la tarjeta.
type Channels is new Integer range 0..N_CHANNELS-1;

-- Documentation:
--     Parametros de la imagen.
type Param is(Brigthness,Contrast,V_Sat,U_Sat,Hue);

-- Documentation:
--     Valores maximo , minimo y nominal (por defecto)
--     de un parametro.
type Param_Value is record
   MIN:integer;
   MAX:integer;
   NOMINAL:integer;
end record;

-------------------------------------------------------------------------
-- Variables de estado
-------------------------------------------------------------------------

-- Documentation
--     Valor del brillo.
Brigthness_Values: constant Param_Value:= (0,255,163);

-- Documentation
--     Valor del contraste.
Contrast_Values: constant Param_Value:= (0,511,256);

-- Documentation
--     Valor de la saturacion V.
V_Sat_Values: constant Param_Value:= (0,511,180);

-- Documentation
--     Valor de la saturacion U.
U_Sat_Values: constant Param_Value:= (0,511,254);

-- Documentation
--     Valor de la intensidad.
Hue_Values: constant Param_Value:= (0,255,128);

-------------------------------------------------------------------------
-- Operaciones
-------------------------------------------------------------------------

-- Documentation:
--     Inicializa la tarjeta y reserva la memoria necesaria para dos
--     buffers de almacenamiento: uno para imágenes del vídeo en vivo y
--     otro para capturade imágenes estáticas. También se inicializan los
```

```
--      valores del estado de losparametros de vídeo para cada uno de
--      los canales (que pueden ser de distintovalor en un momento dado).
--      Eleva la excepcion Start_Error si se produce un error.
function Open_Card(n:Natural;r:Ratio:=R_1) return Video_Card;

-- Documentation:
--      Libera los recursos asociados a la tarjeta.
--      Eleva la excepcion End_Error si se produce un error.
procedure Close_Card(C: in out Video_Card);

-- Documentation:
--      Comienza a mostrar el vídeo en vivo en la ventana hwnd.
--      La imagen aparece escalada con dimensiones x por y pixels.
--      La resolución máxima es RESOLUTION_X por RESOLUTION_Y
--      pixels.
--      Se eleva la excepción Passthru_Error si se produce un error
procedure Star_Video(C: in out Video_Card;
                     hwnd:C_Sauce.I_Environment.HWND;
                     x:natural;
                     y:natural);

-- Documentation:
--      Se detiene el video en vivo.
--      Se eleva la excepción Passthru_Error si se produce un error
procedure Stop_Video(C: in out Video_Card);

-- Documentation:
--      Muestra una imagen superpuesta sobre el video en vivo
--      bmp es el nombre del fichero donde se encuentra la imagen
--      x e y son las dimensiones de la imagen de video que se
--      desea solapar; han de ser iguales o menores que los valores
--      de los parametros de la funcion VideoON
--      col indica el color de los pixels de la imagen que
--      desaparecen para permitir observar el video en vivo
--      Si translucent=true , la imagen superpuesta es toda
--      ella translucida.
--      Se eleva la excepción Passthru_Error si se produce un error
procedure Overlays_ON(C: in out Video_Card;
                      bmp:in string;
                      x:in integer;
                      y:in integer;
                      col:in PixelRGB;
                      trans:in boolean);
-- Documentation:
--      Finaliza el overlay
--      Es necesario invocar la funcion con el video en vivo ya detenido
--      Se eleva la excepción Passthru_Error si se produce un error
procedure Overlays_OFF(C: in out Video_Card);


-- Documentation:
--      Captura una imagen y la almacena en el buffer de imagen
--      estática con la resolución máxima permitida.
--      Se requiere que cuando se invoque no esté el video en vivo.
--      Se eleva la excepcion Acquisition_Error si se produce un error
procedure Acquire(C: in out Video_Card);

-- Documentation:
--      Copia en el buffer de la imagen estática la última imagen
--      almacenada en el buffer de vídeo en vivo.
--      Se requiere que cuando se invoque el vídeo esté en vivo.
--      Se eleva la exception Acquisition_Error si se produce un error.
procedure Snap(C: in out Video_Card);
```

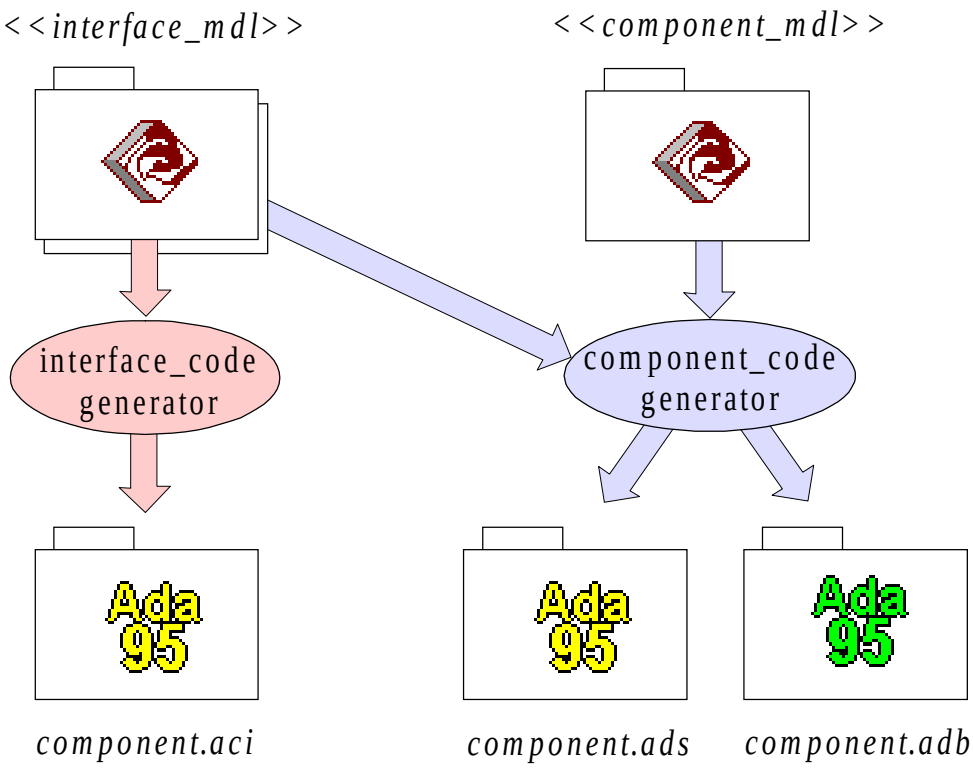
```
    -- Documentation:
    --     Transfiere a la memoria principal la imagen capturada
    --     anteriormente con Acquire o Snap.
    --     Transfiere el rectángulo determinado por los índices de data.
    --     Se eleva la excepción Read_Error si se produce un error.
    procedure Read(C: in out Video_Card;
                   data:in out PixelData);

    -- Documentation:
    --     Muestra en una ventana la imagen capturada en el buffer de
    --     la imagen estática. Es necesario haber capturado previamente
    --     una imagen con Acquire o Snap.
    --     Se eleva la excepción Draw_Error si se produce un error.
    procedure Draw_Image(C: in out Video_Card;
                         hwnd:C_Sauce.I_Environment.HWND);

    -- Documentation:
    --     Selecciona el canal de entrada de vídeo.
    --     Se eleva la interrupción Config_Error si se produce un error.
    procedure Set_Channel(C: in out Video_Card;
                          ch:Channels);

    -- Documentation:
    --     Modifica un parámetro de las imágenes capturadas o mostradas
    --     en vídeo en vivo en el futuro.
    --     Se eleva la excepción Config_Error si se produce un error.
    procedure Set_Parameter(C: in out Video_Card;
                          p:Param;
                          value:integer;
                          ch:Channels);

    -- Documentation:
    --     Devuelve el valor actual de un parámetro de las imágenes
    --     capturadas o mostradas en vídeo en vivo en el futuro.
    --     Se eleva la excepción Config_Error si se produce un error.
    function Get_Parameter(C:Video_Card;
                          p:Param;
                          ch:Channels)return integer;

    --=========================================================================
    -- Declaration of exported exceptions                                     =
    --=========================================================================

    Start_Error:exception;
    End_Error:exception;
    Draw_Error:exception;
    Read_Error:exception;
    Passthru_Error:exception;
    Acquisition_Error:exception;
    Config_Error:exception;

    --=========================================================================
    -- Private interface declaration                                          =
    --=========================================================================

    private
       type Video_Card is new Integer range 0..MAX_CARD-1;

    end I_Image_Grabber;
    --=========================================================================

--*****************************************************************************
```

```
-- Private component declaration                                              *
--*****************************************************************************

end;
```

II.4. Proceso de automatización de la gestión de componentes

La metodología de diseño basada en componentes que se propone en este trabajo se basa en el modelado UML de las interfaces y de los componentes. Esto requiere disponer de herramientas que ayuden a la generación y validación del código Ada que implementa estos componentes y de herramientas que generen el entorno UML que se requiere para el desarrollo de una aplicación. Estas herramientas no son desarrolladas en el presente proyecto, pero si que trataremos en este apartado de obtener las claves para su implementación en el futuro.

En la siguiente figura se muestran las dos herramientas básicas de generación de código Ada que se necesitan:



- Generador del código de interfaces: Una interfaz representa una especificación de servicio y un modelo de información autocontenido. La herramienta que se propone tiene dos objetivos:

  _ Verificar la consistencia y completitud del modelo UML que constituye la descripción de la interfaz.

_ Generar el código fuente Ada de la interfaz. Este código debe ser una especificación Ada compilable que pueda ser utilizada para compilar las aplicaciones de usuario que la utilicen.

- Generador del código de componentes: La herramienta integra la información contenida en el modelo del componentes y de las interfaces que implementa para generar el código compilable que constituye la implementación utilizable del componente. Esta herramienta lleva a cabo las siguientes tareas:

  _ Verifica la consistencia y complititud del modelo UML del componente.

  _ Verifica la existencia de los ficheros que contienen la descripción de las interfaces que el componente debe implementar, y comprueba la consistencia de la descripciones que contienen.

  _ Genera un fichero fuente Ada con la especificación completa y compilable del componente.

  _ Genera un fichero fuente Ada con el esqueleto del cuerpo del componente, conteniendo el encabezamiento del propio cuerpo del paquete y con la estructura de paquetes internos y de encabezamientos de los procedimientos y funciones que se correspondan con la especificación. Este fichero deberá ser completado por el programador con el código que requiera la implementación de las partes declarativas y los cuerpos de los procedimientos y funciones cuyas cabeceras se hayan incluido.

III. MODELO DE TIEMPO REAL DE LOS COMPONENTES

III.1. Modelo de tiempo real de una aplicación basada en componentes

El modelo de tiempo real de un componente contiene la información estructural y los datos necesarios para poder formular el modelo de tiempo real de cualquier aplicación que los incluya. El modelo de tiempo real de un componente aporta los modelos de los recursos hardware y software, los requerimientos de procesamiento y los artefactos de sincronización que se incorporan a la aplicación con el uso o acceso al componente.

El modelo de tiempo real de un componente solo aporta aquellos elementos que están incluidos en el componente, y por ello, no constituye en sí una instancia de modelo, sino un descriptor parametrizado que permitirá contruir el modelo de la aplicación que hace uso de él, cuando se definan los modelos de los otros componentes de los que hace uso el componente para ofrecer su funcionalidad y cuando se concreten los parámetros que establezcan la forma en que hace uso de él la aplicación.

El modelo de tiempo real de un componente, se describe como una clase UML de estereotipo <<component_descr>> que aporta:

- Identificador que debe ser invocado para instanciar un componente que corresponda al modelo.
- Descriptor de componente del que, en su caso, puede heredar sus características.
- Un dominio de nombres, que puede ser utilizado para referenciar los componentes internos que están declarados en el modelo y que se instancia con cada instancia del componente.
- Parámetros con sus tipos, y en su caso, valores por defecto de los parámetros que deben ser establecidos en la instanciación del componente, para especificar el uso que de él se hace.
- Referencias de los modelos de otros componentes, de los que depende la instancia del modelo del componente, y que deben ser resueltos en la instanciación del modelo del componente.
- Conjunto de modelos de las operaciones, en su caso, organizadas por interfaces, que pueden ser referenciadas por la aplicación o por los modelos de otros componentes, a fin de formular su propio modelo de tiempo real.

En este capítulo se muestra la estructura y la información asociada a dos componentes característicos de los desarrollados en este proyecto. En primer lugar, se describe el modelo del componente NT_Processor_Mdl_1 que representa la plataforma hardware/software sobre la que se ha desarrollado la aplicación. Este es un componente típico de un recurso hardware/ software, que modela una plataforma. En segundo lugar, se describe el componente C_IG_DT3153, que modela un componente hardware/software relativo a una tarjeta de captura, digitalización y almacenamiento de imagenes de vídeo, y al driver software de control de la misma.

En el próximo capítulo, se utilizarán instancias de los modelos de estos componentes, así como la de los otros dos componentes desarrollados a fin de modelar el análisis de planificabilidad y estimar el comportamiento de tiempo real del demostrador que se ha construido en este proyecto.

III.2. Descripción de la plataforma WINDOWS NT

Windows NT ha sido diseñado como un sistema operativo de propósito general (GPOS[1]). Sin embargo, al no ser un sistema operativo de tiempo real RTOS[2] puede presentar serias dificultades para el que se ve obligado a desarrollar un sistemas de tiempo real con él.

Para que un sistema operativo pueda considerarse RTOS[3] ha de tener entre otras las siguientes características [20]:

- Tiene que ser multitarea y expulsor.
- Ha de tener capacidad de priorizar las tareas.
- Tiene que implementar mecanismos de sincronización de tareas que tengan una temporización predecible.
- Ha de existir un sistema de herencia de prioridades que evite el problema de "inversión de prioridad". Este problema se puede evitar con un diseño correcto , pero en sistemas complejos esto puede ser bastante complicado.
- El comportamiento del RTOS ha de ser conocido y predecible.

Es interesante conocer diversos aspectos sobre el funcionamiento del sistema operativo Windows NT relativos a sistemas de tiempo real para comprobar si en nuestro caso puede ser utilizado como RTOS:

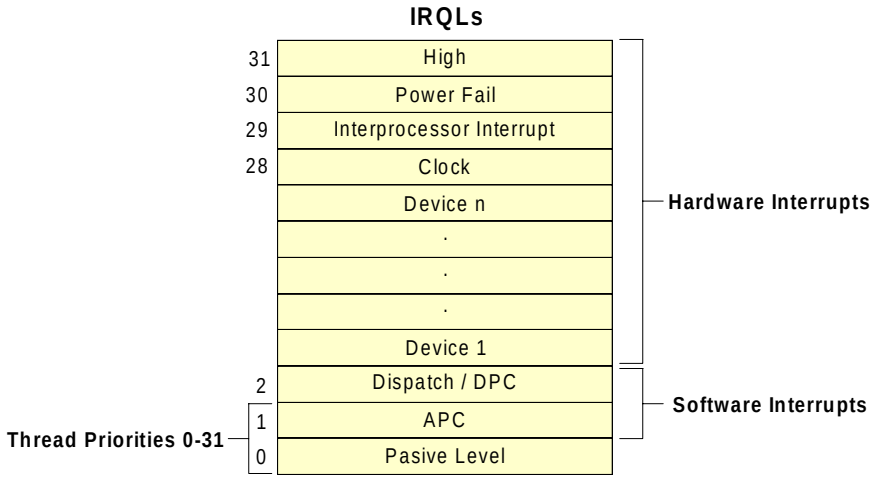
Las aplicaciones de tiempo real utilizan un sistema de interrupciones para asegurar que los eventos externos son detectados por el sistema operativo. Es importante que estas interrupciones sean tratadas oportunamente de acuerdo a su prioridad.

El kernel de Windows NT puede operar a uno de 32 niveles posibles de interrupción , tal como se puede ver en la próxima figura.

Cuando se produce una interrupción, inmediatamente se ejecuta una pequeña rutina de atención llamada ISR[4] que realiza únicamente las labores críticas (típicamente la escritura o lectura de registros hardware). Más adelante se completará la atención a la interrupción mediante la ejecución de una rutina DPC[5]. Este mecanismo es necesario ya que cuando se produce una interrupción de nivel n se enmascaran todas las demás interrupciones de nivel inferior hasta que se completa su ISR correspondiente.

1. General Purpose Operating System.
2. Real Time Operating System.
3. Real Time Operating System.
4. Interrupt Servicing Routine.
5. Deferred Procedure Call.

Los APC[1] permiten generar interrupciones software desde los programas de usuario. varias funciones de la API Win32 , como por ejemplo *ReadFileEx* o *WriteFileEx* , utilizan estos mecanismos.

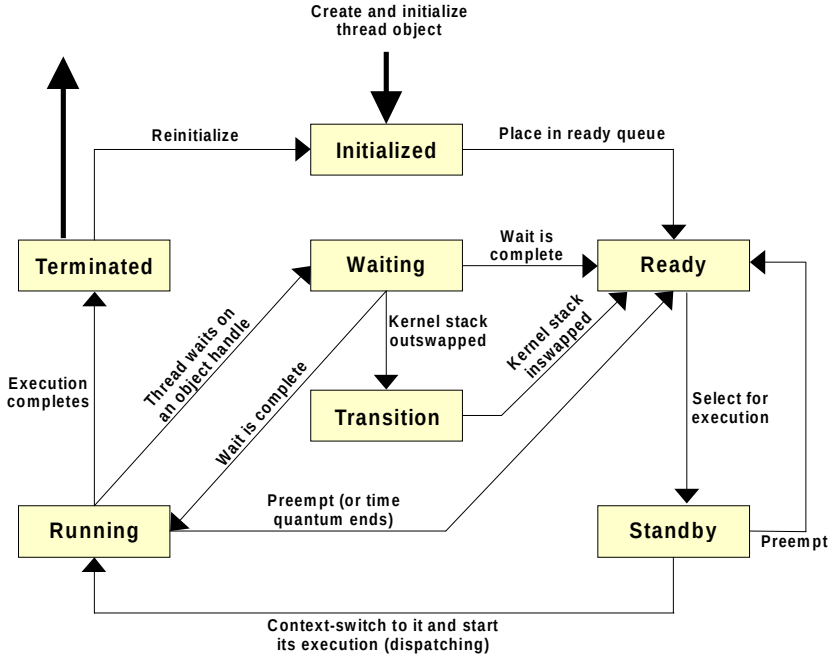


Durante el funcionamiento normal del sistema se ejecutan concurrentemente varios procesos. Cada uno de ellos se compone de uno o varios *threads*. Es el planificador (*dispatcher*) quien decide que *thread* se ejecuta en cada instante de tiempo. Los diferentes estados en que se puede encontrar un *thread* son los siguientes [21]:

- Ready: thread dispuesto para competir en la planificación.
- Standby: thread seleccionado por el planificador para ejecutarse a continuación.
- Running: thread en ejecución actualmente.
- Waiting: thread en espera de un recurso.
- Transition: thread que se puede considerar como ready pero que su stack aún no se ha cargado en memoria.

1. Asynchronous Procedure Call.

- Terminated: thread finalizado.

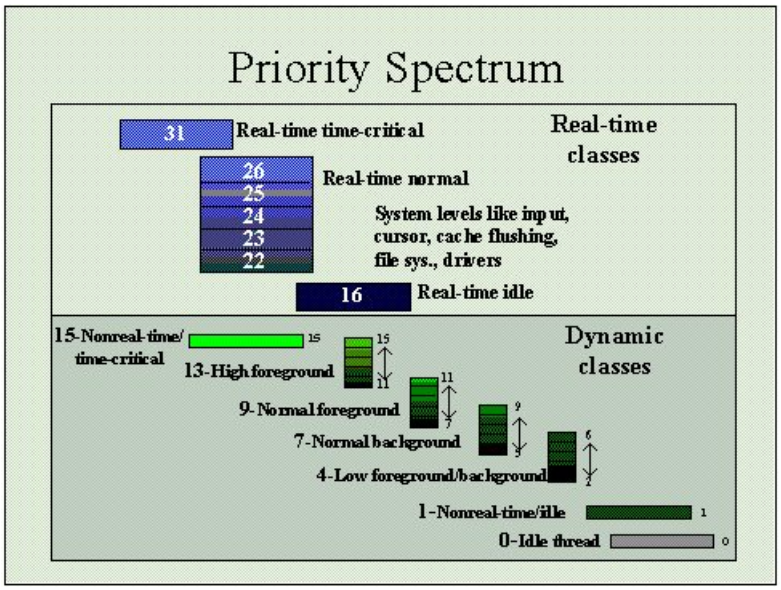


Cuando el planificador tiene que decidir que *thread* se ejecuta a continuación , lo primero que hace es ver si existe algún DPC pendiente. En caso afirmativo éste se ejecuta. En caso negativo se selecciona el *thread* de mayor prioridad. Los niveles de prioridad de los *threads* en Windows NT son 32 (0-31) , aunque en realidad no todos son utilizables. El programador de aplicaciones puede determinarlo fijando las prioridades del proceso (Win32 process priority class) y del *thread* (Win32 thread priority).

Win32 process priority classes

Win32 thread priorities	Real Time	High	Normal	Idle
Time critical	31	15	15	15
Highest	26	15	10	6
Above normal	25	14	9	5
Normal	24	13	8	4
Below normal	23	12	7	3
Lowest	22	11	6	2
Idle	16	1	1	1

Se distinguen dos tipos de prioridades: las dinámicas (0-15) y las de tiempo real (16-31).



En los *threads* que tienen asignada una prioridad dinámica, es posible que el sistema operativo cambie su prioridad en un momento dado (*boost priority*) para optimizar el rendimiento (por ejemplo al entrar en un estado wait , después de una I/O , en tareas que tras muchos ciclos aún no han conseguido el control de la CPU , ...). Todos los *threads* de procesos propios del sistema operativo tienen prioridades dentro de este rango.

En los *threads* que tienen asignada una prioridad de tiempo real , ésta no varia durante toda su ejecución (salvo que el programador lo especifique así en el código de la aplicación). Sólo pueden tener este rango de prioridades *threads* propios de aplicaciones de usuario, de modo que siempre serán más prioritarios que los procesos del sistema operativo.

III.3. Modelo de tiempo real de la plataforma WINDOWS NT

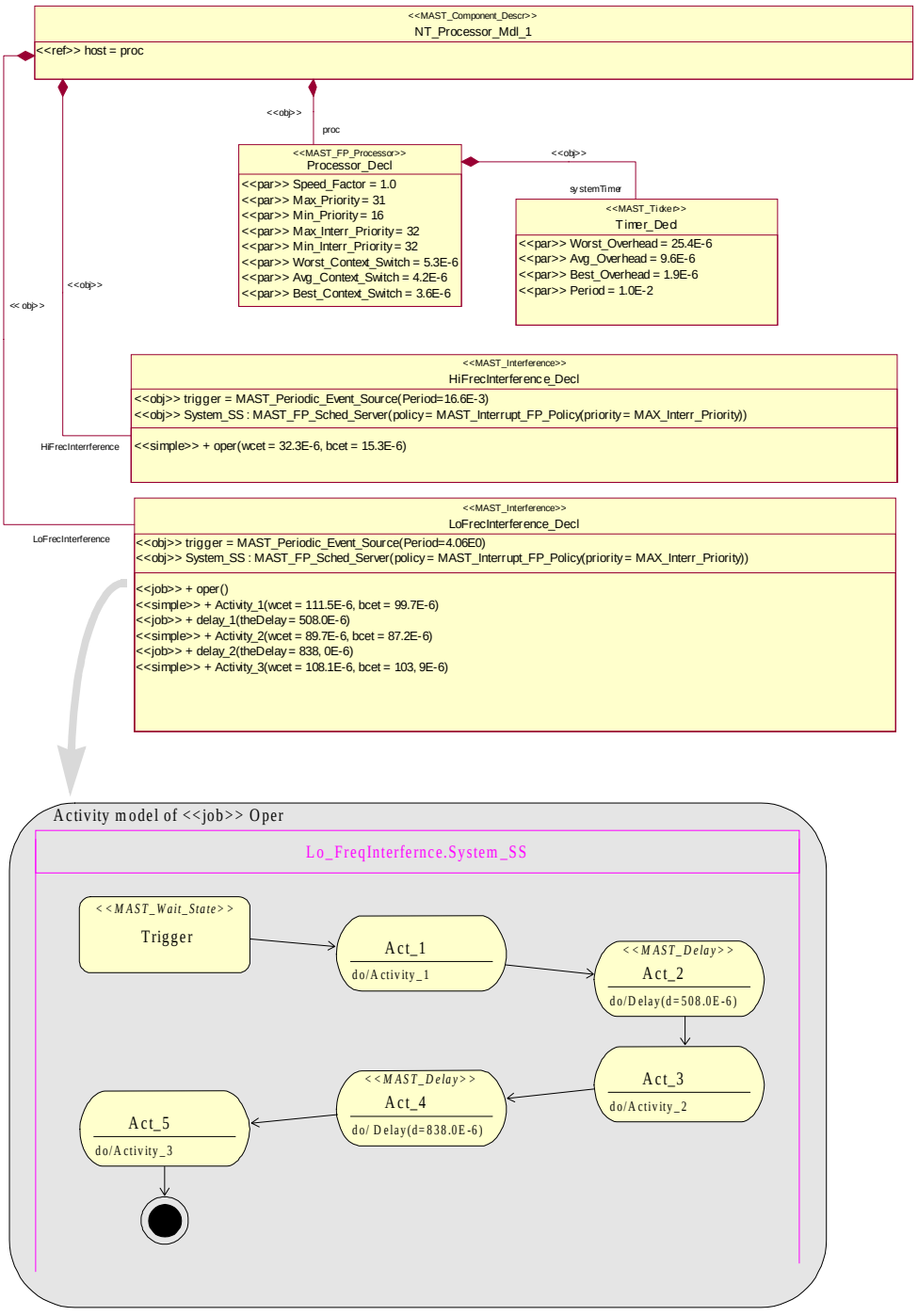
De acuerdo a lo expuesto en el apartado anterior, podemos contemplar la utilización del sistema operativo Windows NT para sistema de tiempo real bajo unas determinadas condicionees:

- Los *threads* de la aplicación tienen prioridades de tiempo real.
- No existe conexión a red de comunicación externa.
- Se dispone de suficiente memoria como para evitar el uso de memoria virtual (I/O a disco).
- Es la única aplicación que se está ejecutando en el computador.

En este trabajo se ha formulado el modelo de tiempo real de la plataforma Windows NT que utilizaremos posteriormente en la aplicación que se desarrolla. El modelo describe un computador concreto que opera bajo el sistema operativo Windows NT Server 4.0.

El componente NT_Processor_Mdl_1 es el modelo CBSE_MAST del procesador que se ha utilizado como plataforma para desarrollar la aplicación desarrollada como demostrador en este proyecto. Es un computador Pentium III a 800 MHz que opera con el sistema operativo Windows NT 4.1 Server. El modelo NT_Processor_Mdl_1 describe la potencia de procesado que suministra el procesador, así como la totalidad de las tareas de background que introduce el sistema operativo para gestión de los recursos que interfieren la ejecución de la aplicación por ejecutarse a prioridad superior de las que se ejecutan sus tareas.

En la siguiente figura se muestra el modelo de la plataforma utilizando el perfil CBSE_MAST [23]. El modelo se describe mediante el elemento NT_Processor_Mdl_1, que agrega internamente un conjunto de componentes subordinados. El modelo de la plataforma tiene el objeto de describir la capacidad de computación que se dispone para la ejecución de la aplicación, y esta resulta de considerar la que proporciona el procesador del computador, y de restarle la que se consume por las procesos de background que introduce el sistema operativo en la gestión de los diferentes recursos (timer, drivers, rutinas de recuperación, etc.). La potencia que proporciona el procesador se modela mediante un componente del tipo MAST_FP_ Processor, el consumo por gestión del timer se modela mediante un elemento del tipo MAST_ Timer, que en este caso es del tipo especialicado MAST_Ticker, y el consumo de las otras rutinas de gestión se modelan como un conjunto de componentes del tipo MAST_Interference.



El componente NT_Processor_Mdl_1 describe el modelo de tiempo real de la plataforma. Es un elemento contenedor que se usa para proporcionar el identificador con que se puede hacer referencia al descriptor del modelo completo. Tiene los siguientes elementos agregados:

- <<*ref*>>**host**: Es una referencia predefinida para cualquier componente y específica el modelo del procesador en que se instancia el componente. Aunque habitualmente es una referencia cuyo valor es establecido en la instanciación, en este caso se puede

resolver en el propio descriptor. Se le asigna el valor “proc” que es la referencia interna al procesador que también se declara en el propio componente.

- <<MAST_FP_Processor>> **proc**: Es el componente de modelado que describe la capacidad de procesamiento del procesador que ejecuta el código de la aplicación. La capacidad de cómputo que representa se emplea en la ejecución de las actividades de la aplicación que tiene asignadas y en la ejecución de las tareas de gestión, monitorización y cambio de contexto entre actividades. Opera bajo una estrategia de prioridad fija. Sus elementos agregados son:

  _ <<*par*>>**Speed_Factor**: es un atributo escalar que describe la capacidad de procesamiento del recurso (el tiempo físico de ejecución de cualquier actividad que se ejecute en el recurso se obtendrá dividiendo el tiempo normalizado de ejecución (WCET , ACET , BCET , ...) establecido en las operaciones que lleva a cabo , por el factor de velocidad). En este caso tiene asignado el valor 1.0. ya que este procesador es el que se ha utilizado para evaluar los tiempos de ejecución de las diferentes actividades.

  _ <<*par*>>**Max_Priority**: máximo nivel de prioridad para un proceso de aplicación que se ejecuta en el procesador. Se le asigna el valor 31 que corresponde a la prioridad mas alta para un proceso de aplicación de tiempo real en NT_Window.

  _ <<*par*>>**Min_Priority:** mínimo nivel de prioridad para un proceso de aplicación que se ejecuta en el procesador. Se le asigna el valor 16 que corresponde a la prioridad mas baja para un proceso de aplicación de tiempo real en NT_Window.

  _ <<*par*>>**Max_Interr_Priority**: máximo nivel de prioridad para una rutina de interrupción de aplicación que se ejecuta en el procesador.Se le asigna el valor 32, pero su valor es irrelevante salvo en que es superior a las prioridades de aplicación.

  _ <<*par*>>**Min_Interr_Priority**: mínimo nivel de prioridad para una rutina de interrupción de aplicación que se ejecuta en el procesador.Se le asigna el valor 32, pero su valor es irrelevante salvo en que es superior a las prioridades de aplicación.

  _ <<*par*>>**Worst_Context_Switch**: tiempo máximo de cómputo (peor caso) que el procesador emplea en realizar un cambio de contexto entre procesos de aplicación.

  _ <<*par*>>**Avg_Context_Switch**: tiempo promedio de cómputo que el procesador emplea en realizar un cambio de contexto entre procesos de aplicación.

  _ <<*par*>>**Best_Context_Switch**: tiempo mínimo de cómputo (mejor caso) que el procesador emplea en realizar un cambio de contexto entre procesos de aplicación.

  _ <<*par*>>Worst_ISR_Switch: tiempo máximo de cómputo (peor caso) que el procesador emplea en realizar un cambio de contexto a una interrupción. Se le asigna el valor 0.0 (default value)

_ <<*par*>>Avg_ISR_Switch: tiempo promedio de cómputo que el procesador emplea en realizar un cambio de contexto a una interrupción. Se le asigna el valor 0.0 (default value)

_ <<*par*>>Best_ISR_Switch: tiempo mínimo de cómputo (mejor caso) que el procesador emplea en realizar un cambio de contexto a una interrupción. Se le asigna el valor 0.0 (default value)

_ <<*obj*>>**SystemTimer**: Modelo de gestión del timer. Se describe en la siguiente sección.

- <<MAST_Ticker>> **proc.SystemTimer:** Modela la carga de procesamiento que supone para el procesador la gestión de un timer basado en interrupciones hardware periódicas. Así mismo modela la granularidad en la medida del tiempo por parte del sistema operativo que introduce el timer. Sus parámetros son:

  _ <<*par*>>**Worst_Overhead**: tiempo máximo de cómputo (peor caso) que el procesador emplea en gestionar la atención a los plazos de temporización pendientes. Se le asigna el valor 25.4E-6 segundo.

  _ <<*par*>>**Avg_Overhead**: tiempo promedio de cómputo que el procesador emplea en gestionar la atención a los plazos de temporización pendientes. Se le asigna el valor 9.6E-6 segundo.

  _ <<*par*>>**Best_Overhead**: tiempo mínimo de cómputo (mejor caso) que el procesador emplea en gestionar la atención a los plazos de temporización pendientes. Se le asigna el valor 1.9E-6 segundo.

  _ <<*par*>>***Period***: periodo de tiempo entre dos evaluaciones del ticker. Este tiempo representa la granularidad de apreciación del tiempo del procesador. Se le asigna el valor 1.0E-2 segundo.

En la siguiente sección III.4., se describe el método de evalución de estos parámetros.

- <<MAST_Interference>> : Modela la carga de procesamiento que supone para el procesador una interrupción periódica que se produce en el procesador y que es atendida por éste. El origen puede ser diverso (tarjetas hardware , software , ...). En el modelo de una plataforma Windows NT puede existir una, varias o ninguna MAST_Interference. Una MAST_Interference tiene agregados dos elementos internos:

  _ <<*obj*>>**Trigger**: Es un componente del tipo << Mast_Periodic_Event_Source>> que modela el patrón de disparo de la interferencia. En este caso,tiene un único parámetro interno <<*par*>>**period**.

  _ <<*obj*>>***System_SS***: Modela el proceso (<<MAST_FP_Sched_Server>>) en el que se planifica y ejecuta las actividades que se ejecutan. La política de planificación que se le ha asociado es del tipo <<MAST_Interrupt_FP_Policy>> y está caracterizada a su vez por el atributo <<*par*>>**priority**. En este caso, se le asigna el valor proc.Max_Interr_Priority definida en el procesador.

_ <<usage>> **Oper:** Operación que describe el código que se ejecuta en cada invocación de la interference. Puede ser simple (<<*simple*>>) o compuesta (<<*job*>>). En este último caso ha de estar descrita en su diagrama de actividad correspondiente.

III.4. Estimación de los valores de los parámetros del modelo de la plataforma

III.4.1. FP_Processor

El valor del *Speed_Factor* es una constante de proporcionalidad respecto a un sistema de referencia. En nuestro caso, como el sistema que modelamos es a su vez el que se utiliza como herramienta de estimación de tiempos de ejecución, *Speed_Factor* = 1.0.

Las prioridades de los threads de nuestras aplicaciones han de ser de tiempo real , de modo que *Max_Priority* = 31 y *Min_Priority* = 16. Hay que tener en cuenta que , a pesar de estos valores , sólo disponemos de 7 valores efectivos de prioridad: 16 , 22 , 23 , 24 , 25 , 26 y 31.

Cualquier interrupción es más prioritaria que cualquier *thread* de usuario. Además , no nos importa cual es más prioritaria entre ellas. Por ello , podemos fijar *Max_Interr_Priority* = *Min_Interr_Priority* = 32.

Los tiempos de cambio de contexto se estimaron mediante dos pruebas de medida:

1) Evaluación del tiempo de lectura del reloj de tiempo real.

Se ejecuta un programa que lee de forma sucesiva y en un gran número de veces el reloj del sistema. El pseudocódigo del programa que realiza esta medida, es:

