XXIV JORNADAS DE AUTOMATICA

León 10, 11 y 12 de septiembre de 2003

Universidad de León

E.II. INDUSTRIAL E INFORMATICA

Organizan:

UNIVERSIDAD DE LEON

CEA IFAC comité español de automática

Colaboran:

MINISTERIO DE CIENCIA Y TECNOLOGIA

JUNTA DE CASTILLA Y LEON

AYUNTAMIENTO DE LEON

DIPUTACION DE LEON

ISBN:84-931846-7-6
Dep. Legal: LE-999-2003

METODOLOGÍA DE DESARROLLO DE APLICACIONES BASADAS EN COMPONENTES PARA AUTOMATIZACIÓN INDUSTRIAL

Patricia López Martínez, Pedro Espeso, Julio Luis Medina y José M. Drake
{lopezpa,medinajl,drakej}@unican.es
Grupo de Computadores y Tiempo Real (www.ctr.unican.es).
UNIVERSIDAD DE CANTABRIA

Resumen

Se describe una metodología de diseño de aplicaciones de tiempo real estricto basada en componentes software. A las características de componibilidad y robustez que son la base de la tecnología de componentes, en el caso de tiempo real hay que añadir también la característica de predictibilidad. Para ello, tanto el contrato de uso que describe los servicios que ofrece, como el contrato de instalación que especifica el entorno y los servicios que se requieren para que pueda operar un componente de tiempo real, deben incorporar un modelo específico que describa las características temporales de las respuestas que ofrece y las de los recursos que para ello se necesitan. En esta comunicación se presentan los conceptos básicos de la metodología que en esta dirección se está desarrollando, así como su aplicación al desarrollo de una familia de componentes destinados al dominio de la automatización industrial.

Palabras Clave: CBSE, Software Component, Real-Time, Schedulability.

1 COMPONENTES SOFTWARE Y APLICACIONES DE TIEMPO REAL.

Al igual que otras ingenierías, la ingeniería software trata de aplicar la tecnología de componentes a fin de reducir los costos y plazos de desarrollo y abordar la creciente complejidad de las aplicaciones informáticas. Mientras que en muchos dominios de aplicación, como en multimedia, ofimática, o interfaces gráficas, etc., la tecnología de componentes está plenamente consolidada, en otros, como es el caso de los sistemas de tiempo real su aplicación presenta problemas que no están aún resueltos y en consecuencia su uso no es habitual. Sin embargo, la tecnología de componentes en sistemas de tiempo real está siendo requerida por la industria de automatización, de potencia y equipamientos eléctricos, aviación, automoción, etc. [2],[6],[7]y[10]. Empresas líderes en estos campos como ABB o Boeing hacen grandes inversiones en su desarrollo, obteniendo resultados prometedores en cuanto al mantenimiento y gestión de la evolución de sus productos, pero basándose hasta ahora en soluciones propias que no han llegado a imponerse con carácter general.

El diseño de componentes de tiempo real es más complejo que el diseño de componentes que tienen sólo requerimientos funcionales. En primer lugar, los requisitos de temporización implican especificar las capacidades de colaboración y sincronización entre componentes a un nivel más bajo del que proporcionan las interfaces. En segundo lugar, las plataformas habituales de los sistemas de tiempo real son heterogéneas y proporcionan tan sólo recursos limitados, por lo que los entornos estándar de componentes (EJB, .NET, CCM, etc.) no están soportados. Y en tercer lugar, para que un componente pueda ofrecer prestaciones de tiempo real, debe estar soportado por sistemas operativos, sistemas de comunicación, bases de datos, etc. que ofrezcan servicios específicos de gestión de tiempo, de sincronización y de planificación predecibles. Sin embargo, está actualmente admitido [5],[11] que la principal dificultad para la implantación de la tecnología de componentes de tiempo real surge de la falta de estrategias y experiencia de cómo implementar la tecnología de componentes en los entornos de tiempo real.