```
Procedure Test_Lectura;
```

T_Atención_Mínimo = 1.9 μseg.

Para la interrupción 2:

Periodo = 16.3 mseg.

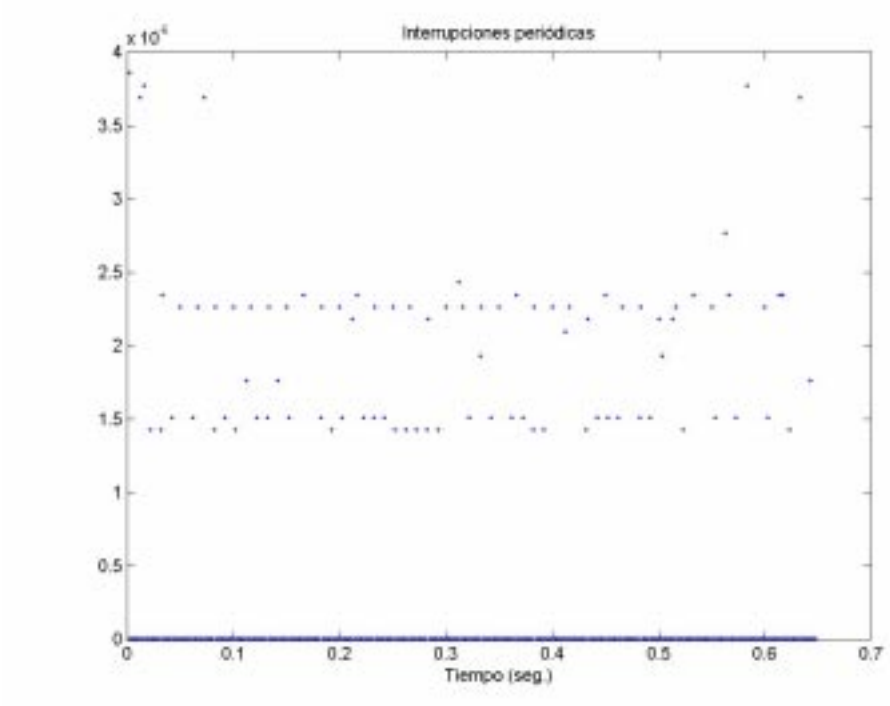
T_Atención_Máximo = 32.3 μseg.

T_Atención_Medio = 23.5 μseg.

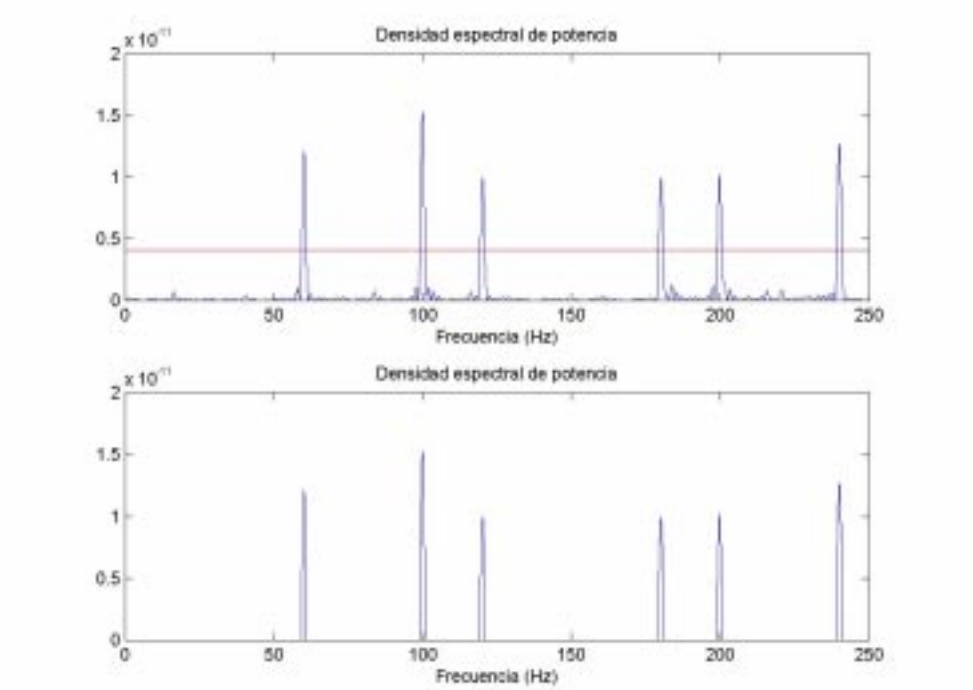
T_Atención_Mínimo = 15.3 μseg.

Estos valores se pueden obtener de forma automatizada implementando un algoritmo de análisis espectral. Se ha hecho utilizando la herramienta informática MATLAB 5.3. de la empresa MathWorks , Inc. Se han seguido los siguientes pasos:

- Se elimina el nivel de continua (TClock) de los datos y se normaliza el tiempo de muestreo (en nuestro caso a 1 mseg.).



- Se aplica la transformada rápida de Fourier y se obtienen las frecuencias fundamentales. En nuestro caso resultan ser 61.34 Hz (periodo de 16.3 mseg.) y 100 Hz (periodo de 10 mseg.).



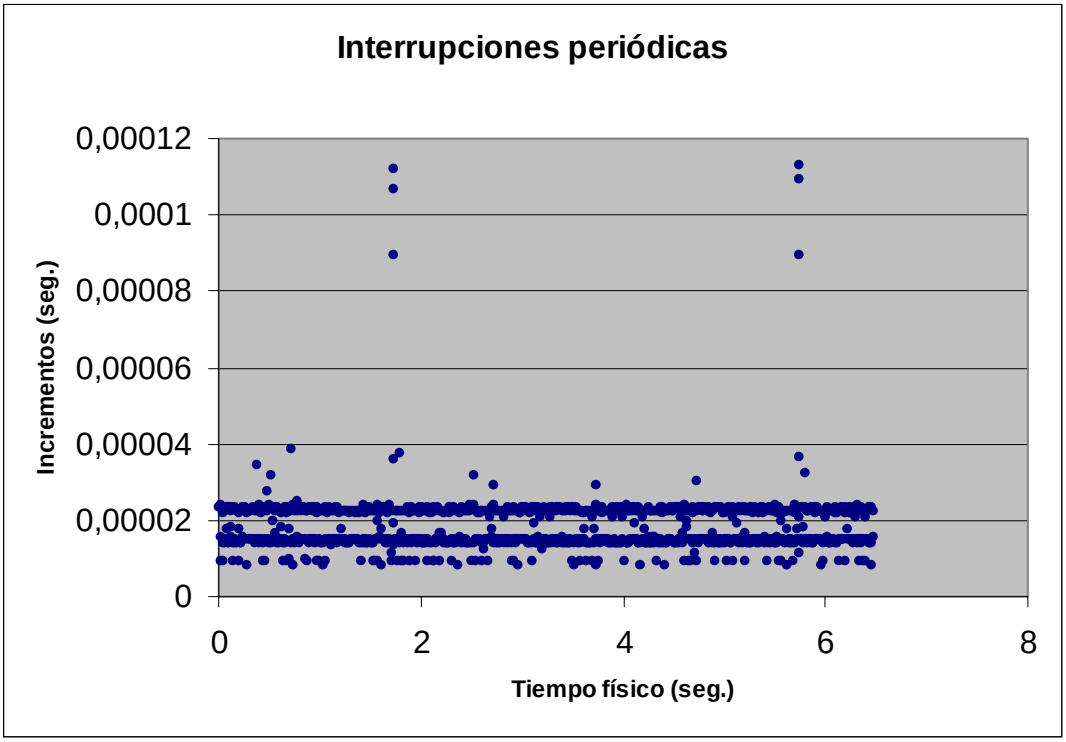
- Mediante un análisis de correlación se obtienen los índices de las muestras que pertenecen a cada interrupción. Se eliminan las que pertenecen a más de una serie y con el resto se realiza un análisis estadístico para obtener los tiempos máximo , medio y mínimo.

De todas las interrupciones simples detectadas hay que identificar una de ellas como el *Ticker*. Como ya se ha comentado anteriormente, el periodo de esta interrupción representa la granularidad de apreciación del tiempo del procesador. Por ello, necesitamos ejecutar el siguiente test:

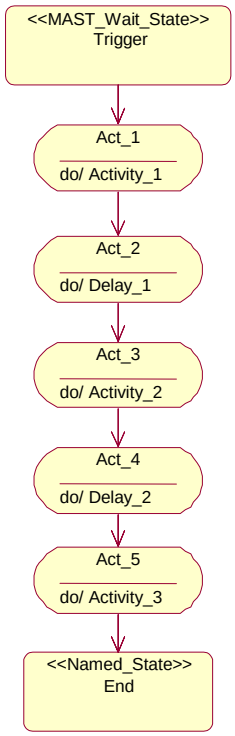
```
Procedure Test_Ticker;
      T:array (1..100000) of T_fisico;
      D:T_Fisico;
      cont:integer=1;
begin
      Establece prioridad a un nivel de tiempo real;
      for i in 1..1000 loop
            D=0.1*(1/i);
            for n in 1..100 loop
                  T[cont]=Lee_tiempo_fisico;
                  cont=cont+1
                  Sleep D;
            end loop;
      end loop;
      Guarda el array en un fichero para su analisis;
end;
```

Analizamos el array de resultados y obtenemos el valor mínimo de D para el cual T[i+1]-T[i] es aproximadamente igual a D. Ese valor es precisamente el periodo del *Ticker* , que en nuestro caso resulta ser de 10 mseg.

Además de las dos interrupciones simples aparece otra más compleja cada 4 segundos. Ésta consiste en tres interrupciones simples que se repiten siempre siguiendo un patrón fijo.



Si representamos como antes en una gráfica los incrementos de tiempo respecto a la muestra anterior frente al tiempo físico apreciamos las dos bandas de las interrupciones simples de alta frecuencia (en torno a 15 y 23 µseg. de incremento) con cierta dispersión y las tres interrupciones de la *interference* compuesta cada 4 segundos.



Ésta última se modela mediante un diagrama de actividad con los siguientes valores:

Activity_1:

wcet = 115.6 µseg.

bcet = 99.7 µseg.

Delay_1:

theDelay = 508 µseg.

Activity_2:

wcet = 89.7 µseg.

bcet = 87.2 µseg.

Delay_2:

theDelay = 838 µseg.

Activity_3:

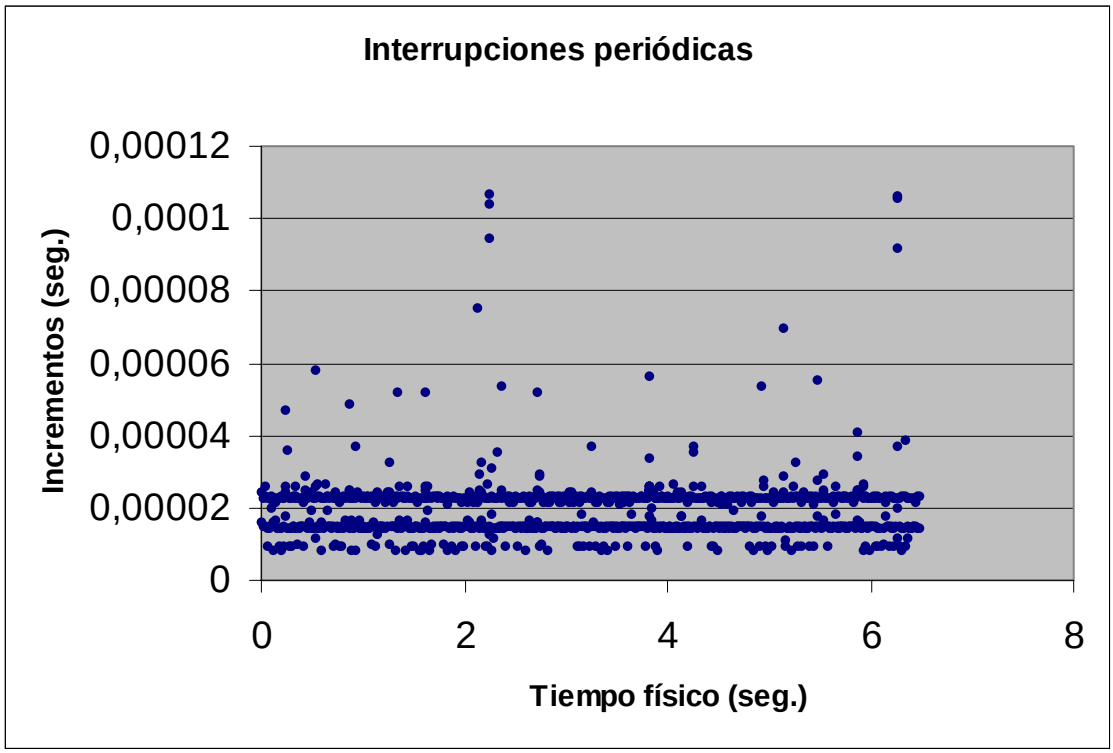
wcet = 108.1 µseg.

bcet = 103.9 µseg.

En esta ocasión la obtención automatizada de estos valores es más complicada , aunque en un futuro se pueden evaluar mecanismos que permitan llevarla a cabo.

El periodo exacto (en valor medio) de esta *interference* compuesta es de 4.06 segundos.

Por último, realizamos el mismo test pero en esta ocasión con la conexión a la red ethernet externa activa, simplemente para comprobar que en este caso el modelo no sería válido. El resultado es el siguiente:

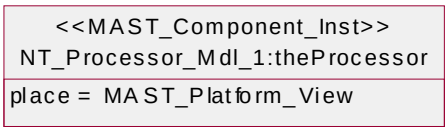


Ahora se aprecian nuevas interrupciones aparentemente esporádicas que no pertenecen a ninguna de las tres *interferences* modeladas anteriormente. Son interrupciones provocadas por

la red, y en principio no tenemos un modelo de ella, así que mantendremos la premisa de inabilitarla para implementar aplicaciones de tiempo real sobre esta plataforma.

III.5. Código MAST-File del componente NT_Processor_mdl_1

A efecto de documentar de forma completa la naturaleza y significado del modelo CBSE_Mast de la plataforma se listan las líneas de código que se insertan en el modelo Mast de la aplicación como consecuencia de la compilación del <<Mast_Component_Inst>> theProcessor que es una instancial del <<component>> NT_Processor_mdl_1.



De acuerdo con el modelo CBSE_Mast del procesador NT que se ha descrito en la sección III.3., los elementos que el modelo introducen en el MAST_file son:

- <<Mast_FP_Processor>> **theProcessor.proc**: Que modela el procesador que incluye el componente. A su vez este procesador incluye el modelo del timer:

  _ <<Mast_Ticker>> theProcessor.sysTimer: que modela tanto el overhead que requiere la atención del reloj interno del procesador, como la granularidad temporal que introduce en la medida del tiempo dentro del procesador.

- <<Mast_Interference>> **theProcessor.HiFrecInterference**: que model el overhead que introduce en el procesador una interrupción con periodo 16.3 ms (de origen y con finalidad desconocida). La compilación de esta Interference requiere introcuir en el código:

  _ <<Mast_FP_Sched_Server>> theProcessor.HiFrecInterference.System_SS: que modela el scheduling server que con una política de planificación MAST_Interrup_FP_Policy y con prioridad 32.

  _ <<Mast_Operation>>theProcessor.HiFrecInterference.Oper: que modela la actividad que se ejecuta en cada invocación de la interferencia.

  _ <<MAST_Regular_Transaction>> theProcessor.HiFrecInterference: Que modela la ejecución periodica de la operación previa. Esta transacción incluye el patrón de generación del evento externo theProccessor.HiFrecInterference.Trigger que dispara la interferencia.

- <<Mast_Interference>> **theProcessor.LoFrecInterference**: que modela el overhead que introduce en el procesador una interrupción con periodo 4s (de origen y con finalidad desconocida). Esta interferencia es más compleja ya que incluye la ejecución a nivel de sistema de tres actividades separadas entre sí por tiempos de retraso constantes. La compilación de esta Interference requiere introducir en el código:

_ <<Mast_FP_Sched_Server>>theProcessor.LoFrecInterference.System_SS: que modela el scheduling server que con una política de planificación MAST_Interrup_FP_Policy y con prioridad 32.

_ <<Mast_Simple_Operation>>theProcessor.LoFrecInterference.Activity_1

_ <<Mast_Simple_Operation>>theProcessor.LoFrecInterference.Activity_2

_ <<Mast_Simple_Operation>>theProcessor.LoFrecInterference.Activity_3,

que modelan las actividades que se ejecuta en cada invocación de la interferencia.

_ <<MAST_Regular_Transaction>>theProcessor.LoFrecInterference: Que modela la ejecución periodica de la secuencias de las operaciones previas con los correpondientes retrasos entre ellas. Esta transacción incluye el patrón de generación del evento externo theProccessor.LoFrecInterference.Trigger que dispara la interferencia.

```
--*****************************************************************************
-- Segmento del código MAST-File del modelo RT_Comp_Railway
--
-------------------------------------------------------------------------------
-- theprocessor (: NT_Processor_Mdl_1 <<MAST_Component_Inst>>)
-------------------------------------------------
-- place = MAST_Platform_View (Buscamos ahi el modelo)
  -- NT_Processor_Mdl_1 (<< MAST_COmponent_Descr>>)
  -- referencias
     -- host = proc
  -- objetos:
-------------------------------------------------------------------------------
     -- theprocessor.proc (: MAST_FP_Processor)
  ---------------------------------------------
  -- MAST_FP_Processor
  ---------------------------------------------
  -- Se corresponde a un Fixed_Priority_Processor con todos los atributos que se
  -- pasan como parametros
Processing_Resource
  (Type                    => Fixed_Priority_Processor,
   Name                    => theProcessor.proc,
   Max_Priority            => 31,
   Min_Priority            => 16,
   Max_Interrupt_Priority  => 32,
   Min_Interrupt_Priority  => 32,
   Worst_Context_Switch    => 5.3E-6,
   Avg_Context_Switch      => 4.2E-6,
   Best_Context_Switch     => 3.6E-6,
   System_Timer            =>
     (Type                    => Ticker,
      Worst_Overhead          => 25.4E-6,
      Avg_Overhead            => 9.6E-6,
      Best_Overhead           => 1.9E-6,
      Period                  => 1.0E-2
     ),
   Speed_Factor            => 1.0
  );
-------------------------------------------------------------------------------
    -- theprocessor.HiFrecInterference (<<MAST_Interference>>)
  -------------------------------------------
```

```
  -- MAST_Interference
  -------------------------------------
  -- Declaramos una transaction que tiene:
  --   - External Event Periodico con periodo definido por trigger.Period
  --   - Operation Oper con los WCET,ACET Y BCET que se especifican
  --   - Scheduling server System_SS del tipo MAST_FP_Sched_Server
    ------------------------------
    -- MAST_FP_Sched_Server
    ------------------------------
    -- Declaramos un scheduling server con la policy y priority indicadas
Scheduling_Server
  (Type                      => Fixed_Priority,
   Name                      => theProcessor.HiFrecInterference.System_SS,
   Server_Sched_Parameters   =>
     (Type                      => Interrupt_FP_Policy,
      The_Priority              => 32
     ),
   Server_Processing_Resource => theProcessor.proc
  );
     -- se declara la operacion que ejecuta la interferecia
Operation
  (Type                      => Simple,
   Name                      => theProcessor.HiFrecInterference.Oper,
   Worst_Case_Execution_Time  => 32.3E-6,
   Best_Case_Execution_Time   => 15.3E-6
  );
Transaction
  (Type                      => Regular,
   Name                      => theProccessor.HiFrecInterference,
   External_Events           =>
    ((Type                      => Periodic,
      Name                      => theProccessor.HiFrecInterference.Trigger,
      Period                    => 16.6E-3
    )),
   Internal_Events           =>
    ((Type                      => Regular,
      Name                      => theProcessor.HiFrecInterference.End
    )),
   Event_Handlers            =>
    ((Type                      => Activity,
      Input_Event               => theProccessor.HiFrecInterference.Trigger,
      Output_Event              => theProcessor.HiFrecInterference.End,
      activity_operation        => theProcessor.HiFrecInterference.Oper,
      activity_server           => theProcessor.HiFrecInterference.System_SS
    ))
  );
-----------------------------------------------------------------------
     -- theprocessor.LoFREcInterference  (<<MAST_Interference>>)
  ------------------------------------------
  -- MAST_Interference
  -------------------------------------
  -- Declaramos una transaction que tiene
  -- External Event Periodico con periodo definido por trigger.Period
  -- Operation Oper con los WCET,ACET Y BCET que se especifican
  -- Scheduling server System_SS del tipo MAST_FP_Sched_Server
    ------------------------------
    -- MAST_FP_Sched_Server
    ------------------------------
    -- Declaramos un scheduling server con la policy y priority definidas
Scheduling_Server
  (Type                       => Fixed_Priority,
   Name                       => theProcessor.LoFrecInterference.System_SS,
   Server_Sched_Parameters    =>
```

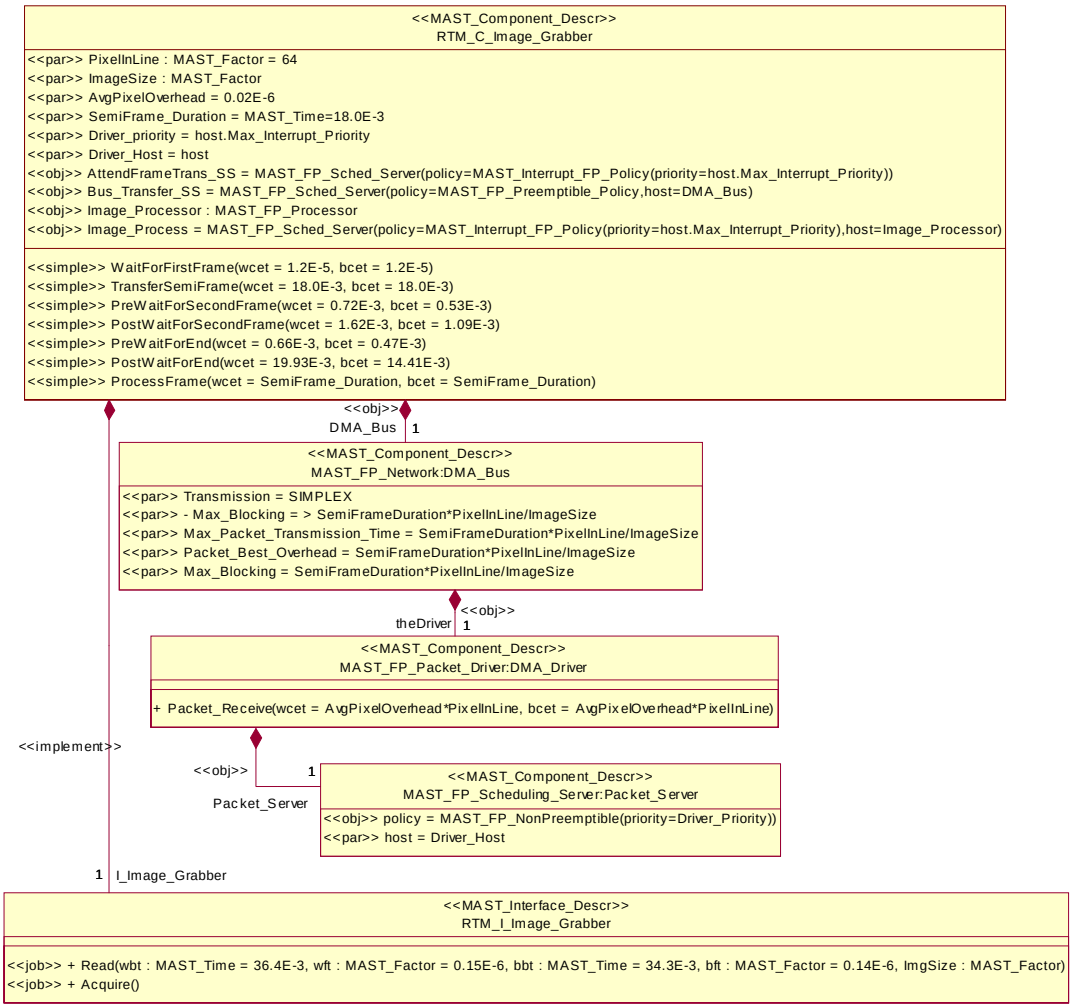
```
    (Type                          => Interrupt_FP_Policy,
     The_Priority                  => 32
    ),
   Server_Processing_Resource   => theProcessor.proc
  );
-- La transacción de la interferencia activa un job compuesto de tres operaciones
-- simples y dos delays:
    --  # oper.Act_1 = Activity_1
    --  # oper.Act_2 = Delay_1 ; que a su vez = delay(theDelay = 508.0E-6)
    --  # oper.Act_3 = Activity_2
    --  # oper.Act_4 = Delay_2; que a su vez = delay (838.0E-6)
    --  # oper.Act_5 = Activity_3
Operation
  (Type                         => Simple,
   Name                         => theProcessor.LoFrecInterference.Activity_1,
   Worst_Case_Execution_Time    => 111.5E-6,
   Best_Case_Execution_Time     => 99.7E-6
  );
Operation
 (Type                          => Simple,
  Name                          => theProcessor.LoFrecInterference.Activity_2,
  Worst_Case_Execution_Time    => 89.7E-6,
  Best_Case_Execution_Time     => 87.2E-6
 );
Operation
 (Type                          => Simple,
  Name                         => theProcessor.LoFrecInterference.Activity_3,
  Worst_Case_Execution_Time     => 108.1E-6,
  Best_Case_Execution_Time => 103.9E-6
 );
Transaction
 (Type                          => Regular,
  Name                          => theProccessor.LoFrecInterference,
  External_Events               =>
   ((Type                          => Periodic,
     Name                          => theProccessor.LoFrecInterference.Trigger,
     Period                        => 4.06E0)),
  Internal_Events               =>
   ((Type                          => Regular,
     Name                          => LFI1
    ),
    (Type                          => Regular,
     Name                          => LFI2
    ),
    (Type                          => Regular,
     Name                          => LFI3
    ),
    (Type                          => Regular,
     Name                          => LFI4
    ),
    (Type                          => Regular,
     Name                          => theProcessor.LoFrecInterference.End
   )),
  Event_Handlers                =>
   ((Type                          => Activity,
     Input_Event                   => theProccessor.LoFrecInterference.Trigger,
     Output_Event                  => LFI1,
     Activity_operation            => theProcessor.LoFrecInterference.Activity_1,
     Activity_server               => theProcessor.LoFrecInterference.System_SS
    ),
    (Type                           => Delay,
     Input_Event                   => LFI1,
     Output_Event                  => LFI2,
```

```
     Delay_Max_Interval           => 508.0E-6,
     Delay_Min_Interval           => 508.0E-6
    ),
    (Type                         => Activity,
     Input_Event                  => LFI2,
     Output_Event                 => LFI3,
     activity_operation           => theProcessor.LoFrecInterference.Activity_2,
     activity_server              => theProcessor.LoFrecInterference.System_SS
    ),
    (Type                         => Delay,
     Input_Event                  => LFI3,
     Output_Event                 => LFI4,
     Delay_Max_Interval           => 838.0E-6,
     Delay_Min_Interval           => 838.0E-6
    ),
    (Type                         => Activity,
     Input_Event                  => LFI4,
     Output_Event                 => theProcessor.LoFrecInterference.End,
     Activity_operation           => theProcessor.LoFrecInterference.Activity_3,
     Activity_server              => theProcessor.LoFrecInterference.System_SS
   ))
 );
--*****************************************************************************
```

III.6. Modelo de tiempo real del componente C_IG_DT3153

A modo de ejemplo, en este apartado se describir el modelo de tiempo real de uno de los componentes software implementados, concretamente describe el componente C_IG_DT3153, que posteriormente utilizaremos en la aplicación de ejemplo para detectar la posición de los trenes en la maqueta.

.



Como se ha descrito en el capítulo II este componente ofrece una interfaz que posibilita capturar una imagen de una cámara de vídeo, transferirla al espacio de memoria de la aplicación o transferirla a una ventana de vídeo en vivo de windows. En este apartado solo se describen los modelos de tiempo real de las operaciones Read y Adquire, que son la que se utilizan en el demostrador.

El componente C_IG_DT3153, modela el componente software diseñado para controlar la tarjeta de digitalización de imágenes (grabber) DT3153 de Data Translation, construida a partir del propio driver que suministra el fabricante y que se ha encapsulado como una DDL a fin de poder ser utilizada desde el código Ada que implementa el componente. Es un software de modelo de tiempo real complejo, ya que incorpora, una tarjeta hardware que realiza de manera autónoma la digitalización de la imagen, y de un canal DMA que transfiere la imagen desde la tarjeta a la memoria del computador en concurrencia con la ejecución de la aplicación.

El elemento principal del modelo es el *MAST_Component_Descr* [23]. Se trata de un ente de modelado que constituye un descriptor genérico y parametrizable que describe en este caso el recurso software del sistema. En el se declaran los recursos (componentes) internos o externos

que requiere para su operación, los párametros que deben establecerse en su intanciación y los modelos de los modos de uso que se le pueden requerir:

Parámetros:

- **<<par>> PixelInLine: Mast_Factor= 64:** Representa el número de píxels que son transferidos en una transacción por el DMA. Es un parámetro de modelado que no corresponde al sistema físico. Se ha introducido para incrementar la granularidad de la transferencia de información por el DMA.
- **<<par>> AvgPixelOverhead: MAST_Time=0.02E-6:** Representa el overhead medio que supone para el procesador la transferencia de un píxel a través del DMA.
- **<<par>> SemiFrame_Duration: Time=18.0E-3:** Representa el tiempo que tarda en transmitirse un semiframe en una señal de vídeo.
- **<<par>> Driver_Priority: MAST_Priority= Host.Max_Interrupt_Priority:** Prioridad en la que se ejecuta el driver control de la tarjeta de captura de imágenes.Se ha establecido como valor por defecto la maxima prioridad de interrupción del computador que ejecuta el driver.
- **<<par>> DriverHost: MAST_FP_Processor= Host:** Establece el computador en que se ejecuta el driver. Por defecto se establece el mismo procesador en que se ha instalado el componente.

Referencias a componentes externos:

- **<<par>>host: NT_Processor_mdl_1**: Referencia al UML *MAST_Processing_Resource* que modela el Processing Resource en el que se ejecuta el componente. Debe ser establecido en la instanciación del componente.

Declaración de los componentes internos agregados:

- **<<obj>> AttendFrameTrans_SS:MAST_FP_Sched_Server:** Tarea del procesador que ejecuta el componente en la que se ejecuta la actividad de background de gestión del proceso de transferencia por DMA de la imágenes desde la tarjeta de digitalización. Esta tarea se planifica con política MAST_Interrupt_Priority_Policy y con la prioridad máxima de interrupción.
- **<<obj>> Bus_Transfer_SS: MAST_FP_Sched_Server:** Scheduling server del network auxiliar DMA_Bus para modelar el overhead que la transferencia de la imagen por DMA introduce sobre el procesador que executa el componente. Este scheduling server se planifica con política MAST_FP_Preemptible_Policy y con la prioridad por defecto.
- **<<obj>> Image_Processor: MAST_FT_Processor:** Procesador que modela las actividades de captura y digitalización de las imágenes en el hardware de la tarjeta de digitalización. Su unica función es modelar la temporización del proceso de camptura de imágenes de la señal de vídeo.
- **<<obj>> Image_Process: MAST_FT_Sched_Server:** Scheduling server en el que se ejecutan las actividades del Image_Processor.
- **<<obj>>DMA_Bus: MAST_FP_Packet_Driver:** Network de tipo MAST_FP_Network, que opera en modo Simplex, y que se utiliza como forma de modelar el overhead que produce el DMA sobre el procesador que ejecuta el

componente. Cuando se transmite una imagen, transfiere un número de paquetes igual al número de líneas (Pixel en imagen/Pixel en linea) y la duracción de la transmisión de cada paquete es tal que la transmisión de cada Semiframe dure el tiempo estimado. Agregado a este componente se definen:

_ **<<obj>>theDriver: MAST_FP_Packet_Driver:** Driver que introduce por cada paquete transmitido el correspondiente overhead en el procesador que ejecuta el componente.

_ **<<obj>>Packet_Server:MAST_FP_Sched_Server**: Scheduling server del procesador en que se ejecuta el componente en el que ejecuta la actividad que modela el overhead que introduce la transferencia por el DMA.

Modelo de tiempos reales de funciones que ofrece su interfaz:

- <<Simple>>WaitForFirstFrame
- <<Simple>>TransferSemiFrame
- <<Simple>>PreWaitForSecondFrame
- <<Simple>>PostWaitForSecondFrame
- <<Simple>>PreWaitForEnd
- <<Simple>>PostWaitForEnd
- <<Simple>>ProcessFrame

Conjunto de operaciones auxiliares utilizadas para el modelo de la operación externa Acquire, que se describe más adelante.

<<MAST_Interface_Descr>>I_Image_Grabber: Interfaz pública del componente que contiene las operaciones públicas que ofrece el componente.

<<job>> **Read:** Modela el procedimiento de transferencia a la memoria principal de la imagen capturada anteriormente con los procedimientos Acquire o Snap. Se trata de una operación sencilla que lo único que hace es copiar los píxels de la imagen capturada por la tarjeta en una estructura de datos accesible por el usuario. Es una operación totalmente software, de modo que lo único que necesitamos conocer son los tiempos de mejor y peor caso. En este caso estos tiempos dependen del tamaño de la imagen.

Se ha hecho un estudio de la variabilidad de los tiempos en función del número de píxels de la imagen. Se ha comprobado que el tiempo depende del número de pixels de forma lineal siguiendo el siguiente patrón:

Tiempo = OFFSET + (numero_pixels * CTE_LIN)

Realizando varias medidas para imágenes de distintos tamaños se han obtenido los valores de OFFSET y CTE_LIN (para el mejor y el peor tiempo). En el modelo de tiempo real describimos este comportamiento del siguiente modo:

En el *MAST_Interface_Descr* declaramos la operación como:

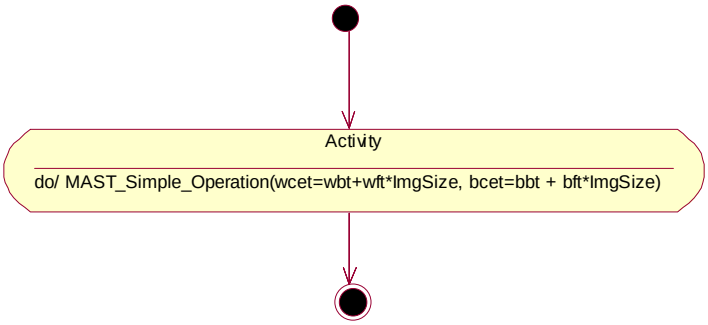
```
<<job>> Read(wbt : MAST_Time = 36.4E-3, wft : MAST_FActor = 0.15E-6, bbt :
```

```
                    MAST_Time = 34.3E-3, bft : MAST_Factor = 0.14E-6, ImgSize :
                    MAST_Factor)
```

Sus parámetros son:

- *wbt* : OFFSET para el cálculo del wcet.
- *wft* : CTE_LIN para el cálculo del wcet.
- *bbt* : OFFSET para el cálculo del bcet.
- *bft* : CTE_LIN para el cálculo del bcet.
- *ImgSize* : número de píxels de la imagen.

Y por lo tanto , el cálculo de los tiempos wcet y bcet se describe mediante el diagrama de actividad asociado a la operación:



Los valores de los parámetros de la operación se han obtenido realizando medidas sobre la plataforma que va a soportar la aplicación. Es posible que estos valores varien si la plataforma fuese otra distinta. Por ello, parece aconsejable que en un futuro se ofrezcan procedimientos para recalcular automáticamente estos valores al cambiar de plataforma.

<<job>> **Acquire:** Se trata de una operación compleja que cuando se invoca captura una imagen y la almacena en la memoria física reservada por la tarjeta. Esta memoria no es accesible por el usuario, por eso disponemos de la operación Read anteriormente comentada.

La tarjeta que utilizamos para la implementación del componente[1] realiza esta función utilizando DMA[2] , lo que provoca unas interrupciones y bajadas de rendimiento del procesador del PC que han de ser descritas en el modelo.

Para comprender su funcionamiento realizamos el siguiente test:

```
Procedure Test_Acquire;

     T:array (1..100000) of T_fisico;
     m:integer=1;

     Tarea_1 is
     Establece prioridad = 25;
```

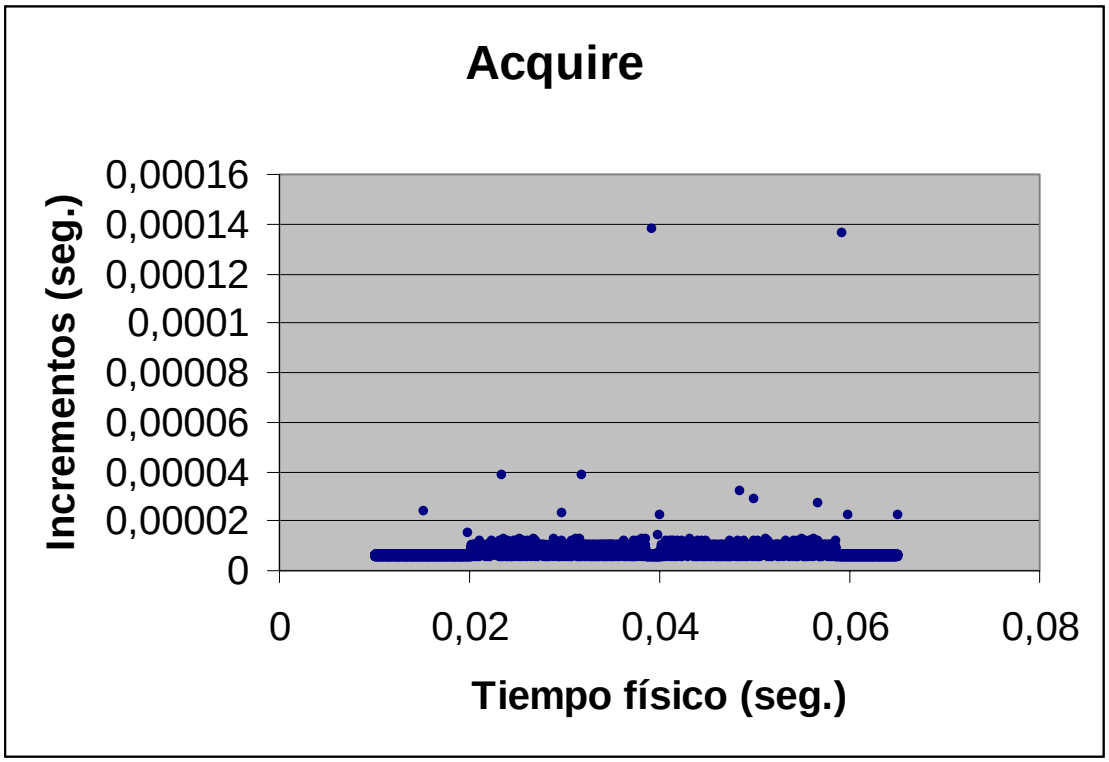
1. DT3153 de Data Translation.
2. Acceso Directo a Memoria.

```
      accept t1;
      entry t2;
      Sleep 0.01; -- para que comienze el acquire de Tarea_2
      for i in 1..100000 loop
           T[i]=Lee_tiempo_fisico;
      end loop;
      entry t2;
      end Tarea_1;

      Tarea_2 is
      Establece prioridad = 24;
      accept t2;
      Acquire;
      accept t2;
      Guarda el array en un fichero para su analisis;
      end Tarea_2;

begin
      Establece prioridad a un nivel de tiempo real;
      entry t1;
end;
```

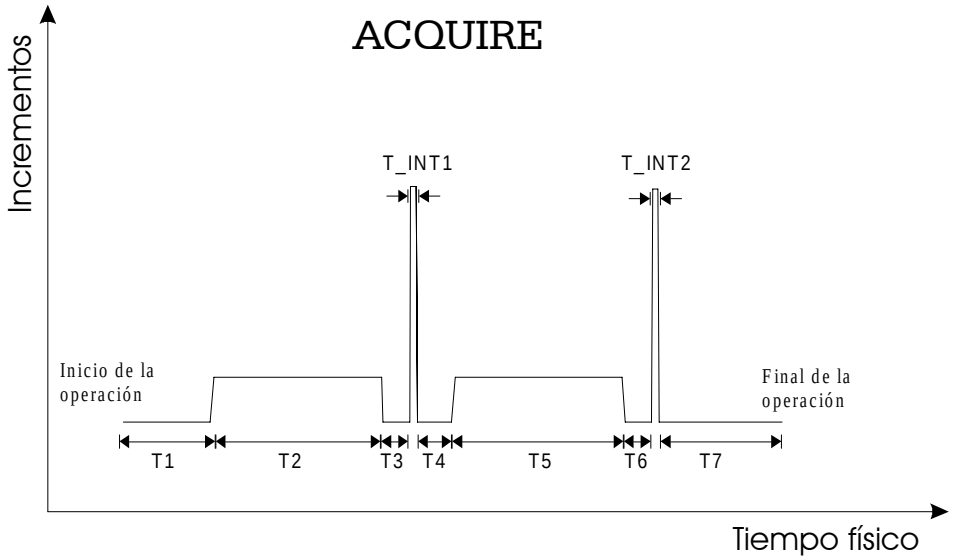
Si la operación Acquire fuese convencional en el array de resultados tendríamos muestras separadas TClock entre sí (salvo las *interferences* periódicas detectadas en la plataforma) , ya que la Tarea_1 es más prioritaria que Tarea_2. Pero en la práctica no es así.

Si representamos en una gráfica los incrementos de tiempo respecto a la muestra anterior frente al tiempo físico tenemos

:

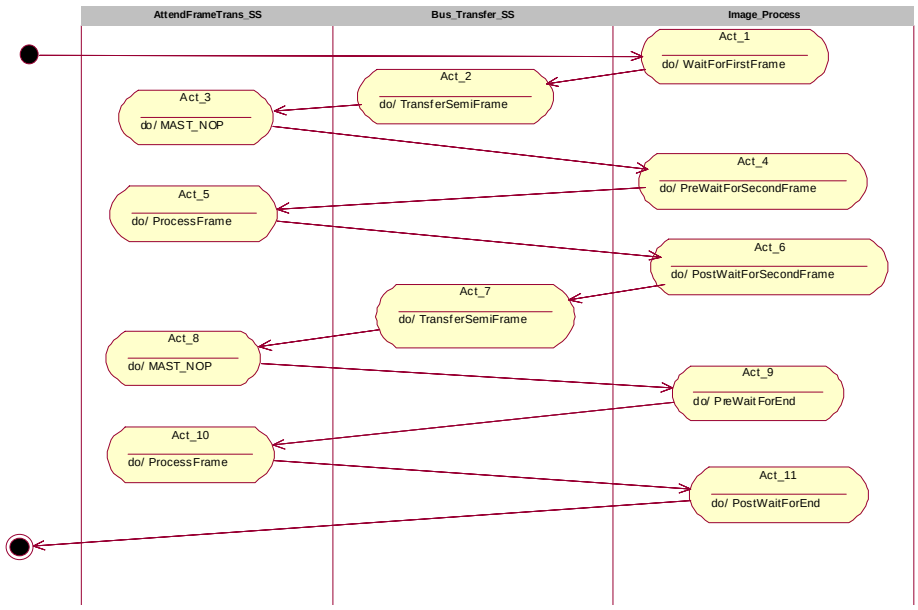


Si representamos de forma esquemática este resultado podemos identificar varias zonas:



- T1 : tiempo transcurrido desde el comienzo de la operación hasta que comienza a transmitirse la imagen. Este tiempo depende de la referencia de sincronismo de la señal analógica de vídeo. En este intervalo las tareas de orden superior se ejecutan normalmente.
- T2 : transmisión de las líneas pares de la imagen. Este tiempo depende del estándar de la señal analógica de vídeo. Durante la transmisión se produce una bajada de rendimiento en el procesador principal debido a que se utiliza DMA , y cuando el procesador de la tarjeta de adquisición utiliza el bus del sistema , el procesador principal tiene que esperar (si desea utilizarlo él).
- T3 : tiempo transcurrido desde el final de la transmisión de las líneas pares hasta la interrupción 1. En este intervalo las tareas de orden superior se ejecutan normalmente.
- T_INT1 : interrupción provocada por el procesador de la tarjeta para indicar que se ha completado la transmisión de las líneas pares.
- T4 : tiempo transcurrido desde la interrupción 1 hasta el comienzo de la transmisión de las líneas impares. En este intervalo las tareas de orden superior se ejecutan normalmente.
- T5 : como T2 pero para las líneas impares.
- T6 : como T3 pero para las líneas impares.
- T_INT2 : interrupción provocada por el procesador de la tarjeta para indicar que se ha completado la transmisión de las líneas impares.
- T7 : tiempo transcurrido desde la interrupción 2 hasta que la operación devuelve el control al flujo de la tarea desde la que se invocó. En este intervalo las tareas de orden superior se ejecutan normalmente.

Describimos este comportamiento con el siguiente diagrama de actividad[1]:



Donde *AttendFrameTrans_SS* representa al procesador principal , *Bus_Transfer_SS* representa al bus del sistema e *Image_Process* representa al procesador de la tarjeta de adquisición de la imagen.

Se han obtenido los parámetros de las distintas operaciones que resultan ser:

- *WaitForFirstFrame* (wcet = 26.68E-3, bcet = 15.55E-3) [T1]
- *TransferSemiFrame* (wcet = 18E-3, bcet = 18E-3) [T2 , T5]
- *PreWaitForSecondFrame* (wcet = 0.72E-3, bcet = 0.53E-3) [T3]
- *PostWaitForSecondFrame* (wcet = 1.62E-3, bcet = 1.09E-3) [T4]
- *PreWaitForEnd* (wcet = 0.66E-3, bcet = 0.47E-3) [T6]
- *PostWaitForEnd* (wcet = 19.93E-3, bcet = 14.41E-3) [T7]
- *ProcessFrame* (wcet = 138.2E-6, bcet = 135.7E-6) [T_INT1 , T_INT2]

Tenemos que modelar de alguna forma la bajada de rendimiento que se produce durante la ejecución de *TransferSemiFrame*. Debido a la sensibilidad de la medida del tiempo en el PC (0.83 µseg.) no llegamos a apreciar con precisión lo que está ocurriendo pero si podemos calcular el overhead total que se produce en el intervalo. Realizando medidas con imágenes de distintos tamaños descubrimos que ese tiempo es proporcional al número de píxels y calculamos el valor del overhead medio por píxel , que resulta ser:

AvgPixelOverhead = 20 nseg.

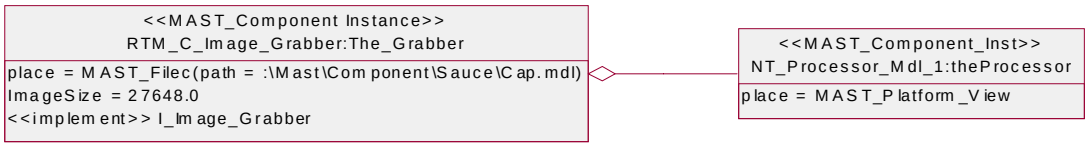
1. Las operaciones MAST_NOP no existen. se han incluido para poder describir el modelo en la herramienta de análisis.

A continuación debemos hacer una suposición sobre el tamaño de los paquetes transmitidos sobre el bus del sistema durante la ejecución de *TransferSemiFrame*. Podríamos suponer que un paquete equivale a un píxel , pero en ese caso el número de iteraciones que debería efectuar la herramienta de análisis de planificabilidad sería demasiado elevado. Para evitar este problema , vamos a suponer que en cada paquete se tranmiten *PixelInLine* píxels. Este valor es completamente arbitrario , ya que lo que nosotros hemos medido es el overhead total. La única restricción que tenemos es que el valor ha de ser múltiplo de todos los tamaños de imagen posibles. Por ello , en nuestro caso fijamos un valor de:

PixelInLine = 64

III.7. Código Mast-File del modelo del componente

A efecto de documentar de forma completa la naturaleza y significado del modelo CBSE_Mast del componente C_IG_DT3153 se listan las líneas de código que se insertan en el modelo Mast de la aplicación como consecuencia de la compilación del <<Mast_Component_Inst>> theGraber que es una instancial del <<component>> C_IG_DT3153.



De acuerdo con el modelo CBSE_Mast del componente C_IG_DT3153 que se ha descrito en la sección III.6, los elementos que el modelo introducen en el MAST_file son:

```
--*****************************************************************************
-- Segmento del código MAST-File del modelo RT_Comp_Railway
--
-------------------------------------------------------------------------------
-- the_Grabber (: RMT_C_Image_Grabber)
-------------------------------------------------------------------------------
-- place :C:/MAST/Component/Sauce/cap.mdl
-- Imagesize = 27648
-- host = theprocessor (para nosotros theprocessor.proc)
-- Buscamos ahí el modelo del componente MAST_C_I_Grabber
  -- parametros
     -- PixelInLine = 64
     -- Imagesize = 27648
     -- AvgPixelOverhead = 0.02E-6
     -- SemiFrame_Duration = 18E-3
     -- Driver_Priority = host.Max_interrupt_Priority = 32
     -- Driver_host = host = theprocessor
  -- RTM_C_Image_Grabber.AttendFrameTRans_SS (:MAST_FP_Sched_Server)
Scheduling_Server
 (Type                       => Fixed_Priority,
  Name                       => the_Grabber.AttendFrameTrans_SS,
  Server_Sched_Parameters    =>
```

```
    (Type                        => Interrupt_FP_Policy,
     The_Priority                => 32
    ),
  Server_Processing_Resource => theProcessor.proc
 );
  -- RTM_C_Image_Grabber.DMA_Bus (MAST_FP_Network)
  ----------------------------
  -- MAST_FP_Network
  ----------------------------
  -- Declaramos una network con los atributos que aparecen en la descripción
Processing_Resource
 (Type                      => Fixed_Priority_Network,
  Name                      => the_Grabber.DMA_Bus,
  Transmission              => Simplex,
  Max_Blocking              => 41.6E-6,
  Max_Packet_Transmission_Time => 41.6E-6,
  Min_Packet_Transmission_Time => 41.6E-6,
  List_of_Drivers           =>
   ((Type                      => Packet_Driver,
     Packet_Server             =>
       (Type                       => Fixed_Priority,
        Name                       => the_Grabber.DMA_Bus.theDriver.Packet_Server,
        Server_Sched_Parameters    =>
         (Type                        => Interrupt_FP_Policy,
          The_Priority                => 32
         ),
        Server_Processing_Resource => theProcessor.proc
       ),
     Packet_Send_Operation    =>
       (Type                     => Simple,
        Name                     => the_Grabber.DMA_Bus.theDriver.Packet_Send,
        Worst_Case_Execution_Time=> 1.28E-6,
        Best_Case_Execution_Time => 1.28E-6),
        Packet_Receive_Operation =>
          (Type                     => Simple,
           Name                     => the_Grabber.DMA_Bus.theDriver.Packet_Receive,
           Worst_Case_Execution_Time=> 1.28E-6,
           Best_Case_Execution_Time => 1.28E-6
          )
       )
 ));
-------------------------------------------------------------------------------
-- RMT_C_Image_GRabber.Bus_TRansfer_SS ( : MAST_FP_Sched_Server)
-------------------------------------------
Scheduling_Server
 (Type                       => Fixed_Priority,
  Name                       => the_Grabber.Bus_Transfer_SS,
  Server_Sched_Parameters     =>
   (Type                        => Fixed_Priority_Policy
   ),
    Server_Processing_Resource   => the_Grabber.DMA_Bus
  );
-------------------------------------------------------------------------------
-- RMT_C_Image_GRabbe.Image_Processor (type :Mast_Device)
-------------------------------------------
-- MAST_Device
----------------------
-- Corresponde a una FP_Processor sin atributos

Processing_Resource
 (Type                       => Fixed_Priority_Processor,
  Name                       => the_Grabber.Image_Processor
 );
```

```
-------------------------------------------------------------------------------
-- MainProcessor.Image_Process (type : Mast_System_SS)
-- Corresponde a un SS Non preemtible con prioridad por defecto
-----------------------------------------
Scheduling_Server
 (Type                       => Fixed_Priority,
  Name                       => the_Grabber.Image_Process,
  Server_Sched_Parameters    =>
    (Type                       => Interrupt_FP_Policy,
     The_Priority => 32
    ),
  Server_Processing_Resource=> the_Grabber.Image_Processor
 );

-- Declaramos ahora todas las operaciones simples que aparecen
Operation
  (Type                       => Simple,
   Name                       => The_Grabber.WaitForFirstFrame,
   Worst_Case_Execution_Time => 1.2E-5,
   Best_Case_Execution_Time  => 1.2E-5
  );
Operation
  (Type                       => Simple,
   Name                       => The_Grabber.TRansferSemiFrame,
   Worst_Case_Execution_Time => 18.0E-3,
   Best_Case_Execution_Time  => 18.0E-3
  );
Operation
  (Type                       => Simple,
   Name                       => The_Grabber.PreWaitForSecondFrame,
   Worst_Case_Execution_Time => 0.72E-3,
   Best_Case_Execution_Time  => 0.53E-3
  );
Operation
  (Type                       => Simple,
   Name                       => The_Grabber.PostWaitForSEcondFrame,
   Worst_Case_Execution_Time => 1.62E-3,
   Best_Case_Execution_Time  => 1.09E-3
  );
Operation
  (Type                       => Simple,
   Name                       => The_Grabber.PreWaitForEnd,
   Worst_Case_Execution_Time => 0.66E-3,
   Best_Case_Execution_Time  => 0.47E-3
  );
Operation
  (Type                       => Simple,
   Name                       => The_Grabber.PostWaitForEnd,
   Worst_Case_Execution_Time => 19.93E-3,
   Best_Case_Execution_Time  => 14.41E-3
  );
Operation
  (Type                       => Simple,
   Name                       => The_Grabber.ProcessFrame,
   Worst_Case_Execution_Time => 18.0E-3,
   Best_Case_Execution_Time  => 18.0E-3
  );
-- Job parametrizables
-----------------------------------------
--Operation(
--     Type  => Simple,
--     Name  => The_Grabber.I_Image_Grabber.Read,
--     Worst_Case_Execution_Time => 36.4E-3 + 0.15E-6*ImgSize,
```

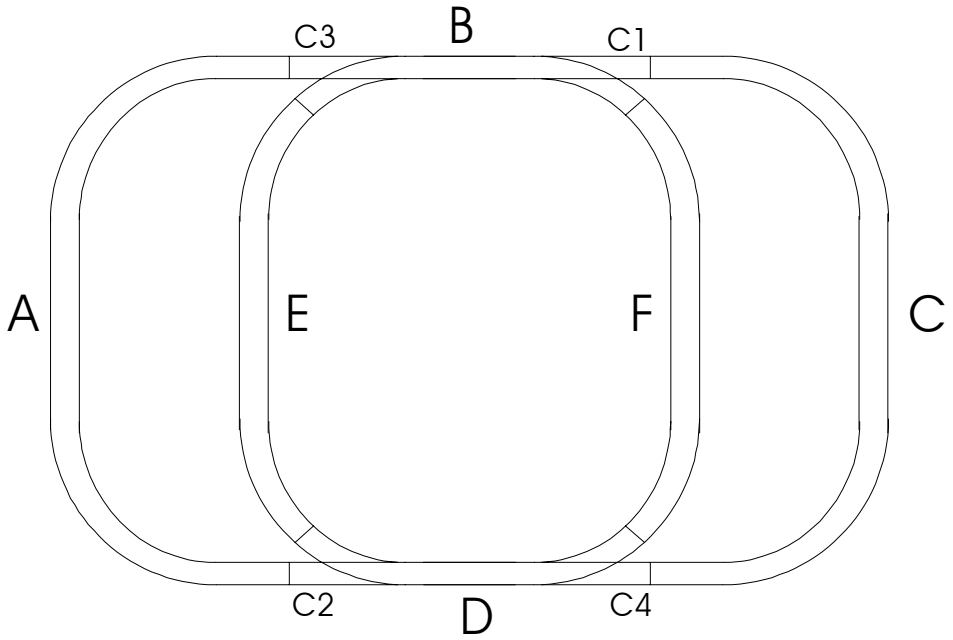
```
--     Best_Case_Execution_Time  => 36.4E-3 + 0.14E-6*ImgSize);
-----------------------------------------
-- Anadida desde al transaccion
Operation
  (Type                      => Simple,
   Name                      => The_Grabber.I_Image_Grabber.Read_1,
   Worst_Case_Execution_Time => 40.55E-3,
   Best_Case_Execution_Time  => 40.27E-3);
--*************************************************************************
```

IV. DEMOSTRADOR DE UN SISTEMA DE TIEMPO REAL BASADO EN COMPONENTES

IV.1. Control por computador de una maqueta de trenes utilizando visión artificial

Se ha desarrollado como ejemplo una aplicación sencilla que hace uso de algunas de las interfaces que son implementadas por los tres componentes desarrollados y que consiste en el control de una maqueta de trenes de escala N, en la que la posición de las locomotoras se detecta mediante visión artificial. Esta aplicación desarrollada tiene la finalidad de validar los componentes, tanto en lo relativo a la validez y completitud de su especificación, a la capacidad de interconectividad y a la predecibilidad que aportan sus modelos de tiempo real, y sobre todo permite obtener experiencia sobre el ciclo de desarrollo de aplicaciones basadas en componentes. También comprobamos la validez del análisis de planificabilidad de peor caso utilizando la herramienta MAST propia para tal fin.

Se dispone de una maqueta de trenes de la marca *Roco* de escala N instalada sobre una base rígida debidamente cableada para facilitar su control de modo electrónico. El aspecto que presenta (vista desde arriba) es el siguiente:

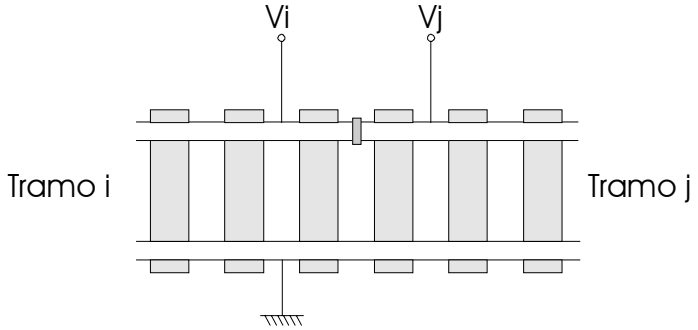


Consta de tres tipos de elementos:

LOCOMOTORAS: ROJA y AZUL .- También denominadas trenes. Se trata de las máquinas que se mueven sobre las vías. En su parte superior llevan instalada una tarjeta de un determinado color que permite identificar su posición mediante visión artificial. En nuestro caso tenemos un tren azul y un tren rojo. Son locomotoras de escala N de la marca *Ibertren*.

TRAMOS: A , B , C , D , E y F .- Segmentos de vía comprendidos entre dos cruces. Cuando una locomotora se encuentra situada sobre un tramo , su motor se alimenta

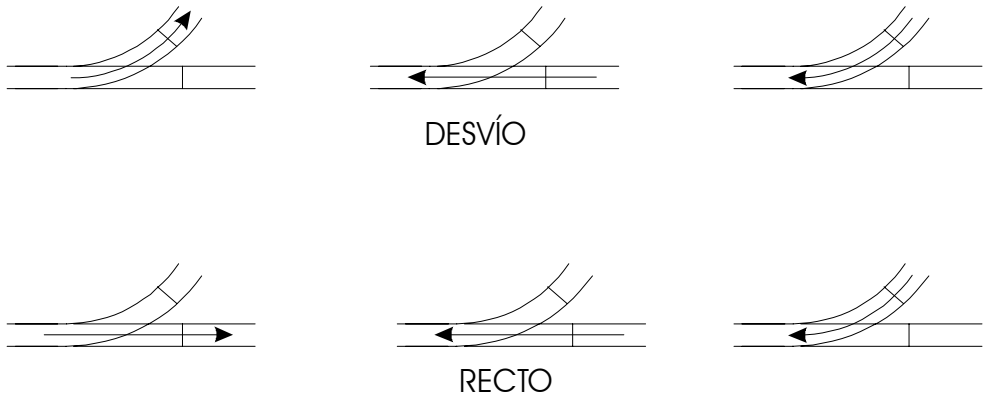
electricamente a través del propio tramo de vía. Para ello se ha cableado la maqueta como se muestra en la figura de modo que cada tramo se puede alimentar con una tensión diferente. La velocidad del tren varía de forma monótona con la tensión de alimentación de la vía. Así mismo, si se invierte la polaridad de la tensión, se invierte el sentido del desplazamiento del tren sobre la vía. Las locomotoras *Ibertren* utilizadas admiten un rango de tensión de alimentación desde 0 hasta 15 voltios, pero en la aplicación del demostrador las locomotoras solo se mueven en un sentido dentro de cada tramo y solo utilizamos tres valores posibles de alimentación:



- **STOPPED**: se aplican 0 voltios al tramo. La locomotora situada en el tramo permanece parada.
- **LOW_SPEED**: se aplican 7 voltios al tramo. La locomotora situada en el tramo se desplaza a velocidad lenta en el sentido de las agujas del reloj.
- **HIGH_SPEED**: se aplican 13.5 voltios al tramo. La locomotora situada en el tramo se desplaza a velocidad rápida en el sentido de las agujas del reloj.

Para que sea posible el control independiente de las dos locomotoras, se establece la restricción de que nunca puedan estar ambas en un mismo tramo.

CRUCES: C1 , C2 , C3 y C4 .- Cuatro cruces idénticos , cada uno de ellos con dos posibles estados denominados *RECTO* y *DESVÍO*. En función del estado y del tramo del que proceda la locomotora , ésta saldrá hacia un tramo determinado tal y como se indica en la figura:

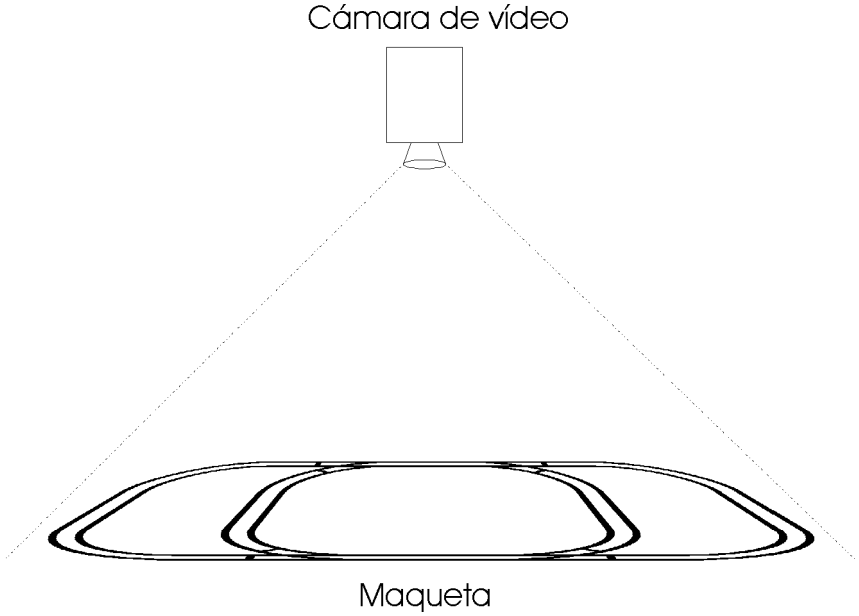


Como en la aplicación las locomotoras siempre se van a desplazar en el sentido de las agujas del reloj sólo necesitamos controlar los cruces C1 y C2 , mientras que el estado de C3 y C4 va a ser indiferente. El control se realiza mediante un electroimán (integrado en la propia pieza que conforma el cruce) con tres conexiones electrónicas: una de ellas a tierra (0 voltios),

otra para pasar a *RECTO* y otra más para pasar al estado *DESVÍO*. La manera de pasar a un determinado estado es aplicar en la conexión correspondiente un pulso de 15 voltios de amplitud de una determinada duración, en nuestro caso 300 mseg (esto no implica que el cambio se haga efectivo al comienzo o al final del pulso , ya que el electroimán es un dispositivo electromecánico). Lo que no disponemos es de información electrónica de retorno que nos indique el estado actual de un cruce, de modo que esta información deberá de ser mantenida por la aplicación software y ser gestionada adecuadamente como variables de estado.

Por la conectividad que se ha establecido en la maqueta, los cruces C1 y C3 son parte[1] del tramo B , mientras que C2 y C4 son parte del tramo D.

Para poder controlar de forma segura la operación de los trenes en la maqueta, se necesita conocer la posición de las locomotoras dentro de la maqueta. La posición de las locomotoras se detecta mediante visión artificial, utilizando para ello una cámara situada sobre la maqueta y una tarjeta de captura y digitalización de imagenes del tipo DT3153 de Data Translation.



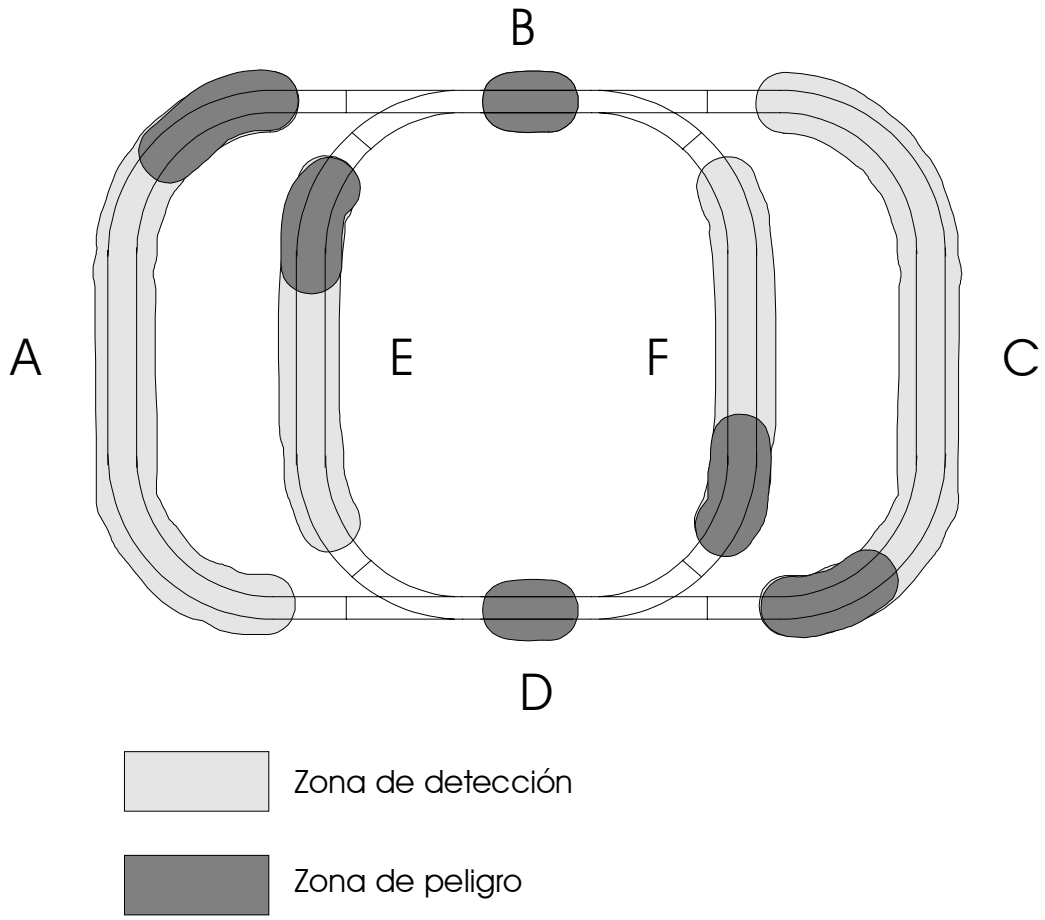
Como el fondo de la base rígida de la maqueta es básicamente verde , un tren rojo y el otro azul podemos detectar sus posiciones dentro de la imagen mediante análisis del color y determinar si una locomotora está situada en una zona definida previamente. Ya que nos interesa que la operación de localización sea lo más rápida posible, debemos trabajar con imágenes con la menor resolución, pero que sean compatibles con la precisión de localización que se necesita. En nuestro caso se utilizan imágenes de 192 por 144 píxels[2] (27648 píxels en total).

Se han definido varias zonas sobre las vías en las que va a ser de interés detectar la presencia de un tren en ella. Estas zonas se han determinado mediante edición manual de una

1. En cuanto a la alimentación de las locomotoras (desplazamiento).
2. Tamaño R_4 del tipo *Ratio* en la interface *Image_Grabber.*

imagen de referencia utilizando una aplicación independiente diseñada para tal fin. También se ha implementado otra aplicación que permite recolocar la posición de la maqueta respecto a la cámara para conseguir que la referencia de las zonas sea válida. Para esta última se ha utilizado una técnica basada en imágenes superpuestas sobre vídeo en vivo (*overlapping*). Hay dos tipos de zonas:

- **Zonas de detección**: se trata de 6 zonas, una por cada tramo, que implican la existencia del tren en el tramo correspondiente.
- **Zonas de peligro**: se trata de 6 zonas, una por cada tramo, que implican la existencia del tren en una posición cercana a un cruce. Cuando se detecte un tren en esta zona se deben tomar una serie de decisiones (parar si el siguiente tramo está ocupado, o si no es así colocar el cruce en el estado adecuado). Estas zonas estan comprendidas dentro de la zona de detección respectiva, y en el caso de los tramos B y D coinciden completamente (debido a su pequeño tamaño).



El control de la maqueta (estado de los cruces y alimentación de los tramos) se realiza utilizando la tarjeta de I/O PCI9111, en concreto 16 de sus salidas digitales. Estas señales son de baja potencia[1] , mientras que los elementos de la maqueta requieren señales de potencia. Por ello ha sido necesario diseñar e implementar una tarjeta intermedia de relés y cablearla de

1. TTL.

manera adecuada entre la tarjeta de adquisición I/O y la maqueta para poder efectuar el control. El diseño y cableado de esta tarjeta intermedia de relés se describe detalladamente en el anexo B.

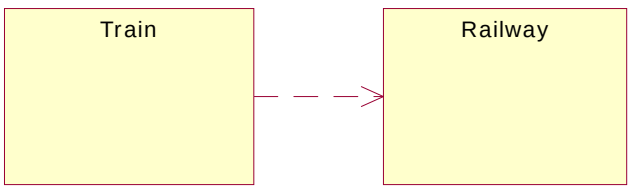
IV.2. Funcionalidad de la aplicación

En el demostrador, cada una de las locomotoras debe recorrer una ruta cerrada de forma cíclica durante un determinado número de vueltas. En nuestro caso se le asigna al tren azul el recorrido externo (tramos A - B - C - D) y al tren rojo el recorrido interno (tramos E - B - F - D). El tren azul inicialmente está situado en una posición central del tramo A y el rojo en una posición central del tramo E de modo que al comenzar la ejecución ambos son detectados en la zona de detección del tramo correpondiente sin ningún problema.

La detección se realiza de forma periódica durante toda la ejecución del programa. Cuando se detecta que un tren llega a alguna zona de peligro lo primero que se hace es establecer su velocidad a *LOW_SPEED*. A continuación se comprueba si el siguiente tramo del recorrido es accesible (no está ocupado) y en ese caso se pone el cruce en la posición adecuada y se pasa la velocidad a *HIGH_SPEED*. Si el siguiente tramo no es accesible se detiene el tren (*STOPPED*) y se espera hasta que quede libre; en ese momento se pone el cruce en la posición adecuada y se pasa la velocidad a *HIGH_SPEED*. Un tramo se considera que queda libre cuando el tren que lo ocupaba llega a su siguiente tramo.

IV.3. Arquitectura software del demostrador

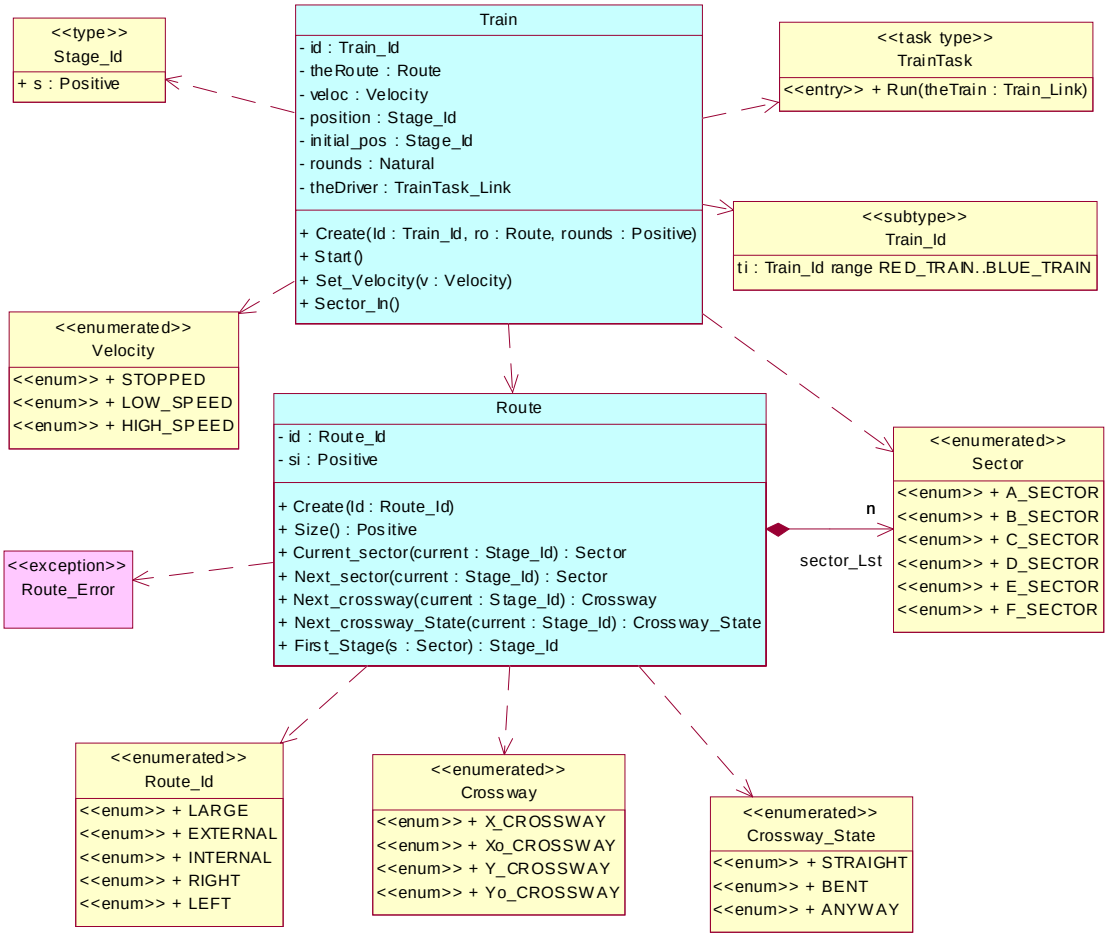
El programa Ada 95 que constituye la aplicación se ha construido a partir de dos clases principales:



- Class **Train**: Representa el módulo software que controla la circulación de un tren de acuerdo con una ruta que se le ha asignado. Existe una instancia de esta clase por cada tren que circula en la aplicación.
- Class **Railway**: Modela la librería de procedimientos de control de los elementos de la red ferroviaria e inspección de su estado.

En la figura de la siguiente página se muestra la descripción UML de la clase Train. En ella se muestra, el conjunto de atributos que representan el estado de cada instancia de la clase, su interfaz constituida por el conjunto de procedimientos y funciones con las que el procedimiento principal que constituye la aplicación gestiona la operación de la instancia y el conjunto de clases auxiliares que definen los diferentes tipos de variables utilizados en la clase Train.

La semántica de las diferentes clases está bastante autoexplicada, y en cualquier caso su explicación detallada queda fuera del objeto de esta memoria.

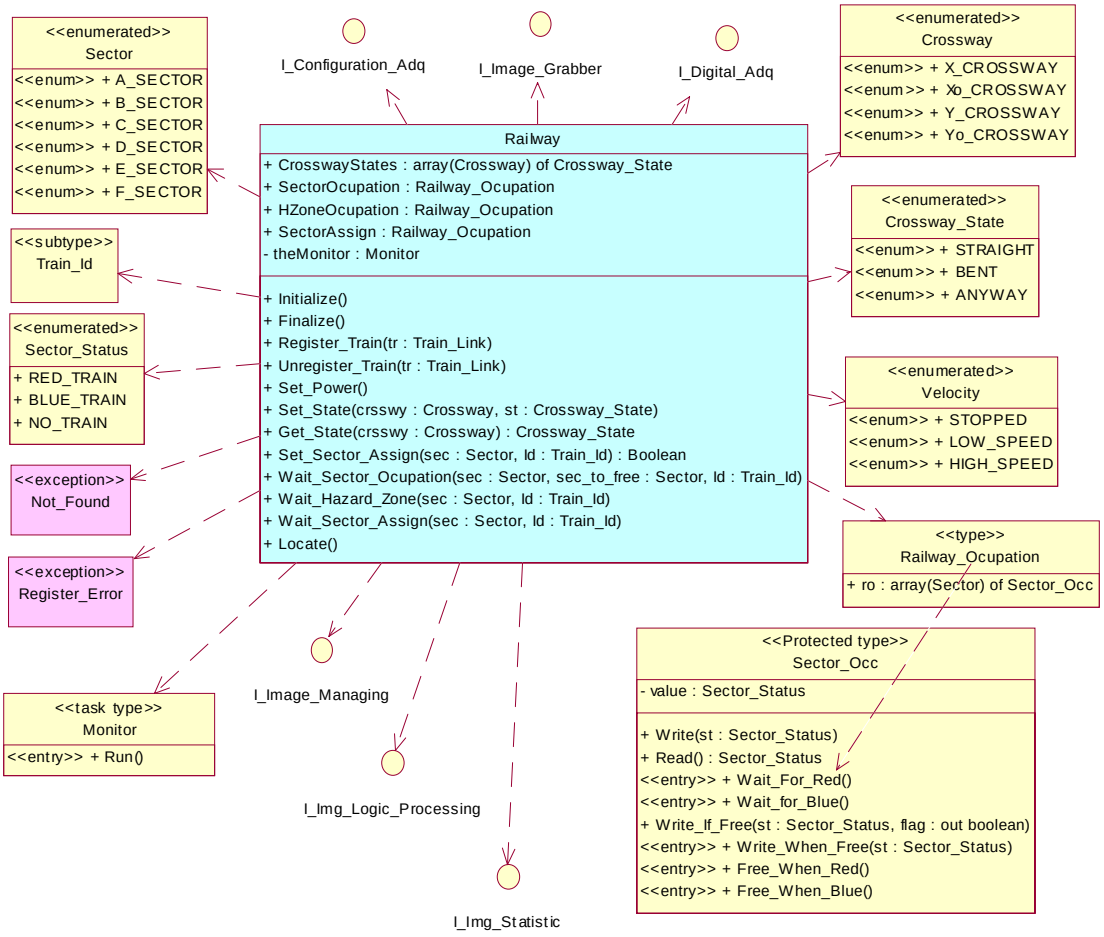


Así mismo, en la figura de la siguiente página se muestra el modelo de la clase Railway, con sus atributos, su interfaz de funciones y procedimientos, y el conjunto de clases asociadas para definir los tipos utilizados en ellos.

En el diagrama de clases de Railway se indica que la clase Railway hace uso de las interfaces, I_Configuration_Adq, I_Digital_Adq, I_Grabber, I_Image_Mannaging, I_Image_Statistic e I_Image_Logic_Processing.

Dentro de la herramienta UML CASE, se han establecido junto a estos diagramas de clases, la documentación que explica la semántica de cada componente, los diagramas de actividad, secuencia y colaboración que ilustran su funcionalidad, y las posibilidades de interacción con otros componentes que ofrecen, y las directivas de generación de código que permiten la generación directa de la especificación del paquete Ada95, y el esqueleto del cuerpo de del paquete Ada 95.

En este caso, a través del diagrama de componentes se ha elegido que el código de ambas clases se incluya en un único módulo de código.



A fin de documentar de forma precisa la arquitectura software de la aplicación se incluye a continuación la especificación del paquete Railway_Resources.ads.