Nuestro grupo está trabajando en una línea de investigación [3],[9] cuyo objetivo es el desarrollo de una metodología de modelado, diseño y análisis de componentes de tiempo real y se basa en dos antecedentes ya resueltos. En primer lugar, se ha desarrollado un núcleo de sistema operativo de tiempo real que satisface el estándar POSIX.13 que define el perfil de sistemas de tiempo real mínimos, y así mismo, se han propuesto extensiones a este perfil que permiten adaptarlo a nuevos tipos de requerimientos (http://marte.unican.es). Las

plataformas basadas en el estándar POSIX y en sus extensiones de tiempo real son candidatas razonables para una tecnología de componentes de tiempo real. En segundo lugar, se ha desarrollado el entorno MAST (http://mast.unican.es) de modelado, diseño y análisis de tiempo real, que se basa en el modelado independiente de la plataforma y de la lógica, y que en consecuencia es especialmente adecuado como metodología de modelado y análisis de sistemas basados en componentes.

Es esta comunicación, se presentan los aspectos básicos de la metodología de componentes de tiempo real que se está desarrollando, así como algunas experiencias de interés en el desarrollo de algunos componentes de tiempo real.

2 INFORMACION ASOCIADA A UN COMPONENTE DE TIEMPO REAL.

Un componente es un módulo software o software/hardware reusable que ha sido definido, diseñado y realizado en función de un dominio de aplicación y con independencia de las aplicaciones que lo utilizan.

2.1 CONTRATOS DE USO Y DE INSTANCIACIÓN.

Con el término "componente" se designan diferentes conceptos que corresponden a diferentes entes, que en cada tecnología se describen o realizan a su vez con especificaciones o códigos diferentes. En la figura 1, se relacionan estos conceptos.

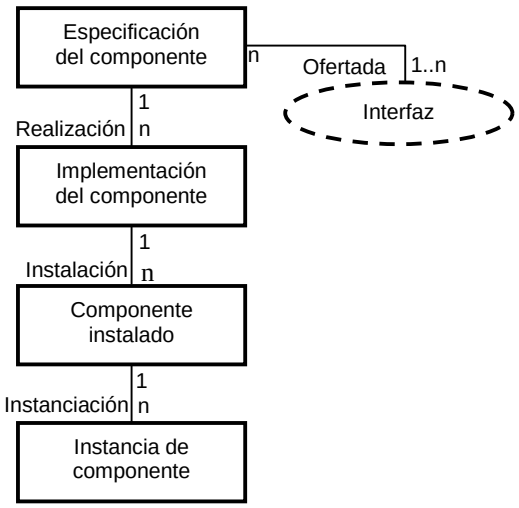

Figura 1: Formas de un componente

La "especificación de un componente" es un concepto de diseño y especificación, consistente en un descriptor de los servicios que el componente ofrece (contrato de uso) y de los servicios que requiere del entorno o de otros componentes de soporte para que el componente pueda operar (contrato de instalación). Contiene toda la información que se necesita para decidir su utilidad, instalarlo y hacer uso de él. La especificación de un componente se define con independencia de cualquier tecnología o lenguaje de programación y en nuestro caso se formula a través de un conjunto de modelos UML.

La "interfaz" es el recurso básico de inter-conectividad funcional entre componentes. Describe e implementa de forma autocontenida un servicio que es ofertado o requerido por un componente. Las interfaces se definen como recurso para modelar, describir y gestionar un dominio de aplicación. Para conseguir reusabilidad, deben ser definidas antes de iniciar el diseño de los componentes y deben concebirse con independencia de ellos. Las interfaces constituyen la base de la componibilidad en la metodología de componentes (si una aplicación o un componente requiere un servicio que corresponde a una determinada interfaz, éste se puede suplir con cualquier componente que ofrezca esa interfaz).

La especificación de una interfaz debe contener:

- El identificador de la interfaz, por el que se la referencia en cualquier componente que la ofrezca o la requiera.
- El modelo de información, esto es, los atributos, tipos, asociaciones, etc. definidos unos en función de otros hasta proporcionar un modelo completo y cerrado que defina el servicio.
- La descripción de las operaciones a través de las 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).
- El 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.

En nuestro caso, la interfaz se formula mediante un modelo UML. El descriptor interfaz se representa como una clase con el estereotipo <<Interface>> que se describe junto al modelo de información que se necesita para formular tanto sus estados como los servicios que ofrece. Este modelo de información se describe utilizando todos los recursos que ofrece UML.