```
--*****************************************************************************
-- Demostrador: CBSE_Railway
-- Especificación del paquete Railway_Resources
--*****************************************************************************
with List_Generic;

package Railway_Resources is

   ---- TIPOS -------------------------------------------------------

   type Sector_Status is (RED_TRAIN,BLUE_TRAIN,NO_TRAIN);
   subtype Train_Id is Sector_Status range RED_TRAIN..BLUE_TRAIN;
   type Sector is (A_SECTOR,B_SECTOR,C_SECTOR,D_SECTOR,E_SECTOR,F_SECTOR);
   type Velocity  is (STOPPED,LOW_SPEED,HIGH_SPEED);
   type Crossway is (X_CROSSWAY,Xo_CROSSWAY,Y_CROSSWAY,Yo_CROSSWAY);
   type Crossway_State is (STRAIGHT,BENT,ANYWAY);
   type Route_Id is (LARGE,EXTERNAL,INTERNAL,RIGHT,LEFT);
```

```
type Stage_Id is Positive;

type Train is private;
type Train_Link is access Train;

type Route is private;

protected type Sector_Occ is
   procedure Write(st:Sector_Status);
   function Read return Sector_Status;
   entry wait(Train_Id);
   entry Wait_For_Red;
   entry Wait_For_Blue;
   procedure Write_If_Free(st:Sector_Status;flag:out Boolean);
   entry Write_When_Free(st:Sector_Status);
   entry Free_When_Red(sector_to_free:Sector);
   entry Free_When_Blue(sector_to_free:Sector);
private
   value:Sector_Status;
end Sector_Occ;

type Railway_Ocupation is array(Sector)of Sector_Occ;

---- VARIABLES DE ESTADO ---------------------------------------------

CrosswayStates :array(Crossway)of Crossway_State;
SectorOcupation:Railway_Ocupation;
HZoneOcupation :Railway_Ocupation;
SectorAssign   :Railway_Ocupation;

---- OPERACIONES-----------------------------------------------------

-- Railway

procedure Initialize;
procedure Finalize;
procedure Register_Train(tr:Train_Link);
procedure Unregister_Train(tr:Train_Link);
procedure Set_Power;
procedure Set_State(crsswy:Crossway;st:Crossway_State);
function Get_State(crsswy:Crossway)return Crossway_State;
function Set_Sector_Assign(sec:Sector;Id:Train_Id)return Boolean;
procedure Wait_Sector_Ocupation(sec:Sector;sec_to_free:Sector;Id:Train_Id);
procedure Wait_Hazard_Zone(sec:Sector;Id:Train_Id);
procedure Wait_Sector_Assign(sec:Sector;Id:Train_Id);
procedure Locate;

-- Train

procedure Create(Self: Train_Link;Id: Train_Id;ro: Route;rounds: Positive);
procedure Start(Self: Train_Link);
procedure Set_Velocity(Self: Train_Link;v: Velocity);
procedure Sector_In(Self: Train_Link);

-- Route

procedure Create(Self:in out Route;Id:Route_Id);
function Size (Self:Route) return Positive;
function Current_sector(Self:Route;current:Stage_Id)return Sector;
function Next_sector(Self:Route;current:Stage_Id)return Sector;
function Next_crossway(Self:Route;current:Stage_Id)return Crossway;
function Next_crossway_State(Self:Route;current:Stage_Id)return Crossway_State;
function First_Stage(Self:Route;s:Sector)return Stage_Id;
```

```
   ---- EXCEPCIONES ----------------------------------------------------

   Register_error : exception;
   Not_Found      : exception;
   Route_Error    : exception;

private

   RegisterTrains: array(Train_Id)of Train_Link;

   task type TrainTask is
      entry Run(theTrain:Train_Link);
   end TrainTask;

   type TrainTask_Link is access TrainTask;

   type Train is record
      id         : Train_Id;
      theRoute   : Route;
      veloc      : Velocity;
      position   : Stage_Id;
      initial_pos: Stage_Id;
      rounds     : Natural;
      theDriver  : TrainTask_Link:=new TrainTask;
   end record;

   package Sector_Object_List is new List_Generic (Sector);

   type Route is record
      id         : Route_Id;
      sector_Lst : Sector_Object_List.List;
      si         : Positive;
   end record;

   task type Monitor is
      entry Run;
   end Monitor;

   theMonitor: Monitor;

end Railway_Resources;
--*****************************************************************************
```

El próximo listado de código muestra el programa principal de la aplicación demostrador que hace uso de los recursos ya definidos.

```
--*****************************************************************************
-- Demostrator: CBSE_Railway
-- Main procedure
--*****************************************************************************
with Railway_Resources;
with Ada.Text_IO;
with Ada.Exceptions;
with Win32;
with Win32.Winbase;
with Win32.Winnt;

procedure Main is

   trenRojo: Railway_Resources.Train_Link;
```

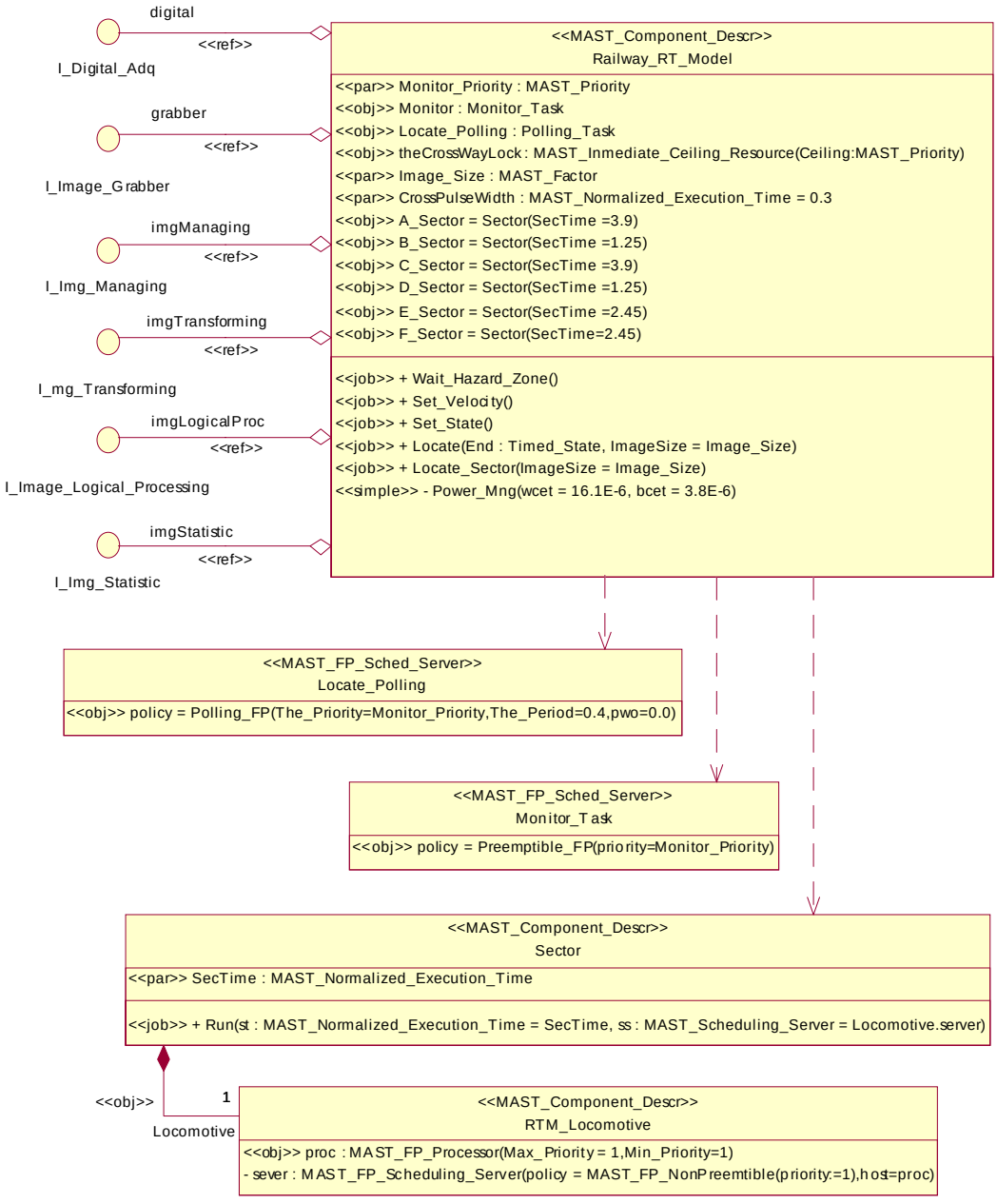
```
   trenAzul: Railway_Resources.Train_Link;
   rutaR   : Railway_Resources.Route;
   rutaB   : Railway_Resources.Route;

   hProcess : Win32.Winnt.HANDLE;
   b        : Win32.BOOL;

begin

   hProcess:=Win32.Winbase.GetCurrentProcess;
   b:=Win32.Winbase.SetPriorityClass(hProcess,
                                     Win32.Winbase.REALTIME_PRIORITY_CLASS);

   Railway_Resources.Create(rutaR,Railway_Resources.INTERNAL);
   Railway_Resources.Create(rutaB,Railway_Resources.EXTERNAL);

   Railway_Resources.Create(trenRojo,
                            Railway_Resources.RED_TRAIN,
                            rutaR,
                            3);
   Railway_Resources.Create(trenAzul,
                            Railway_Resources.BLUE_TRAIN,
                            rutaB,
                            3);

   Ada.Text_IO.Put_Line("Pulsa intro para comenzar");
   Ada.Text_IO.Skip_Line;

   Railway_Resources.Start(trenRojo);
   Railway_Resources.Start(trenAzul);

   Ada.Text_IO.Put_Line("Pulsa intro cuando se detengan los trenes");
   Ada.Text_IO.Skip_Line;

   Railway_Resources.Finalize;

exception

   when Railway_Resources.Register_Error =>
      Ada.Text_IO.Put_Line("Error Register_Error");
      raise;
   when Railway_Resources.Route_Error =>
      Ada.Text_IO.Put_Line("Error Route_Error");
      raise;
   when Railway_Resources.Not_Found =>
      Ada.Text_IO.Put_Line("Error Not_Found");
      raise;
   when E:others =>
      Ada.Text_IO.Put_Line("Error desconocido");
      Ada.Text_IO.Put_Line(Ada.Exceptions.Exception_Message(E));
      raise;
end Main;
--*****************************************************************************
```

IV.4. Modelo de los componentes lógicos de la aplicación

Para construir el modelo de tiempo real de la aplicación, es necesario construir en primer lugar el modelo de las dos clases Train y Railway diseñadas específicamente para construir la aplicación. En ambos casos, el modelo se ha formulado como un descriptor <<MAST_component_descr>>, que incluye todos los elementos necesarios para construir el modelo de la instancia.

IV.4.1. Modelo MAST de la clase Railway

En la siguiente figura se muestra el modelo MAST de la clase lógica Railway.



El modelo MAST de la clase lógica Railway contiene la siguiente información:

• Identificador: **Railway_RT_Model**: Identificador que debe utilizarse para declarar un modelo que sea una instancia de este descriptor.

• Parámetros:

- <<par>> **Monitor_Priority** : MAST_Priority: Permite establecer en cada instancia la prioridad en que se ejecuta la tarea que realiza el análisis del estado de la maqueta mediante visión artificial.
- <<par>> **Image_Size** : MAST_Factor: Permite establecer en cada instancia el tamaño en píxels de las imágenes adquiridas para la detección. Son utilizadas para escalar los tirempos de procesado de las imágenes.
- <<par>> **CrossPulseWidth** : MAST_Normalized_Execution_Time = 0.3. Permite establecer en cada instancia del modelo la duración del pulso que debe ser aplicado a los electroimanes de los cruces de vía. Aunque es un tiempo normalizado, se corresponde con un tiempo físico, ya que se ejecuta en un Device que tiene SpeedFactor=1.0.

• Referencias:

- <<ref>> **digital** :MAST_Interface_Descr: Permite establecer en cada instancia el modelo del objeto que implementa la interface del tipo I_Digital_Adq que se utiliza.
- <<ref>> **grabber** :MAST_Interface_Descr:Permite establecer en cada instancia el modelo del objeto que implementa la interface del tipo I_Image_Grabber que se utiliza.
- <<ref>> **imgManaging** :MAST_Interface_Descr:Permite establecer en cada instancia el modelo del objeto que implementa la interface del tipo I_Img_Managing que se utiliza.
- <<ref>> **imgTransforming** :MAST_Interface_Descr:Permite establecer en cada instancia el modelo del objeto que implementa la interface del tipo I_Img_Transforming que se utiliza.
- <<ref>> **imgLogicalProc** :MAST_Interface_Descr:Permite establecer en cada instancia el modelo del objeto que implementa la interface del tipo I_Img_Logical_Processing que se utiliza.
- <<ref>> **imgStatistic** :MAST_Interface_Descr:Permite establecer en cada instancia el modelo del objeto que implementa la interface del tipo I_Img_Statistic que se utiliza.

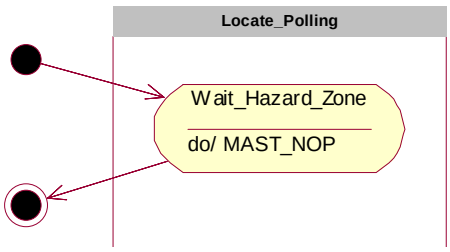
• Objetos:

- <<obj>> **Monitor** : Monitor_Task: Por cada instancia del modelo se intancia un componente del tipo Monitor_Task, tambíen declarado en el modelo y que es un

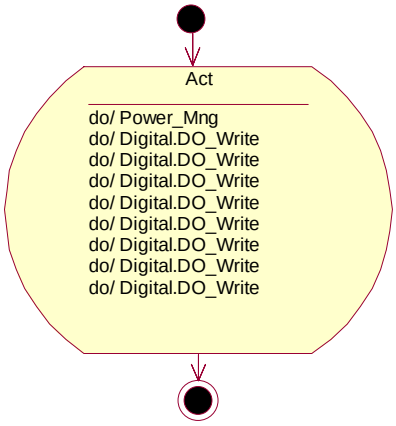
MAST_FP_Sched_Server, con politica y prioridad declarada. Modela la tarea en la que se ejecuta el código de evaluación del estado de la maqueta por visión artificial. Ejecuta periodicamente (cada 350 mseg.) las operaciones Locate y Locate_Sector que detecta la posición de los trenes.

- <<obj>>**Locate_Polling** : Polling_Task: Por cada instancia del modelo se intancia un componente del tipo Locate_Polling, tambíen declarado en el modelo y que es un MAST_FP_Sched_Server, con politica de planificación del tipo Polling_FP. La tarea *Polling_Task* es una tarea periódica que solo existe en el modelo y que se ha introducido para modelar el hecho de que no detectamos los trenes en el preciso instante en que llegan a la zona de peligro, sino que tenemos una cierta incertidumbre debido a que la detección se realiza cada 350 mseg. En esta tarea se ejecuta la operación *Wait_Hazard_Zone* , que tiene duración nula ya que se trata simplemente de una espera hasta que el resultado de la detección nos indica que el tren a llegado a una determinada zona de peligro.
- <<obj>> **theCrossWayLock** :MAST_Inmediate_Ceiling_Resource(Ceiling: MAST_ Priority): Por cada instancia del modelo se intancia un shared resource que modela el acceso con exclusión mútua al hardware que controla los cruces de vías de la maqueta.
- <<obj>> **A_Sector** = Sector(SecTime =3.9), <<obj>> **B_Sector** = Sector(SecTime =1.25), <<obj>> **C_Sector** = Sector(SecTime =3.9), <<obj>> **D_Sector** = Sector(SecTime =1.25), <<obj>> **E_Sector** = Sector(SecTime =2.45) y <<obj>> **F_Sector** = Sector(SecTime=2.45): modelan los seis tramos de la maqueta. Cada uno de ellos tiene un parámetro *SecTime* que indica el tiempo en segundos que tarda un tren (suponemos que el tren rojo y el azul son idénticos y por tanto tardan lo mismo) en recorrer el sector si no se detiene nunca (en nuestro caso se midió con un sólo tren recorriendo todos los sectores de la maqueta). La operación *Run* representa la acción de recorrer todo el sector a velocidad HIGH_SPEED, de modo que sus tiempos (wcet y bcet) son iguales al parámetro *SecTime*. De cualquier modo, en este tiempo el procesador no se utiliza y queda libre para realizar otras operaciones.

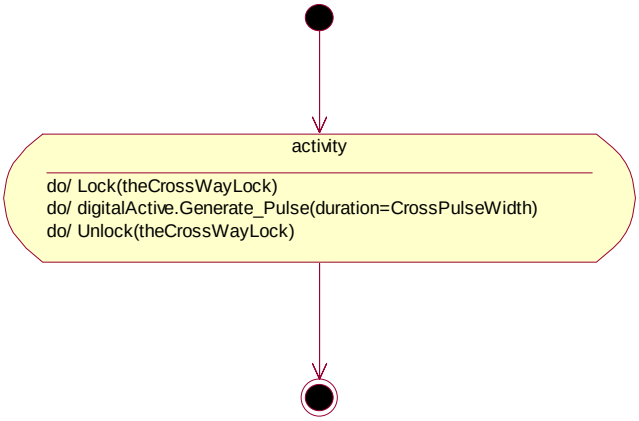
- Modos de uso:
  - <<job>> + **Wait_Hazard_Zone**(): Modela la incertidumbre que resulta en la determinación de un tren a una zona de peligro como consecuencia de que el estado de la maqueta se determina con una granularidad de 350 ms. Consiste en la ejecución de una operacion de duración nula (MAST_NOP) en el scheduling server LocatePolling que tiene una política de planificación del tipo Polling_FP.



- <<job>> + **Set_Velocity**():modela la asignación de las tensiones a los distintos tramos en función de la velocidad de los trenes que se encuentran sobre ellos. Se describe mediante el siguiente diagrama de actividad:

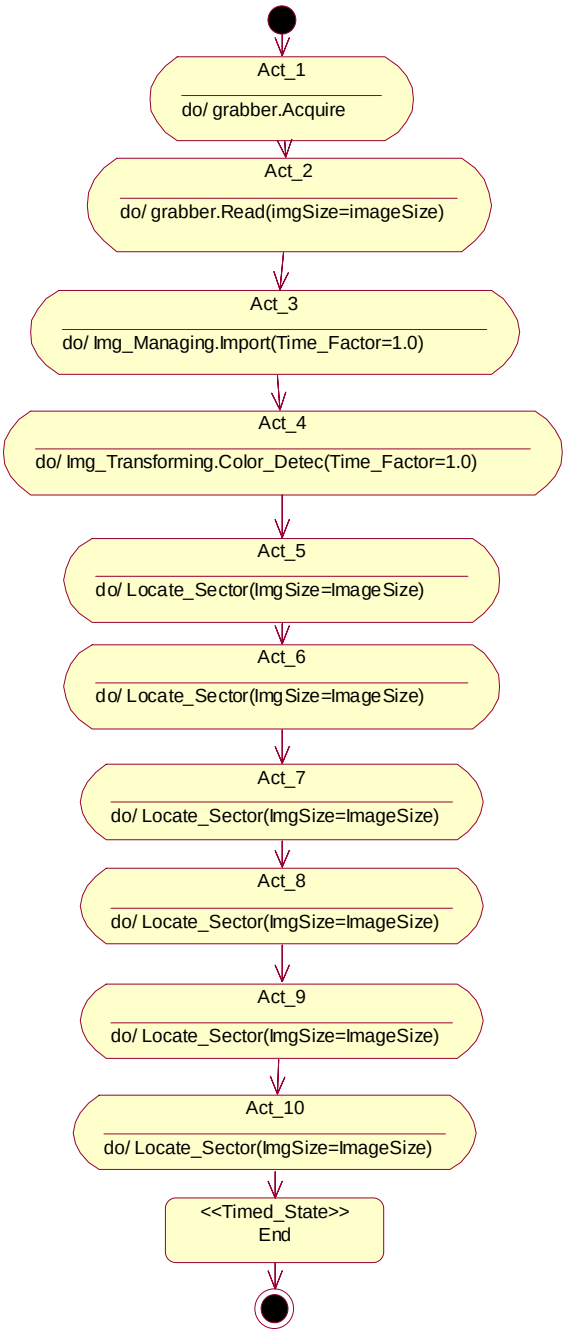


- <<job>> + **Set_State()** modela el control de los cruces y se describe mediante el siguiente diagrama de actividad:



- <<job>> + **Locate**(End : Timed_State, ImageSize = Image_Size): Modela la se cuencia de operaciones que se ejecutan para capturar, digitalizar, y procesar la imagen a fin de localizar la posición de las locomotoras sobre la vía. Se describe mediante el siguiente diagrama de actividad

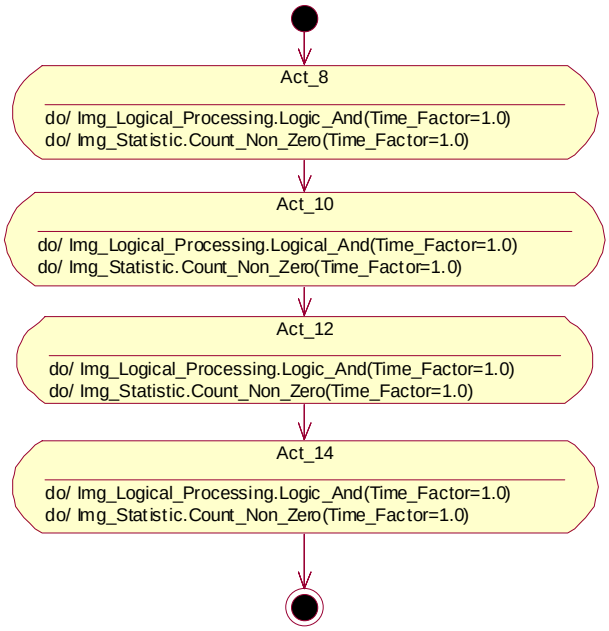
:



Se adquiere la imagen , se importa a un formato propio del componente de análisis de imagen y se detectan los trenes (colores rojo y azul) utilizando la operación *Color_Detec*, en nuestro caso con los siguientes umbrales de detección:

	Rmax	Rmin	Gmax	Gmin	Bmax	Bmin
Tren Rojo	255	150	200	50	255	50
Tren Azul	110	1	255	90	255	175

- **Locate_Sector**(ImageSize = Image_Size): Finalmente se ejecuta para cada uno de los seis sectores la operación *Locate_Sector* , descrita mediante el siguiente diagrama de actividad:



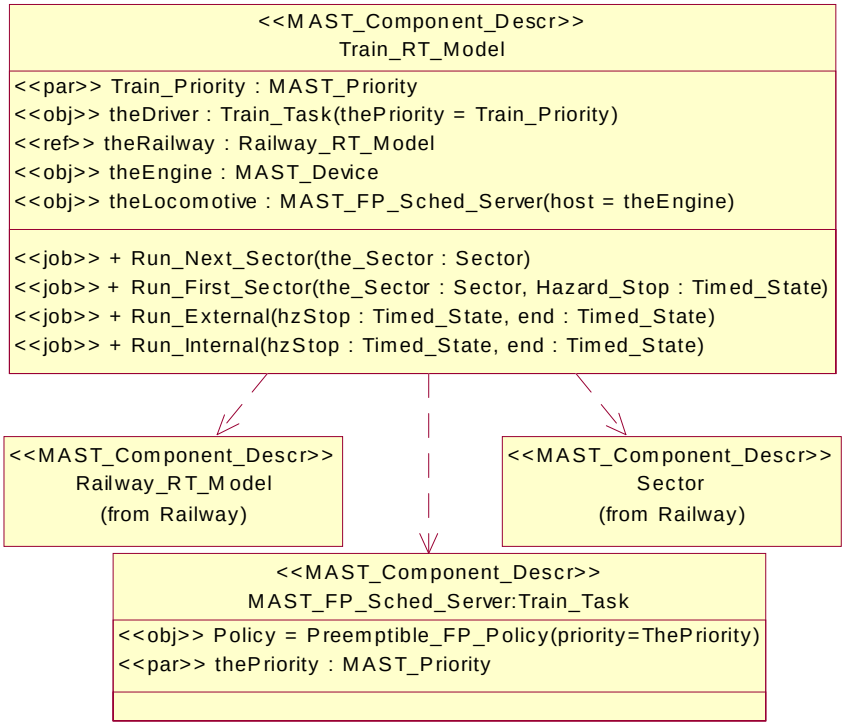
Se repite cuatro veces (para las zonas de detección y peligro del sector para el tren rojo y el azul) la siguiente operación: se ejecuta una operación lógica And entre el resultado del análisis de color y la máscara de la zona correspondiente (almacenada en memoria en la inicialización de la aplicación) y se cuenta el número de píxels blancos de la imagen obtenida. Si este número supera un umbral (en nuestro caso es de 8 píxels) suponemos que la detección es positiva , y si no lo supera suponemos que es negativa.

- <<simple>> - **Power_Mng**(wcet = 16.1E-6, bcet = 3.8E-6): Modela las operaciones de gestión para determinar las posiciones de los relés de la tarjeta intermedia de control en función de las tensiones necesarias en los distintos tramos y se describe como una operación <<*simple*>>. Las dos primeras escrituras digitales determinan las dos tensiones necesarias (T1 y T2) y las seis siguientes las asignan a los seis tramos adecuadamente[1].

1. Ver anexo B.

IV.4.2. Modelo MAST de la clase Train

En la siguiente figura se muestra el modelo MAST de la clase lógica Train.



El modelo MAST de la clase lógica Train contiene la siguiente información:

- Identificador: Train_RT_Model:Identificador que debe utilizarse para declarar un modelo que sea una instancia de este descriptor.

- Parámetros:
  - <<par>> **Train_Priority** : MAST_Priority: A través de este parámetro se puede asignar a cada instancia del modelo, la prioridad de la tarea en que se se ejecuta el código de control de la locomotora.

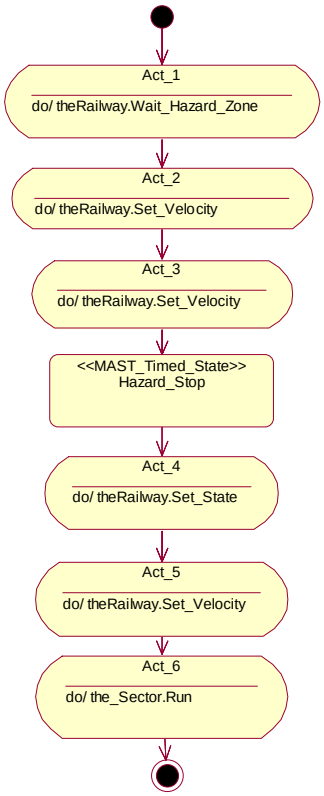
- Referencias:
  - **theRailway :** Railway_RT_Model: Referencia al modelo del objeto de la clase Railway que proporciona los recursos que requiere el movimiento del tren.

- Objetos:

- <<obj>> **theDriver** : Train_Task(thePriority = Train_Priority): Objeto de la clase Train_Task definida en el modelo y que representa un MAST scheduling server con política preemtible, en el que se ejecuta el código de control del tren.
- <<obj>> **theEngine** : MAST_Device: Modela a la locomotora física, capaz de recorrer sucesivamente sectores de acuerdo con su ruta. Es un MAST processing resource con los valores por defecto asignados a sus parametros (son irrelevantes ya que solo ejecuta operaciones de forma secuencial).
- <<obj>> **theLocomotive** : MAST_FP_Sched_Server(host = theEngine). Representa el scheduling server auxiliar para ejecutar las operaciones de recorrer tramos en el processing resource theEngine.

• Modos de usos:

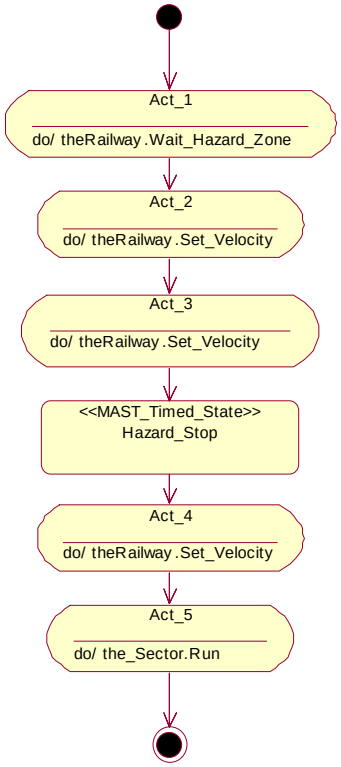
- <<job>> + **Run_Next_Sector**(the_Sector : Sector): Modela el paso de un sector a otro y se modela con el siguiente diagrama de actividad:



En primer lugar el sistema se queda en espera hasta que el tren llega a la zona de peligro. Cuando llega se pasa la velocidad a LOW_SPEED. A continuación , si el siguiente tramo no es accesible , se detiene el tren (velocidad a STOPPED) hasta que sea accesible[1]. Cuando ya sea accesible se coloca el cruce correpondiente en la

posición adecuada , se pasa a velocidad HIGH_SPEED y se comienza a recorrer el siguiente tramo.

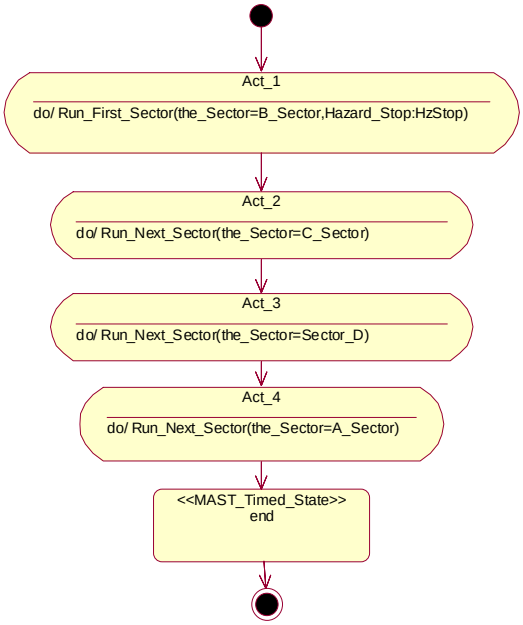