Como se muestra en la figura 2, el modelo de una interfaz se encapsula como un paquete con el estereotipo <<interface_model>>, que tiene como identificador el identificador de la interfaz, y que se

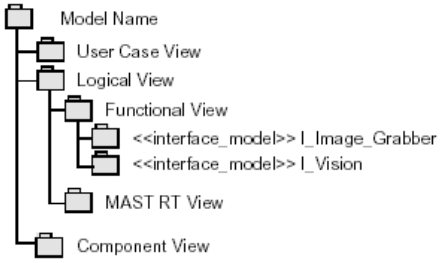

Figura 2: Descripción UML de una interfaz.

ubica dentro del paquete Functional _View, que a su vez esta definido en la Logical View del modelo.

La especificación de un componente está directamente relacionada con los dos contratos que definen su funcionalidad.

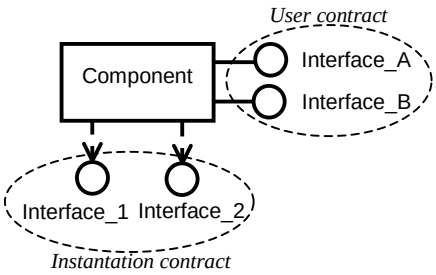

Figura 3. Especificación de un componente

En la metodología que utilizamos, toda la información relativa a la descripción de un componente se encapsula en un paquete UML con estereotipo <<component_model>> que se encuentra en el paquete <<Functional_View>>. Dentro de él, un componente se declara en UML mediante una clase con estereotipo <<component>>, e incluye:

- Identificador: que coincide con el identificador de la clase estereotipada como <<component>>.
- Contrato de uso: que representa el conjunto de interfaces que ofrece. Se formulan como atributos o clases agregadas con el estereotipo <<offered>> y hacen referencia a instancias descritas de forma independiente, sea en el mismo o en otro modelo.
- Contrato de instalación: que representa las interfaces que deben estar instanciadas para que el componente pueda operar. Se formulan como atributos o clases agregadas con el estereotipo <<required>> y hacen referencia a instancias descritas independientemente en el mismo modelo o en otros.
- Restricciones entre interfaces: Describen las restricciones que se introducen en el componente como consecuencia de que éstas se ofrecen conjuntamente en el mismo.
- Restricciones de acceso a componentes: que limitan las formas de uso con que las diferentes realizaciones pueden acceder a los componentes que se requieren para implementar su funcionalidad.

El término “Componente implementado” que se plantea en la figura 1 corresponde a un concepto de implementación y representa el producto entregable que constituye el componente implementado dentro del contexto de una tecnología de componentes concreta. En nuestro caso, es un paquete Ada de código fuente compilable o un paquete Ada compilado, que implementa a su vez la descripción de un componente y las de las interfaces que ofrece el componente.

El término “Componente instalado” corresponde a un concepto de despliegue o registro del código del componente en un nudo concreto dentro de una plataforma distribuida. Un componente instalado se encuentra dispuesto para ser activado cuando sea requerido en la fase de instanciación de una aplicación. Es un concepto que aún no se utiliza en nuestro caso, ya que no hemos adoptado una plataforma distribuida de tiempo real en nuestra metodología. Es una línea sobre la que se está actualmente trabajando.

El término “Componente instanciado” corresponde al concepto de objeto instanciado y asociado a una aplicación concreta que se está ejecutando. En nuestro caso, corresponde al código y entorno de información que resulta de compilar el paquete Ada y enlazarlo con la aplicación que hace uso de él.

2.2 MODELO DE TIEMPO REAL.

Para abordar el concepto de componente de tiempo real hay que extender a las características relativas al comportamiento temporal de los servicios que ofrece, el principio central de la metodología de componentes, que es la independencia entre la especificación y la implementación. Esto requiere disponer de una metodología que incorpore a la especificación del componente como parte de los contratos de uso y de instalación, la descripción del comportamiento temporal de los servicios que ofrece con independencia de la implementación concreta del componente que se esté utilizando.