- <<job>> + **Run_First_Sector**(the_Sector : Sector, Hazard_Stop : Timed_State):es un caso particular de la anterior que modela el primer paso de sector (A - B para el recorrido externo o E - B para el recorrido interno). Se ha modelado en otra operación ya que , como veremos posteriormente , vamos a aplicar una restricción temporal en un estado interno de la operación. Como es lógico , su diagrama de actividad es muy similar:



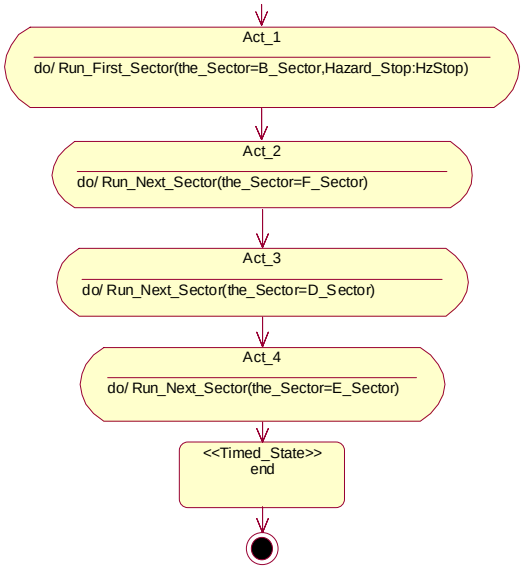
La única diferencia con el anterior es que en esta ocasión no aparece la actividad *Set_State* para fijar el estado del cruce. Al ser el primer paso de sector no es necesario , ya que tanto para el recorrido exterior (tramo A - tramo B) como para el interior (tramo F - tramo B) se pasa por el cruce C3 , cuya posición no es relevante teniendo en cuenta el sentido de la marcha. En otros cruces en los que es necesario actuar sobre el cruce es necesario tomar un recurso del sistema (el semáforo del propio cruce) , y actualmente la herramienta MAST de análisis no permite aplicar restricciones temporales en estos casos. De ahí que se ha modelado la operación *Run_First_Sector* , con el propósito de aplicarle una restricción temporal para el análisis de peor caso.

1. Esta actividad aparece en el modelo ya que deseamos analizar el peor caso.

- <<job>> + **Run_External**(hzStop : Timed_State, end : Timed_State): Modela el recorrido exterior (tramos A - B - C - D) y se describe con el siguiente tramo de actividad:



- <<job>> + **Run_Internal**(hzStop : Timed_State, end : Timed_State): Modela el recorrido interior (tramos E - B - F - D) y se describe con el siguiente tramo de actividad:



IV.5. Modelo de las situaciones de tiempo real de la aplicación

La situación de tiempo real **Railway_Operation** que se ha analizado a través del modelo de tiempo real corresponde a la operación normal de la maqueta con la locomotora BLUE_TRAIN circulando por la ruta externa y con la locomotora RED_TRAIN circulando por la ruta interna. A fin de poder modelar y analizar todos los requerimientos temporal con las herramientas que actualmente se dispone, se considera la condición inicialcuando los dos trenes se encuentran respectivamente en los sectores A y E, y justamente cuando se encuentran entrando en la zona de peligro de acceso al cruce C3.

En la maqueta existen cinco requerimientos de tiempo estricto que deben ser considerados:

_ 1) Se requiere que cada 350 ms se realice una evaluación por visión artificial del estado de la maqueta, y en consecuencia, que se finalice la evaluación antes de que se inicie el siguiente ciclo.

_ 2) Se requiere que antes de que antes de que transcurran 700 ms desde que la locomotora BLUE_TRAIN entra en la zona prohibida del tramo A, la locomotora debe parar, si el tramo B estuviera ocupado.

_ 3) Se requiere que antes de que antes de que transcurran 700 ms desde que la locomotora BLUE_TRAIN entra en la zona prohibida del tramo C, la locomotora debe parar, si el tramo D estuviera ocupado.

_ 4) Se requiere que antes de que antes de que transcurran 700 ms desde que la locomotora RED_TRAIN entra en la zona prohibida del tramo E, la locomotora debe parar, si el tramo B estuviera ocupado.

_ 5) Se requiere que antes de que antes de que transcurran 700 ms desde que la locomotora RED_TRAIN entra en la zona prohibida del tramo F, la locomotora debe parar, si el tramo D estuviera ocupado.

En el modelo de las situaciones de tiempo real de la aplicación que estamos considerando, 1), 2) y 4), pero dada la simetría de la situación, los requerimientos 2) y 3) así como los requerimientos 4) y 5) son equivalentes.

En la situación de tiempo real que se considera, se han introducido dos nuevos requerimientos de tiempo real, que no son propios de la maqueta, sino que solo se introducen a fin de obtener información del análisis. Estos requerimientos son:

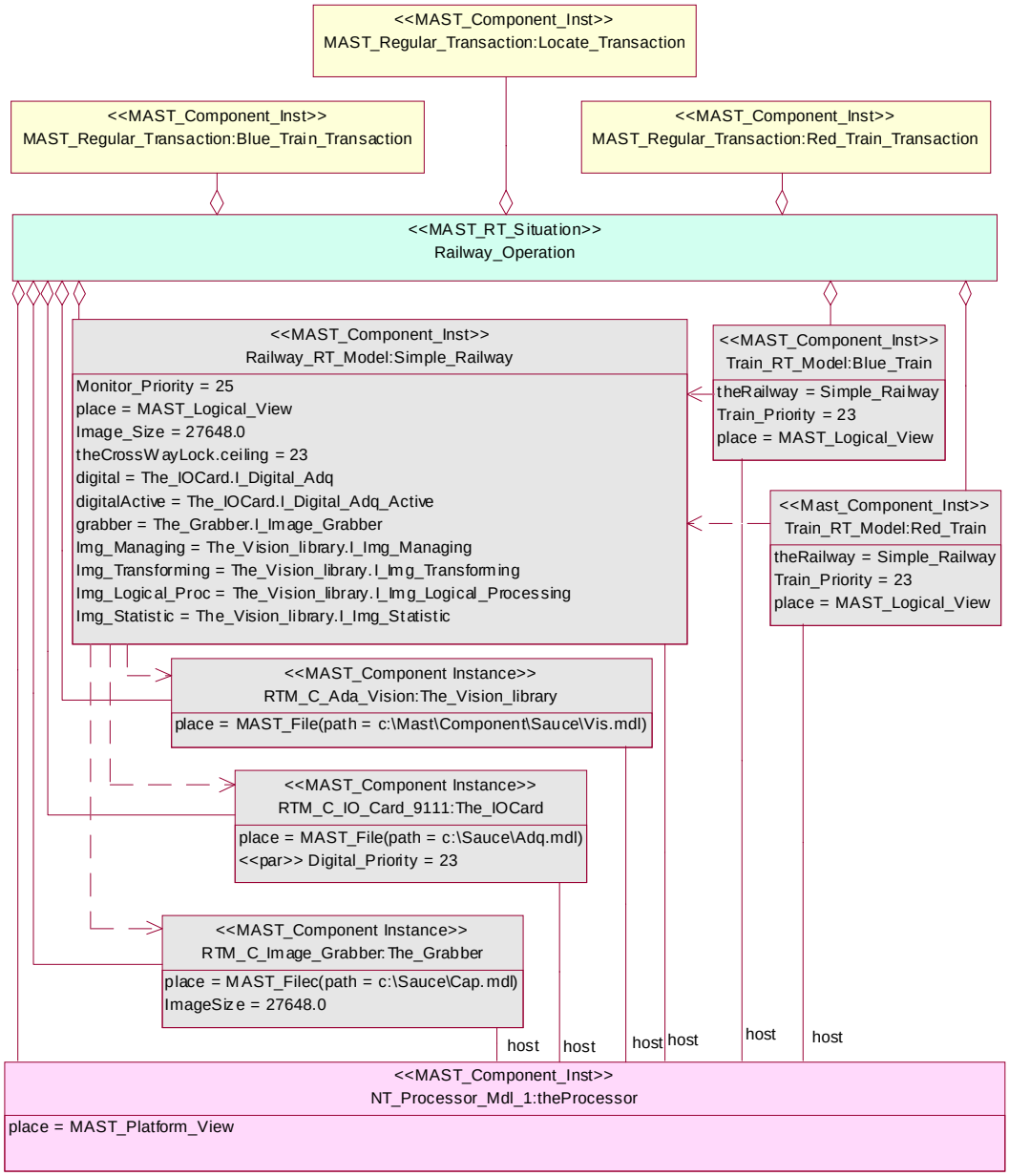
- La locomotora BLUE_TRAIN debe completar una ciclo completo de su ruta antes de que transcurran 17 s.
- La locomotora RED_TRAIN debe completar una ciclo completo de su ruta antes de que transcurran 15 s.

Para describir el modelo de una situación de tiempo real, es necesario describir:

- La **configuración del sistema**: Esto es, la declaración de todas las instancias de componente que participan en ella.
- El **conjunto de transacciones** que concurren en ellas, incluyendo en ellas, los eventos externos que las disparan y los requerimientos de tiempo real que incluyen.

IV.5.1. Configuración de la RT_Situation

En el siguiente diagrama de clases, se muestra los elementos básicos que constituyen la declaración de <<MAST_RT_Situation>> **Railway_Operation**.



En esta RT_Situation se han declarado seis instancias de componentes, y en ellas se han explicitado los valores concretos de los parámetros que se asignan a cada una de las instancias, y las referencias a otras instancias también declaradas en ella que eran requeridas en los correspondientes descriptores.

- <<MAST_Component_Inst>>Railway_RT_Model: **Simple_Railway**: que modela la librería de recursos de control y supervisión de la maqueta que se utiliza. Los valores y referencias establecidas en la instancia son:

Parámetros:

- •host=theProcessor Procesador en el que se instala el componente.
- •Monitor_Priority = 25 Prioridad de la tarea de visión artificial.
- •place = MAST_Logical_View Locatización del descriptor de la instancia.
- •Image_Size = 27648.0 Tamaño de las imágenes que se procesan.
- •theCrossWayLock.ceiling = 23 Techo de prioridad de los semáforos de los cruces.

Referencias:

- •digital = The_IOCard.I_Digital_Adq
- •grabber = The_Grabber.I_Image_Grabber
- •Img_Managing = The_Vision_library.I_Img_Managing
- •Img_Transforming = The_Vision_library.I_Img_Transforming
- •Img_Logical_Proc = The_Vision_library.I_Img_Logical_Processing
- •Img_Statistic = The_Vision_library.I_Img_Statistic

- <<MAST_Componet_Inst>>Train_RT_Model: **Blue_Train**: que modela el software de control de la locomotora Blue. Los valores y referencias establecidas en la instancia son:

Parámetros:

- •host=theProcessor Procesador en el que se instala el componente.
- •Train_Priority = 23 Prioridad de la tarea de control de la locomotora.
- •place = MAST_Logical_View Locatización del descriptor de la instancia.

Referencias:

- • theRailway = Simple_Railway

- <<MAST_Componet_Inst>>Train_RT_Model: **Red_Train**: que modela el software de control de la locomotora Red. Los valores y referencias establecidas en la instancia son:

Parámetros:

- •host=theProcessor Procesador en el que se instala el componente.
- •Train_Priority = 23 Prioridad de la tarea de control de la locomotora.
- •place = MAST_Logical_ViewLocatización del descriptor de la instancia.

Referencias:

- • theRailway = Simple_Railway

- <<MAST_Componet_Inst>>RTM_C_Ada_Vision: **The_Vision_Library**: Modela la instancia del componente C_Ada_Vision, que se utiliza para implementar las interfaces I_Img_Managing, I_Img_Transforming, I_Img_Logical_Processing y I_Img_Statistic, que se requiere en las operaciones de visión artificial Los valores y referencias establecidas en la instancia son:

Parámetros:

- •host=theProcessor Procesador en el que se instala el componente.
- •place = MAST_File(path = c:\Mast\Component\Sauce\Vis.mdl)

- <<MAST_Componet_Inst>>RTM_C_IO_Card_9111: **The_IOCard**: Modela la instancia del componente C_IO_Card_9111, que se utiliza para implementar la interfaz I_Digital_Adq, que se requiere en las operaciones de control del hardware de la maqueta. Los valores y referencias establecidas en la instancia son:

Parámetros:

- •host=theProcessor Procesador en el que se instala el componente.
- •Digital_Priority = 23 Prioridad del semáforo de acceso a los cruces.
- •place = MAST_File(path = c:\Sauce\Adq.mdl).

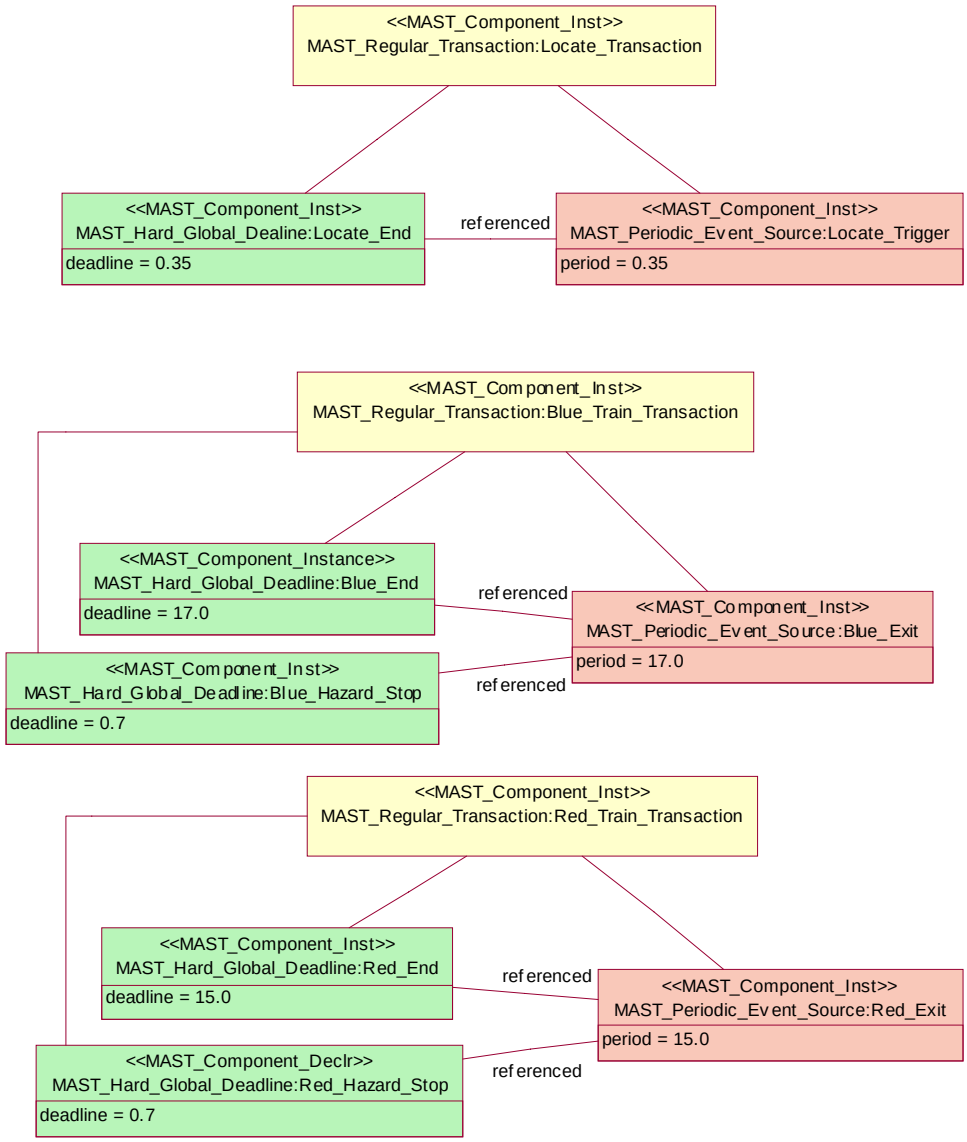
- <<MAST_Componet_Inst>>RTM_C_Image_Grabber: **The_Grabber**:Modela la instancia del componente C_IG_DT3153, que se utiliza para implementar la interfaz I_Image_Grabber, que se requiere en las operaciones de captura y digitalización de las imágenes de vídeo. Los valores y referencias establecidas en la instancia son:

Parámetros:

- •host=theProcessor Procesador en el que se instala el componente.
- •ImageSize = 27648.0 Tamaño de las imágenes que se capturan.
- •place = MAST_Filec(path = c:\Sauce\Cap.mdl)

IV.5.2. Declaración de las transacciones

En el siguiente diagrama de clases, se describen las tres transacciones que concurren en la <<MAST_RT_Situation>>Railway_Operation.

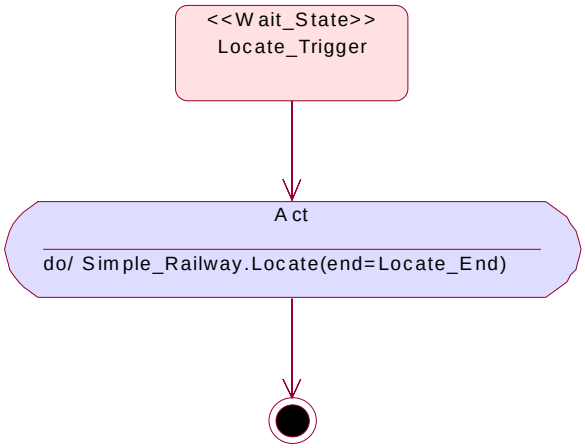


- MAST_Regular_Transaction: **Locate_Transaction:** Corresponde al proceso de localización se realiza periódicamente y que debe acabar dentro del periodo.

  _ Trigger: MAST_Periodic_Event_Source: **Locate_Trigger**: Patrón de generación periódico con periodo de 350 ms.

  _ Requerimiento temporal: MAST_Hard_Global_Dealine : **Locate_End:** Se requiere que debe alcanzarse el estado Locate_End, antes del inicio del siguiente periodo.

_ Actividad: Se describe en el siguiente diagrama de actividad:



- MAST_Regular_Transaction: **Blue_Train_Transaction:** Corresponde al proceso decontrol de la locomotora BLUE_TRAIN a lo largo de su ruta externa.

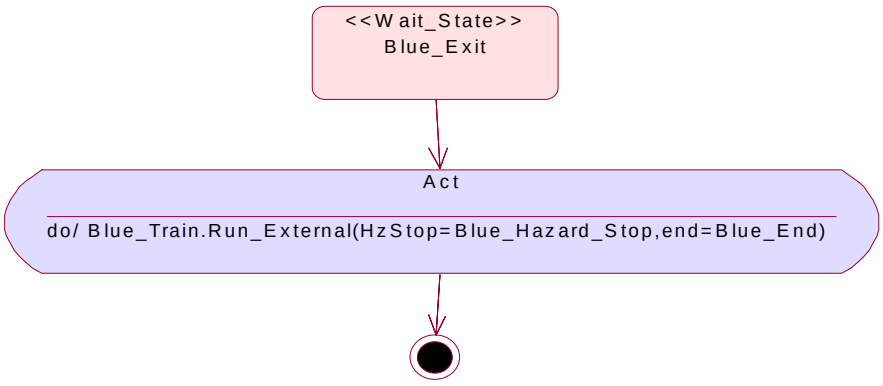
  _ Trigger: MAST_Periodic_Event_Source: **Blue_Exit**: Patrón de generación periódico con periodo de 17 s, que corresponde con el inicio de cada vuelta de la locomotora BLUE_TRAIN.

  _ Requerimientos temporales:

    _ MAST_Hard_Global_Dealine: **Blue_Hazard_Stop:** Se requiere que el programa de control pare antes de 700 ms el tren si el sector B está ocupado.

    _ MAST_Hard_Global_Deadline: **Blue_End:** Se requiere que la locomotora de una vuelta completa antes de 17 s.

  _ Actividad: Se describe en el siguiente diagrama de actividad:



- MAST_Regular_Transaction: **Red_Train_Transaction:** Corresponde al proceso decontrol de la locomotora RED_TRAIN a lo largo de su ruta interna.

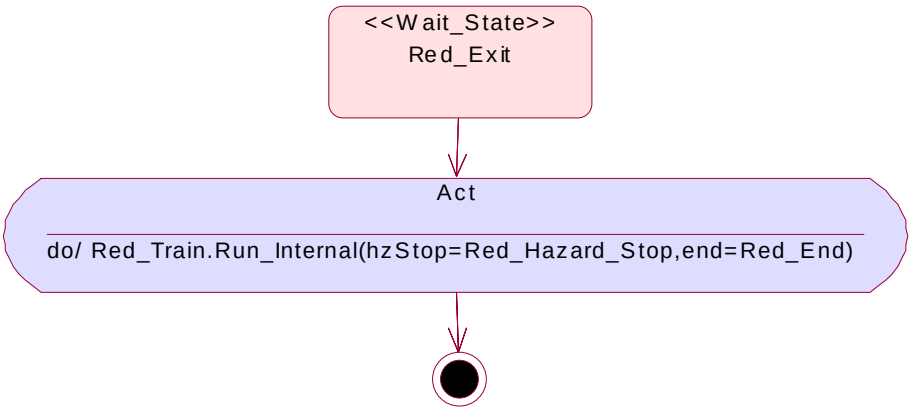
  _ Trigger: MAST_Periodic_Event_Source: **Red_Exit**: Patrón de generación periódico con periodo de 15 s, que corresponde con el inicio de cada vuelta de la locomotora RED_TRAIN.

  _ Requerimientos temporales:

    _ MAST_Hard_Global_Dealine: **Red_Hazard_Stop:** Se requiere que el programa de control pare antes de 700 ms el tren si el sector B está ocupado.

    _ MAST_Hard_Global_Deadline: **Red_End:** Se requiere que la locomotora de una vuelta completa antes de 15 s.

  _ Actividad:Se describe en el siguiente diagrama de actividad.



IV.6. Análisis de tiempo real de la aplicación

Una vez definido el modelo completo de la RT_Situation, se ha generado un fichero MAST_File que describe el modelo en un formato interpretable con las herramientas de análisis que nos va a analizar si el sistema es planificable y si se pueden cumplir las restricciones fijadas.

En un futuro sería interesante automatizar la generación de este fichero a partir del modelo UML, pero en al no existir actualmente la herramienta que compile el modelo CBSE_Mast al fichero MAST_File, se ha hecho la compilación manualmente.

Los resultados más relevantes que se han obtenido son:

La restricción de tiempo de la localización (350 mseg.) se cumple sobradamente , ya que los tiempos resultantes del análisis son:

WCET = 318.9 mseg.

BCET = 162.8 mseg.

Esto quiere decir que aún podríamos haber bajado más el periodo de la tarea de localización. De hecho , en la práctica el sistema funciona para valores del periodo inferiores a 300 mseg. Esto es debido a que el tiempo medio es bastante menor que el WCET. Sin embargo , no tenemos la seguridad de que el sistema funcione en el 100 % de las ocasiones. Lo que está claro es que si fijamos el periodo por debajo de BCET el sistema ni siquiera llega a poner en marcha a los trenes , ya que se ejecuta continuamente la tarea de localización (más prioritaria) y nunca llegan a tener el control de la CPU las tareas de los trenes.

La restricción de tiempo de detención (700 mseg.) es idéntica para ambos trenes , y se cumple sobradamente. Los resultados obtenidos son:

WCET = 583.6 mseg.

BCET = 0.17 mseg.

Se cumple sobradamennte para el peor caso (WCET).

La restricción de tiempo de vuelta para el tren azul (17 seg.) también se cumple , ya que los resultados son:

WCET = 16.74 seg.

BCET = 11.20 seg.

Como era de esperar , el tiempo de mejor caso es ligeramente superior a la suma de los tiempos de los tramos que recorre[1].

La restricción de tiempo de vuelta para el tren rojo (15 seg.) también se cumple , ya que los resultados son:

WCET = 13.84 seg.

BCET = 8.30 seg.

Como era de esperar , el tiempo de mejor caso es ligeramente superior a la suma de los tiempos de los tramos que recorre[2].

1. Ver modelo del Railway : 3.9 + 1.25 + 3.9 + 1.25 = 10.3 seg.
2. Ver modelo del Railway : 2.45 + 1.25 + 2.45 + 1.25 = 7.4 seg.

V. CONCLUSIONES

Se han cubierto practicamente en su totalidad los objetivo del proyecto planteados en el apartado I.7. , y durante el desarrollo de los mismos se han obtenido interesantes conclusiones.

Se ha comprobado la eficiencia del lenguaje de programación Ada 95 para para la implementación de componentes software. A pesar de tratarse de un lenguaje no demasiado utilizado en aplicaciones de propósito general , si está bastante implantado en el campo de las aplicaciones de tiempo real , lo que hace que sea especialmente interesante desarrollar la metodología de desarrollo basado en componentes utilizando dicho lenguaje en este campo concreto.

Ha quedado de manifiesto la validez del perfil CBSE_MAST para el modelado de componentes software de tiempo real , incluso si tenemos la necesidad de modelar operaciones relaivamente complejas (como por ejemplo la operación Acquire del componente C_IG_DT3153).

Tras el desarrollo del presente trabajo se plantean varias líneas abiertas de trabajo para un futuro próximo:

- Se han de desarrollar herramientas que generen automáticamente el código Ada 95 de las interfaces y los componentes a partir de sus respectivas descripciones UML (apartado II.4.).
- Se han de desarrollar herramientas que generen automáticamente el código MAST-File de los componentes (apartados III.5 y III.6.).
- Se ha de plantear la posibilidad de que los componentes ofrezcan procedimientos para recalcular los valores de los parámetros de su modelo de tiempo real al cambiar de una plataforma a otra.
- Se han de evaluar metodologías estadísticas que nos permitan estimar con precisión las medidas temporales (BCET , ACET y WCET) de los modelos de tiempo real de los componentes.

VI. REFERENCIAS

[1] Szyperski C.: “Component Software: Beyond Object-Oriented Programming” Addison-Wesley, 1999.

[2] Medina J.L. y Drake J.M.: “Modelado y Análisis de Tiempo Real de Aplicaciones Basadas en Componentes” V Jornadas de Tiempo Real. Cartagena , 2002.

[3] Cameron A.: “Designing Component Kits and Archiquctures with Catalysis” TriReme International Ltd. http://www.trireme.com

[4] Kirtland M.: “Object-Oriented Software Development Made Simple with COM+ Runtime Services” Noviembre 1997. Microsoft Systems Journal at http://www.microsoft.com/com, “White Papers”.

[5] Schmidt D.: “Overview of CORBA” at http://cgi.omg.org/corba/beginners.html

[6] Baker S.: “CORBA Distributed Objects” Addison-Wesley 1997.

[7] Drake J.M. , Medina J.L. y González M.: “Entorno para el diseño de sistemas basados en componentes de tiempo real” X Jornadas de Concurrencia , Jaca 2002.

[8] Cheesman J. y Daniels J.: “UML Components: A simple Process for Specifying Component-Based Software”, Addison-Wesley,2000.

[9] D’Souza D.F. y Cameron A.: “Objects, Components and Frame-works in UML: the Catalysis approach”. Addison-Wesley, 1998.

[10] Hassan Gomaa: “Designing Concurrent, Distributed and Real-Time Application with UML”. Addison-Wesley, 2000.

[11] Isovic D. y Nostrom C.: “Components in Real-Time Systems” RTCSA’2002 Tokyo.

[12] Lüders F. y Crnkovic I.: “Experiences with Component-Based Sofware Development in Industrial Control” Mälardalen Real-Time Research Center , 2001.

[13] González M., Gutiérrez J.J., Palencia J.C. y Drake J.M.: "MAST: Modeling and Analysis Suite for Real-Time Applications" Proceedings of the Euromicro Conference on Real-Time Systems, June 2001.

[14] Medina J.L., González M. y Drake J.M.:" "MAST Real-time View: A Graphic UML Tool for Modeling Object_Oriented Real_Time Systems", RTSS, 2001, December, 2001.

[15] Medina J.L., Gutierrez J.J., González M. y Drake J.M.:"Modeling ans Schedulability Analysis of Hard Real-Time Distributed Systems based on ADA Componentes." Ada Europe, 2002

[16] Kobryn C.: “Modeling Components and frameworks with UML.” Communications of the ACM , Octubre 2000/Vol. 43 , Nº 10.

[17] Medina J.L.: “Metodología y herramientas UML para el modelado y análisis de sistemas orientados a objetos de tiempo real.” Tesis Doctoral (en preparación). Universidad de Cantabria

[18] ------: ”Intel Image Processing Library: Reference Manual” Intel Corporation,1998. Order Num.:663791-002. http://developer.intel.com

[19] ------: ”Open Source Computer Vision Library: Reference Manual” Intel Corporation, 2000, Order Num.:TBD. http://developer.intel.com

[20] Timmerman M.: “Windows NT as Real-Time OS?” Real-Time Magazine , Febrero 1997.

[21] Solomon D.: “Inside Windows NT” Second Edition. Microsoft Press , 1998.

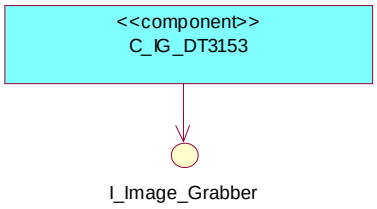
[22] Drake J.M., Medina J.L.: “UML MAST Metamodel 1.0” Grupo Computadores y Tiempo Real , Universidad de Cantabria , Febrero 2001.

[23] Drake J.M.: “CBSE_MAST_Concepts”Grupo Computadores y Tiempo Real , Universidad de Cantabria , Octubre 2002.

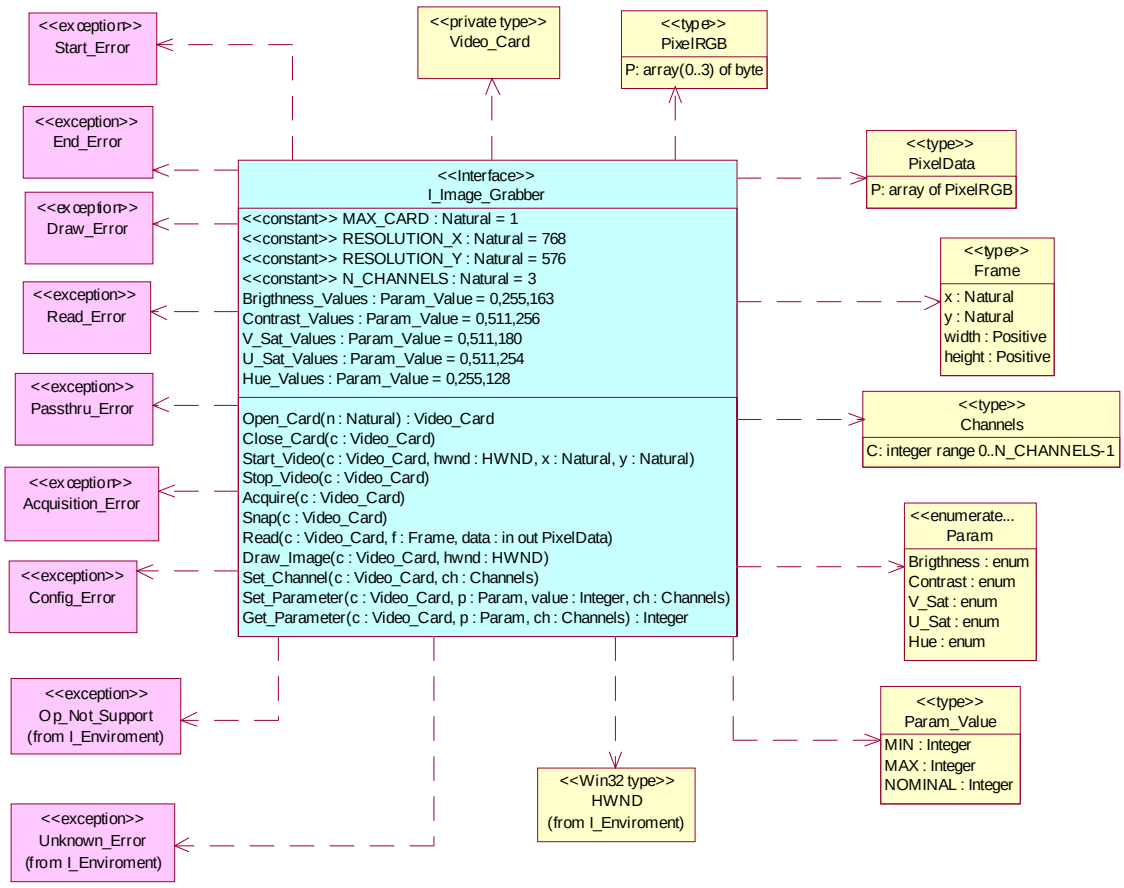
Apéndice A: ESPECIFICACIÓN DE LOS COMPONENTES

A.1.Componente C_IG_DT3153

Especificación funcional

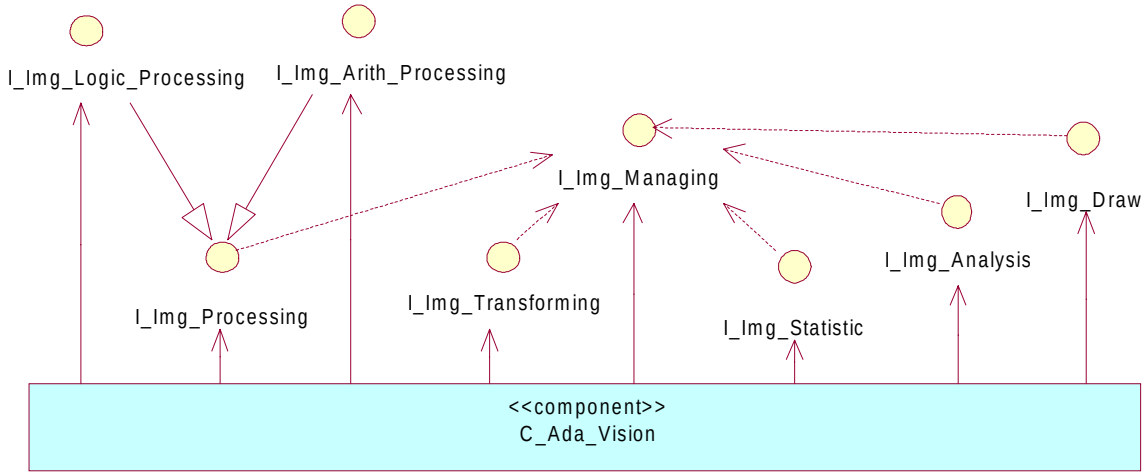


Interface I_Image_Grabber:

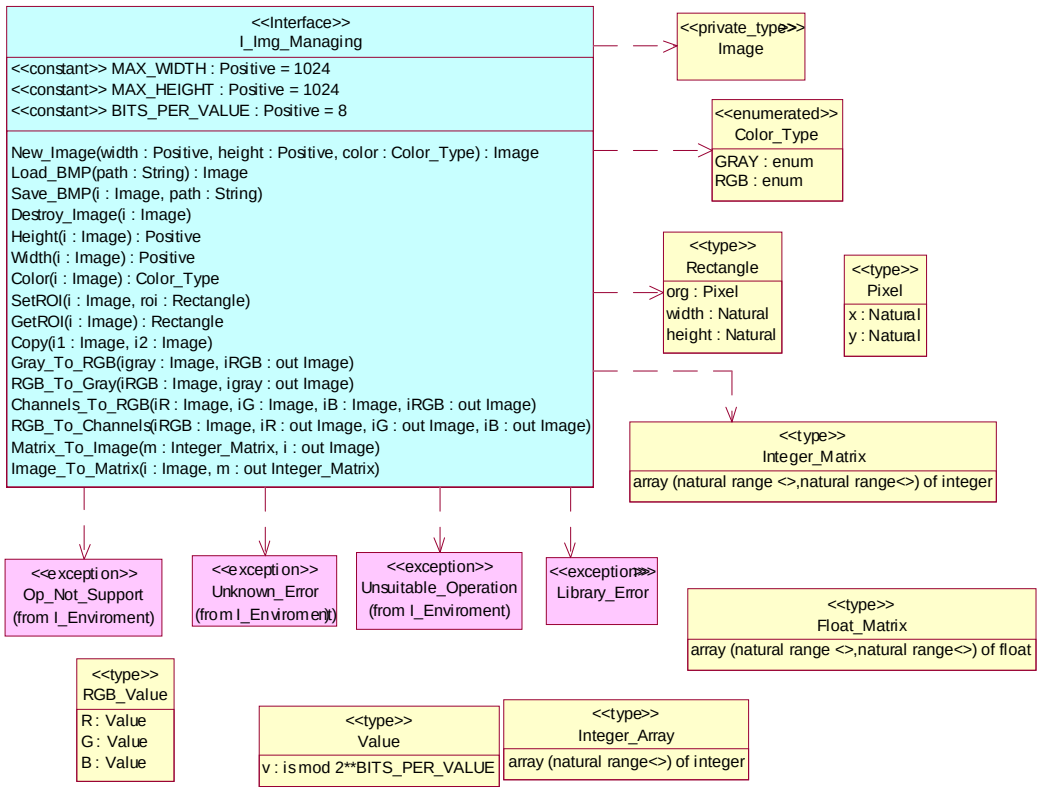


A.2.Componente C_Ada_Vision

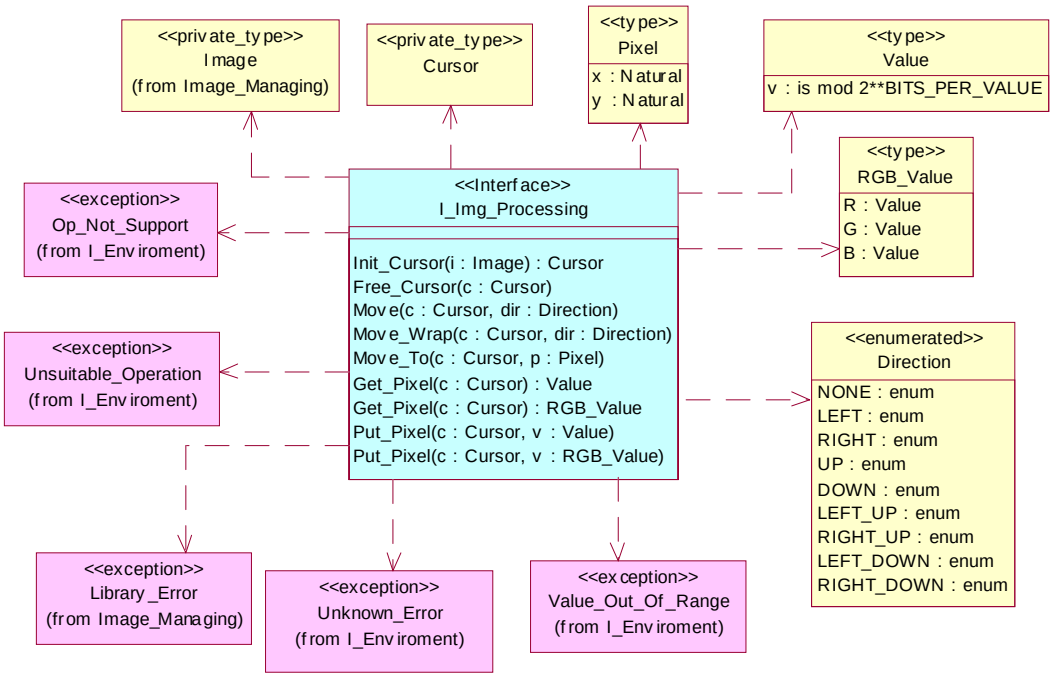
Especificación funcional.



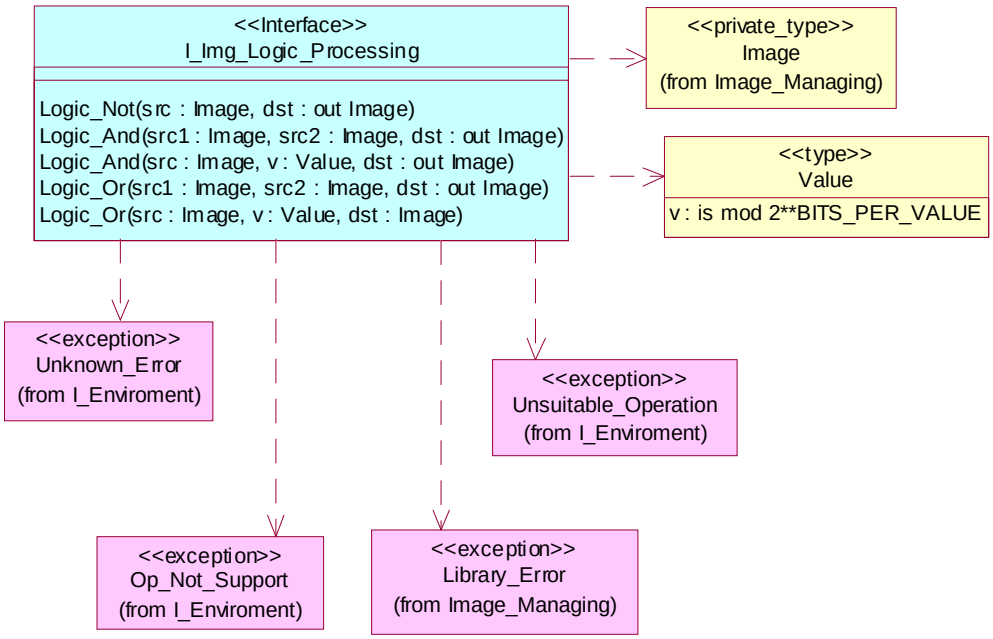
Interface I_Img_Managing:



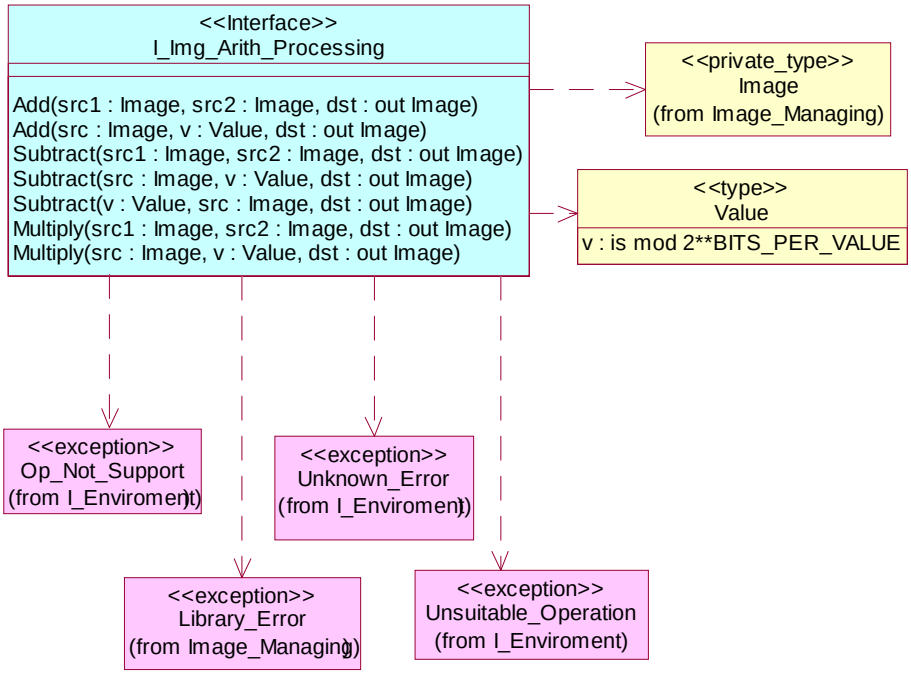
Interface I_Img_Processing:



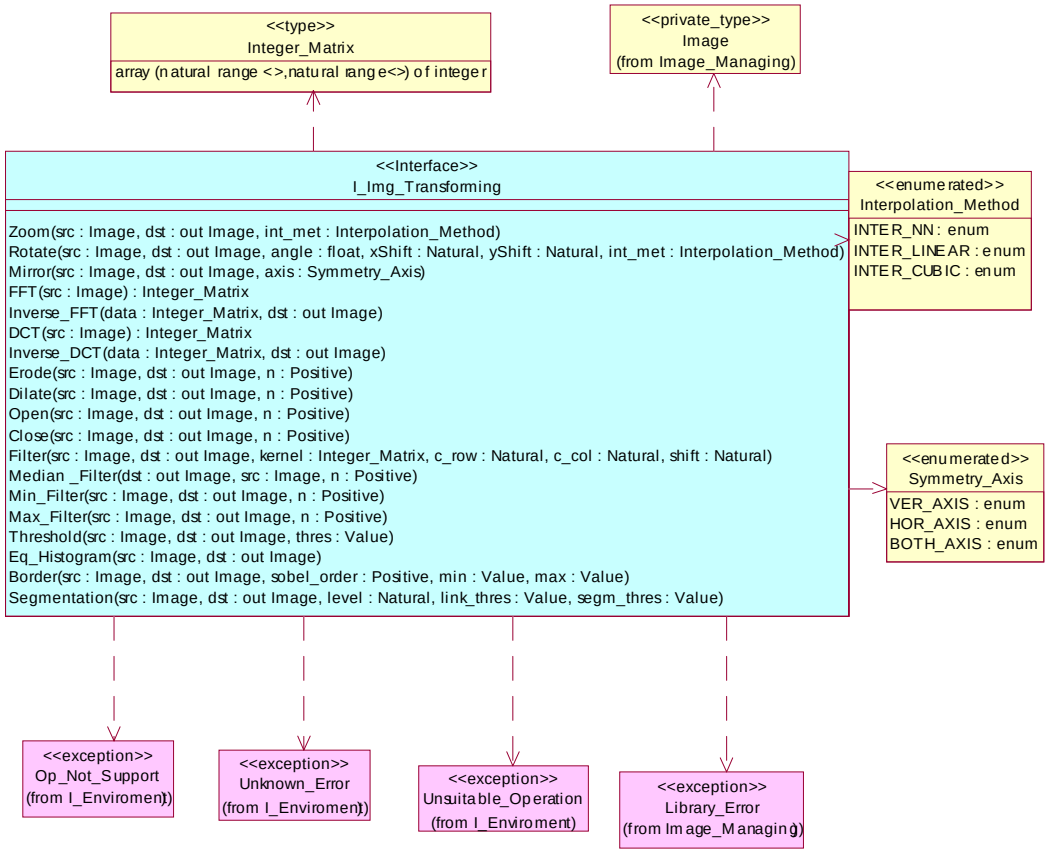
Interface I_Img_Logic_Processing:



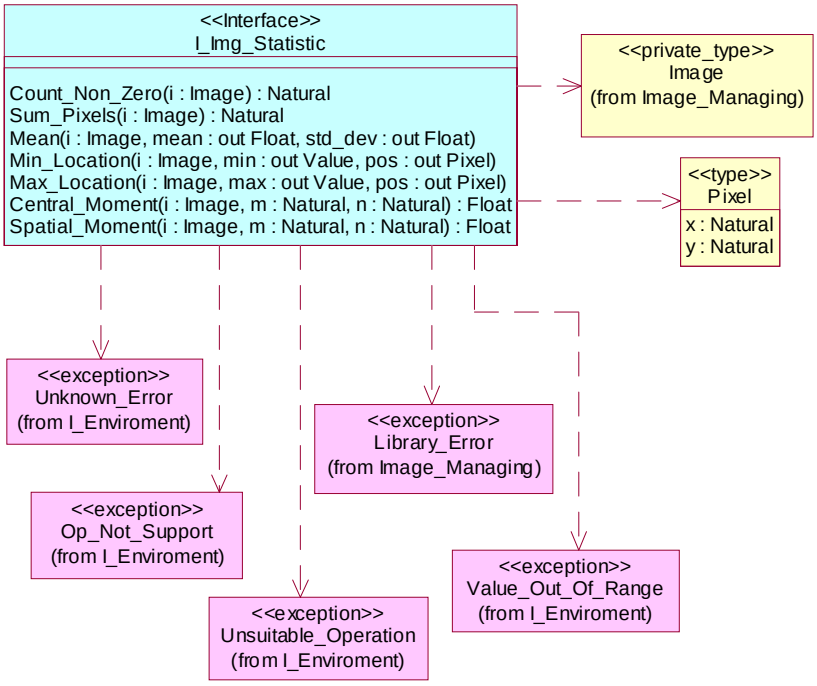
Interface I_Img_Arith_Processing:



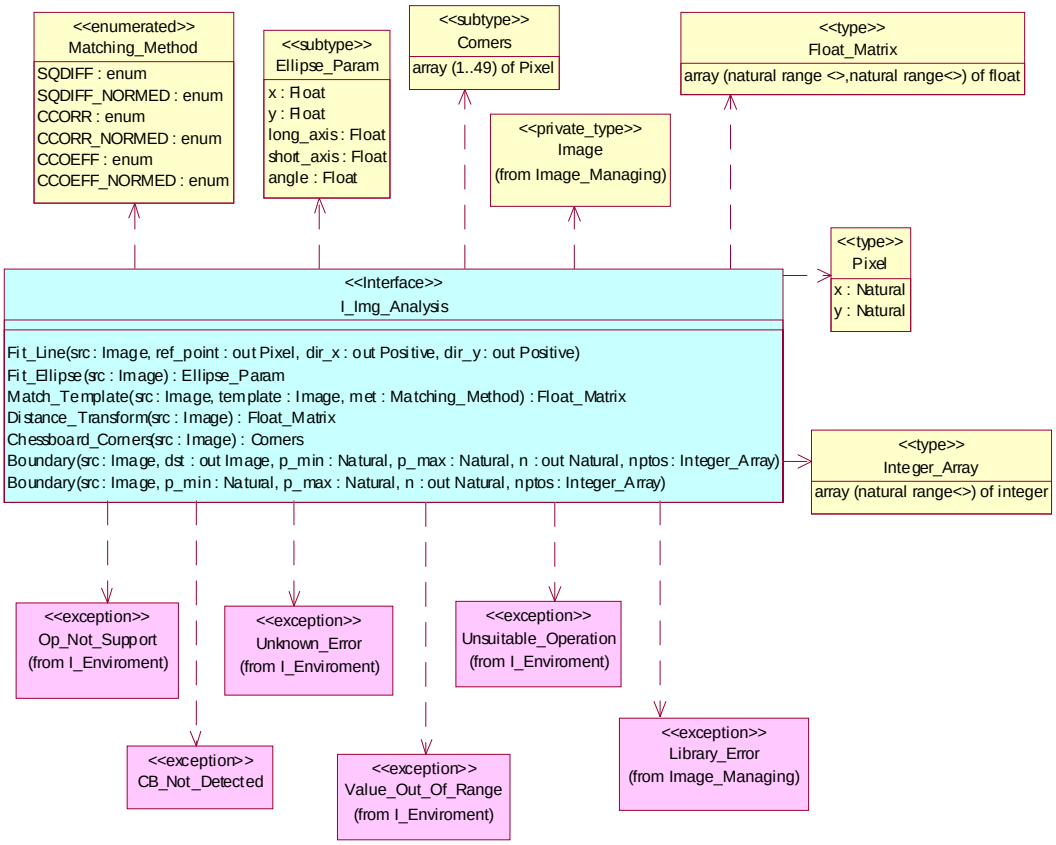
Interface I_Img_Transforming:



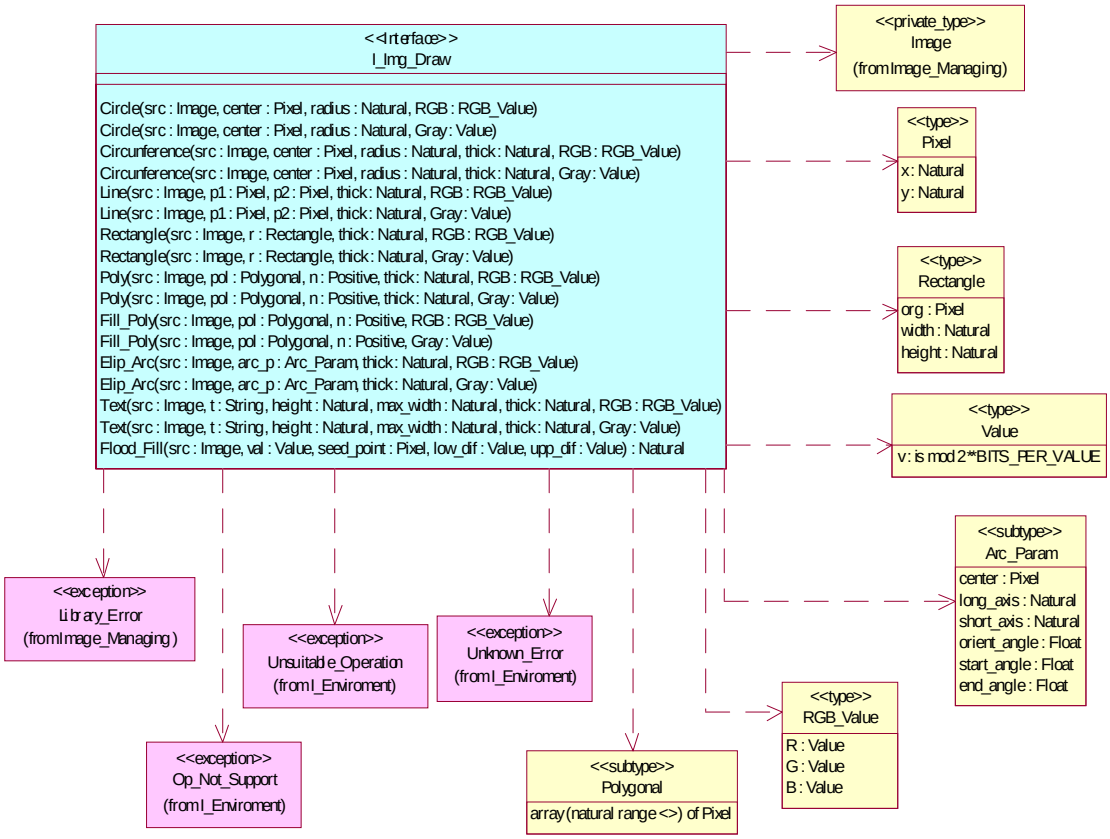
Interface I_Img_Statistic:



Interface I_Img_Analysis:

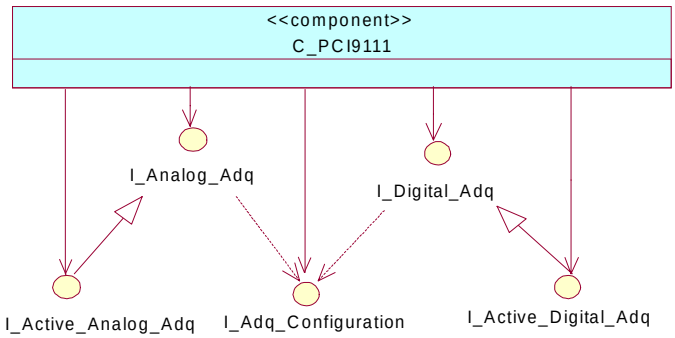


Interface I_Img_Draw:

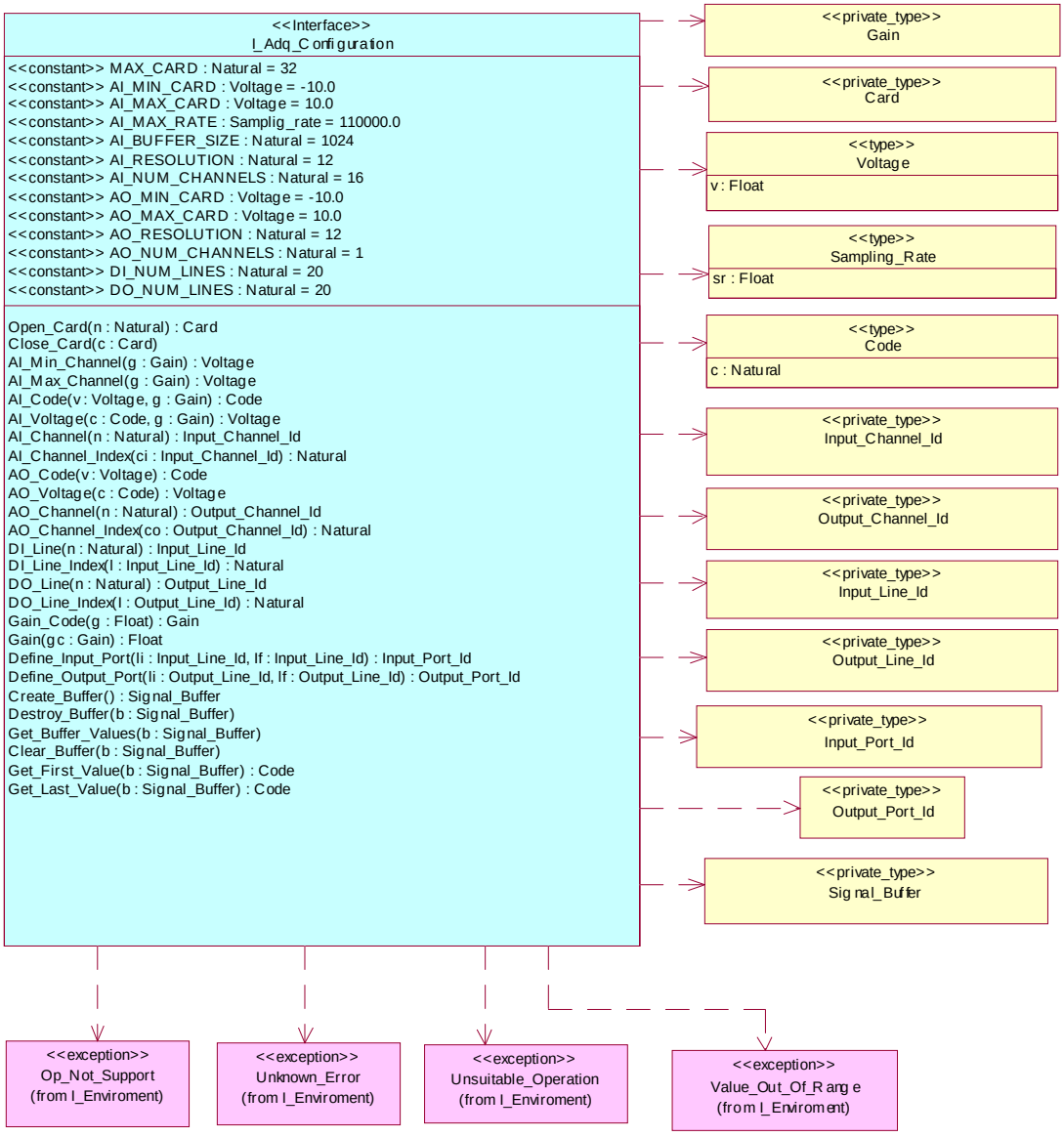


A.3.Componente C_PCI9111

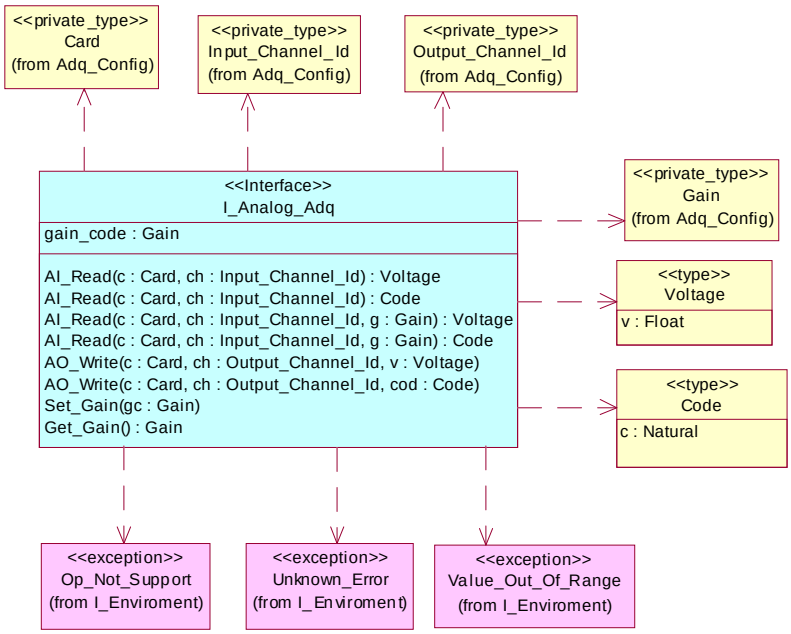
Especificación funcional.



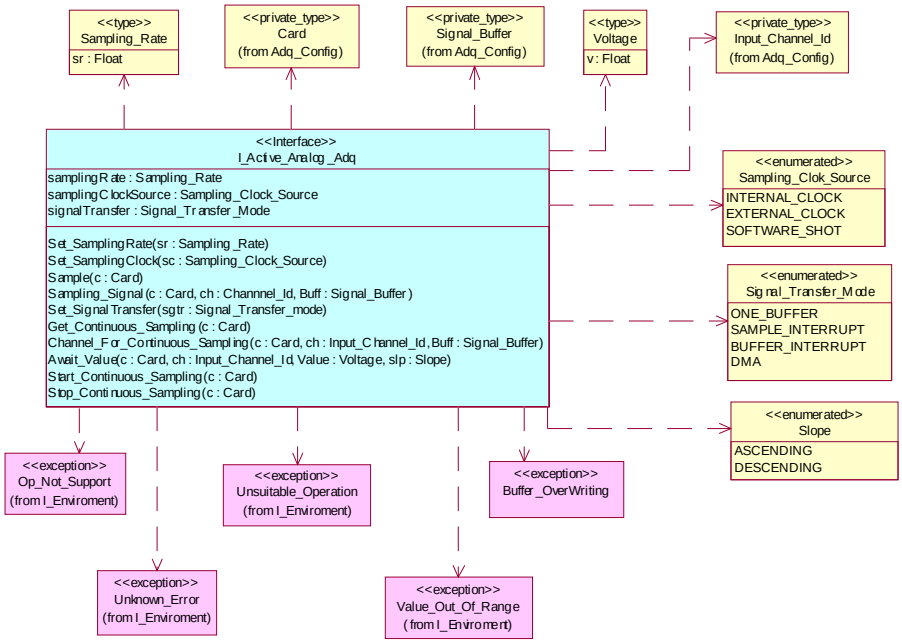
Interface I_Adq_Configuration:



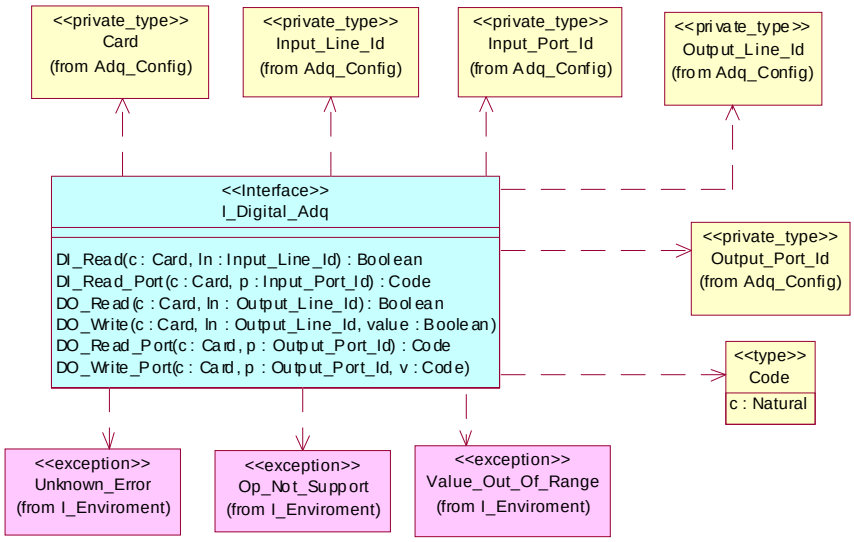
Interface I_Analog_Adq:



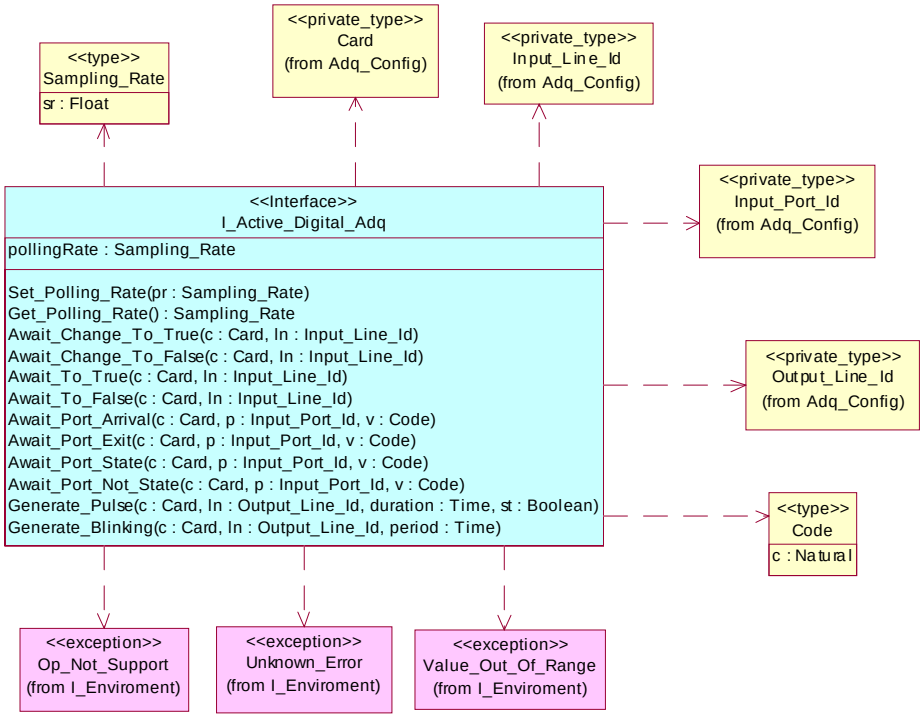
Interface I_Active_Analog_Adq:



Interface I_Digital_Adq:



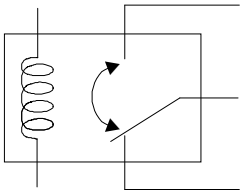
Interface I_Active_Digital_Adq:.



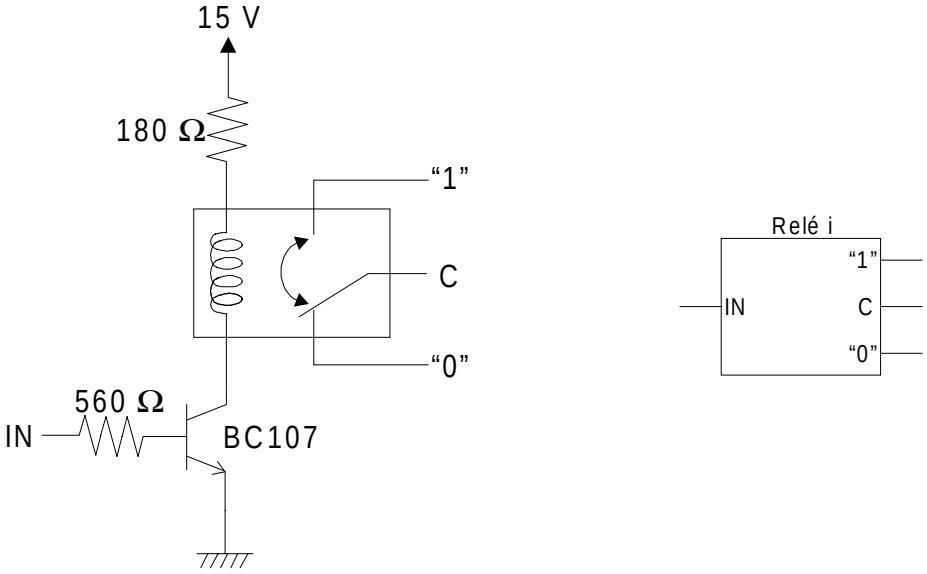
Apéndice B: DISEÑO Y CABLEADO DE LA TARJETA INTERMEDIA DE RELÉS.

Se ha diseñado y construido una tarjeta intermedia de 16 relés controlados por entradas TTL , necesario para el control de la maqueta desde las salidas digitales de la tarjeta de I/O 9111-DG de la empresa Adlink utilizada en la implementación del componente C_PCI9111.

El elemento básico del diseño es un relé de tres terminales como el que se muestra en la figura. Cuando la tensión en la bobina de control es de 0 voltios el terminal común se cortocircuita con el terminal inferior y el superior queda en alta impedancia. Cuando la tensión en la bobina de control es de 12 voltios el terminal común se cortocircuita con el terminal superior y el inferior queda en alta impedancia. En este último caso la bobina consume una intensidad de 16.2 mA[1].



La tarjeta de interfase consiste en la réplica (16 veces) del siguiente circuito que permite controlar los relés con niveles TTL:



Cuando en el terminal IN tenemos un "0" lógico TTL el terminal C está cortocircuitado con el terminal "0". Cuando en el terminal IN tenemos un "1" lógico TTL el terminal C está cortocircuitado con el terminal "1".

El cableado de la maqueta es acesible a través de un conector DB-25 hembra , donde los distintos pines están conectados del modo que se indica en la tabla de la página siguiente.

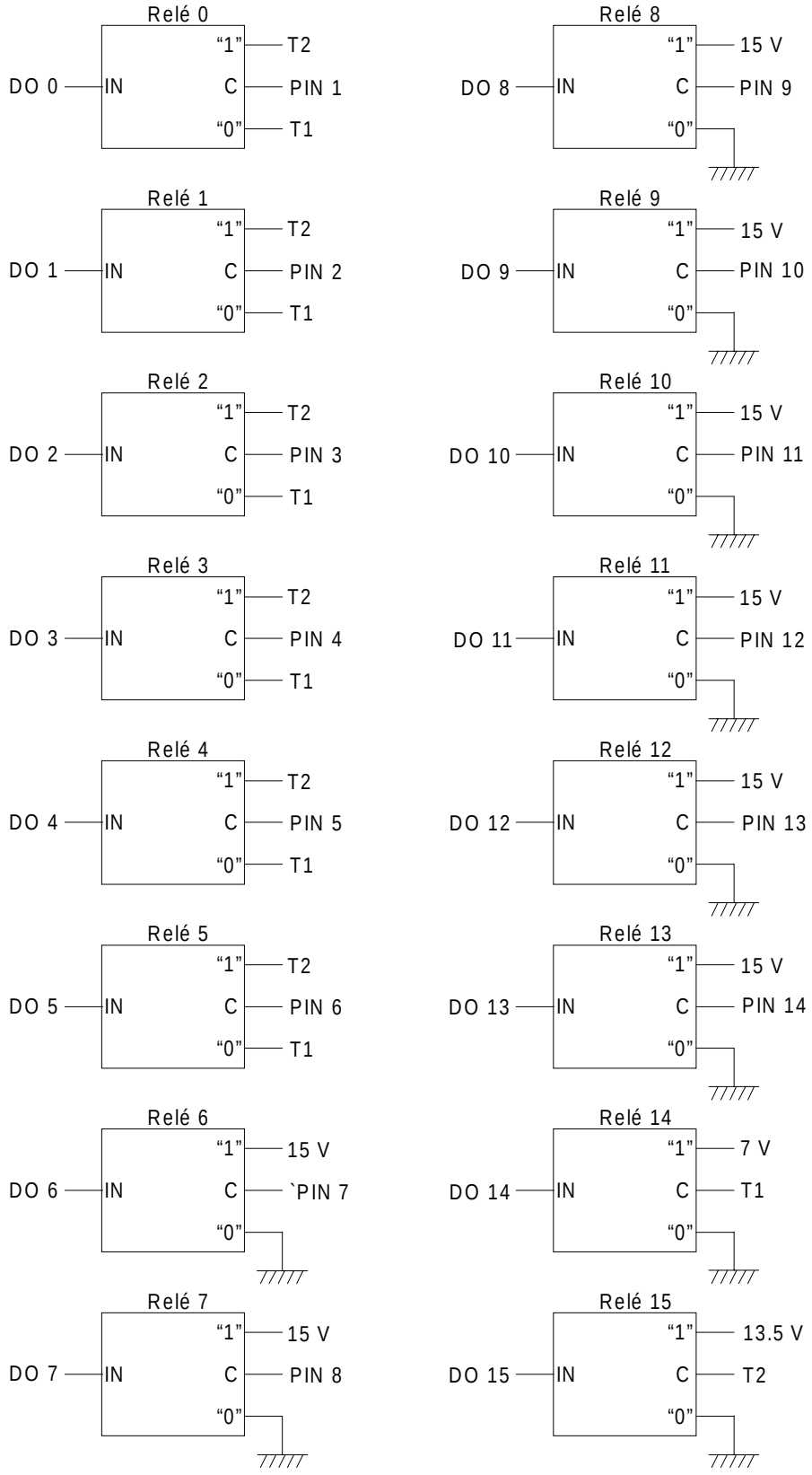
Los pines 1 al 6 están conectados a los polos positivos de los seis tramos. Los polos negativos están conectados a 0 voltios (GND / pin 25).

1. La bobina tiene una impedancia cuya parte real es de 740 Ω.

Los pines 7 a 14 están conectados a los terminales de control de los cuatro cruces. Los terminales “común” de los cruces están conectados a 0 voltios (GND / pin 25).

Pin DB-25	Conexión
1	Tramo A
2	Tramo B
3	Tramo C
4	Tramo D
5	Tramo E
6	Tramo F
7	terminal RECTO C2
8	terminal DESVIO C2
9	terminal RECTO C1
10	terminal DESVIO C1
11	terminal RECTO C4
12	terminal DESVIO C4
13	terminal RECTO C3
14	terminal DESVIO C3
25	GND

El cableado realizado entre la tarjeta intermedia de relés y la maqueta por un lado y las salidas digitales de la tarjeta I/O por otro es:



En el relé 15 se determina si T1 es 0 o 7 voltios (LOW_SPEED).

En el relé 14 se decide si T2 es 0 o 13.5 voltios (HIGH_SPEED).

En los relés del 0 al 5 se decide si cada uno de los tramos se alimenta con T1 o con T2.

En los relés 6 al 13 se controla la posición de los cuatro cruces generando pulsos digitales en el terminal de la posición deseada.