Un método de modelado del comportamiento temporal que sea aplicable a una metodología de componentes, debe abordar dos aspectos principales:

- Proponer un modelo de interacción entre componentes que sirva de base para la descripción del comportamiento temporal.
- Proponer una metodología de modelado que permita describir las interacciones de bajo nivel entre componentes o con otras aplicaciones que aunque son transparentes desde el punto de vista

funcional, y por tanto no son descritas a través de las interfaces, son relevantes desde el punto de vista de la respuesta temporal y deben quedar reflejadas en el modelo de tiempo real.

La metodología de modelado que utilizamos está basada en el entorno MAST [3],[4],[8], el cual ofrece características específicamente adecuadas para describir el comportamiento temporal que se requiere en las metodologías de componentes.

El principio de modelado que utiliza MAST, es la concepción de cada situación de tiempo real que se presente en la aplicación como un conjunto de transacciones cuyas ejecuciones concurren en la plataforma. Cada transacción describe el conjunto de actividades que se ejecutan como consecuencia de los flujos de control que se inician como respuesta a eventos externos o temporizados de disparo. Cada actividad modela la temporización y el uso de recursos del sistema que se requieren para llevar a cabo una acción que ha de ejecutarse dentro de la transacción. La transacción representa también el marco de referencia con el que se caracteriza temporalmente la respuesta y se definen los requerimientos de respuesta temporal.

Dentro de esta estrategia para el modelado de una aplicación, el modelo MAST de un componente describe:

- La secuencia de actividades que se desencadenan como respuesta a la invocación de un servicio del componente y que describe la temporización y el uso de recursos que el componente realiza como consecuencia de la misma.
- El conjunto adicional de transacciones que el componente introduce en la situación de tiempo real como consecuencia de su estado, y que describen la temporización y la utilización de recursos que introduce el componente como consecuencia de su actividad de background.

Cuando modelamos el comportamiento temporal de la respuesta ante la invocación de un servicio ofertado por un componente, encontramos que ésta no depende únicamente del componente, sino que además depende de:

- La capacidad de las plataformas en que se ejecutan las actividades que constituyen el servicio.
- El modelo de respuesta temporal de los servicios de otros componentes que son invocados por él para realizar la funcionalidad que corresponde al servicio.
- Las variaciones de la secuencia de actividades que implica el servicio como consecuencia de datos o estados que se modifican en cada llamada.

A fin de conseguir que el modelo de temporización de un servicio de un componente sólo incluya las características de temporización o de sincronización propias del código del componente, lo que se asocia en el modelo del componente al servicio es un descriptor parametrizado, que representa las características que sean propias del componente, mientras que incluye tan solo como parámetros las referencias a los modelos de los servicios externos que invoca o a los modelos de las plataformas en que se ejecutan las actividades. Cuando dentro de una situación de tiempo real concreta se requiera el modelo temporal del servicio, éste se construye a partir del descriptor asociado al servicio, asignando a sus parámetros las referencias concretas a los modelos de los servicios concretos de los que hace uso y con el modelo concreto de la plataforma en que se ejecuta esa instancia. Así mismo, a fin de reflejar las variaciones de comportamiento temporal de la respuesta de un servicio de un componente, se asocian diferentes modelos al mismo servicio, y en ese caso, debe interpretarse que cada modelo describe una forma de uso diferente del servicio.

A fin de formular el modelo de tiempo real de un componente y de construir el modelo de tiempo real de una aplicación a partir de los modelos de tiempo real de los componentes con que se construye, se ha definido el perfil CBSE-Mast que se formula como un metamodelo UML. En la figura 4 se muestran las clases raíz definidas en este perfil.

El aspecto clave del perfil CBSE-Mast es la sistemática dualidad entre “descriptor de modelo ” e “instancia de modelo”. Cualquier elemento definido dentro de la especificación de un componente tiene asociado un descriptor de modelo, esto es, un modelo parametrizado que describe las características propias del elemento, pero que deja abiertas y asociadas a parámetros, las referencias a los modelos de otros elementos que se requieren para completar éste. Sólo cuando en el contexto de una situación de tiempo real se construye el modelo relativo al elemento, las plataformas y las instancias de los otros elementos en que se apoya su funcionalidad se conocen, por lo que a partir del descriptor del modelo asociado al elemento, y asignando a sus parámetros las referencias y valores concretos, se construye la instancia del modelo Mast del elemento, sobre el que podrán operar las diferentes herramientas de diseño y análisis que ofrece el entorno Mast.

La clase Mast_Component_Descr representa el modelo descriptor de cualquier elemento definido en el perfil. Como clase abstracta raíz sólo es un elemento contenedor, y es en las clases especializadas que se derivan de ella donde se establece la semántica y se define su estructura interna.

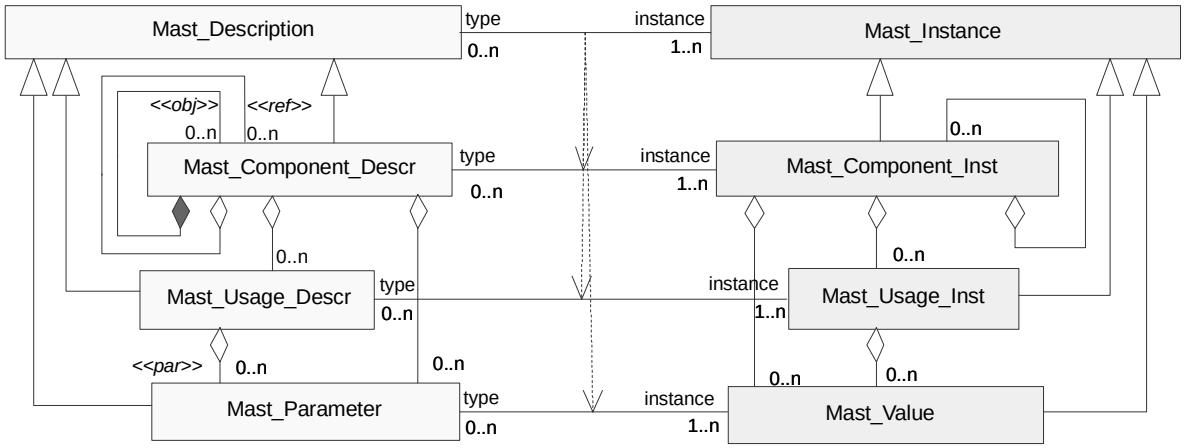

Figura 4. Núcleo del perfil CBSE-Mast.

Una clase Mast_Component_Descr, tiene asociados tres tipos de elementos:

- Conjuntos de otros componentes.
- Conjuntos de parámetros.
- Conjuntos de formas de uso.

Los componentes asociados son a veces agregados (estereotipo <<obj>>), esto es, definidos en ellos, de forma que por cada instancia que se genere del componente se generará una instancia de cada componente agregado. Representan modelos parciales que son parte del modelo que representa el componente que los contiene. Otros componentes están asociados por una relación de referencia (estereotipo <<ref>>), esto es, son componentes definidos externamente a ellos y que proporcionan modelos que son requeridos para construir el propio modelo.

Los parámetros son tipos de datos que describen las características cuantitativas del modelo que representa el componente. Cada parámetro tiene una semántica propia que depende del tipo especializado de componente en que está definido y un tipo, que debe pertenecer al conjunto de tipos básicos definidos en el entorno MAST. Optativamente puede tener asignado un valor por defecto, sin embargo cuando se defina la instancia, el valor del correspondiente Mast_Component_Inst podrá ser establecido sobreescribiendo el valor por defecto.

Los descriptores de formas de uso (“Mast_Usage Descr”) asociados a un Mast_Component_Descr modelan operaciones (simples, compuestas, jobs, transacciones) que están definidas en el modelo del componente, y están descritas (a través de un diagrama de actividad UML agregado) como una actividad o un conjuntos de actividades relacionadas entre sí por relaciones de flujo de control. Las Mast_Usage_Descr son descriptores formulados en función de un conjunto de parámetros, que representan características cuantitativas o de referencia que quedan indefinidas en el descriptor.
Para cada tipo de descriptor definido existen las correspondientes instancias (derivados de la clase raíz Mast_Instance del metamodelo), éstas constituyen instancias de modelos Mast susceptibles de ser analizados por las herramientas de diseño y análisis de planificabilidad, asignación de prioridades, etc. definidas en el entorno Mast. Una instancia se obtiene a partir de un descriptor al asignar valores y referencias a instancias concretas a los parámetros definidos en el descriptor.

En CBSE-Mast se definen conjuntos completos de clases especializadas derivadas de Mast_Component _Descr que permiten modelar sistemas de tiempo real. El entorno Mast es un entorno abierto y es susceptible de incrementar su capacidad de modelado con la definición de nuevas primitivas de modelado. En la versión actual [4],[8]está dotado de un conjunto suficiente de primitivas que permiten modelar sistemas de tiempo real basados en el sistema operativo POSIX o el lenguaje Ada y con estrategias de planificación basadas en prioridades fijas.

2.3 CONTRATO DE INSTANCIACION PARA COMPONENTES DE TIEMPO REAL.

Habitualmente los sistemas de tiempo real se implementan como sistemas cerrados en los que todos los elementos hardware y software están dedicados a él, o al menos todos los elementos del sistema son conocidos. En estos casos, basta disponer de un modelo de tiempo real de ellos para poder analizar la planificabilidad del sistema y predecir su comportamiento temporal.

Actualmente comienzan a tener relevancia los sistemas de tiempo real implementados totalmente o

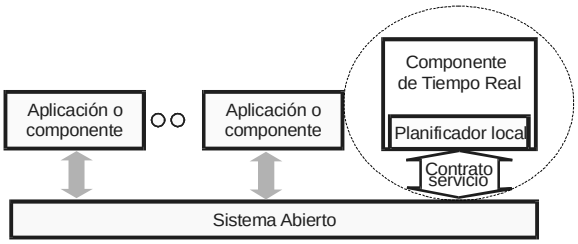

Figura 5. Componente de tiempo real en entorno abierto.

parcialmente sobre entornos abiertos, esto es, el sistema de tiempo real comparte la plataforma con otras aplicaciones no modeladas desde el punto de vista de tiempo real y que pueden interferir sobre su comportamiento temporal. En estos casos, para la correcta operación del componente de tiempo real se tienen que formular ciertas cláusulas en el contrato de instanciación.

El planteamiento sobre el que se trabaja actualmente, consiste en incorporar a los componentes que deban operar en un entorno abierto un planificador local que durante la fase de instanciación negocia con el sistema operativo un contrato de servicio. Si el contrato es aceptado, el planificador local garantizará que los requerimientos temporales ofertados en el contrato de uso del componente serán satisfechos.

La gran importancia de esta estrategia es que el análisis de tiempo real que garantiza la respuesta temporal comprometida puede realizarse "off-line" considerando tan solo el componente de tiempo real y el contrato de servicio.

Para la aplicación de esta metodología, sobre la que actualmente se está trabajando, hay que desarrollar aún los siguiente aspectos:

- Adaptar el sistema operativo para que tenga capacidad de negociar y dar soporte a los contratos de servicio.
- Definir técnicas de análisis de planificabilidad de tiempo real basadas en los contratos de servicio.
- Formular una metodología de diseño de componentes de tiempo real fundamentada en los contratos de servicio.

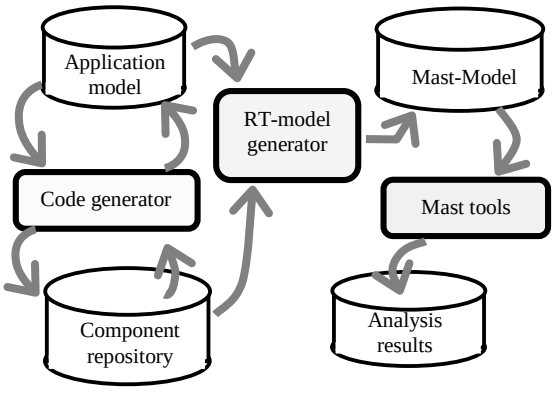

Figura 6. Estructuras de datos y herramientas

3 ENTORNO CASE DE DESARROLLO DE APLICACIONES BASADAS EN COMPONENTES.

El desarrollo de aplicaciones basadas en componentes requiere manejar el considerable volumen de información que se encuentra asociado a cada componente. El volumen de información que debe ser gestionado y la complejidad del procesamiento que esta información requiere, se incrementa notablemente cuando se abordan los modelos de respuesta temporal y de análisis de tiempo real. Todo ello implica que la aplicación de la tecnología de componentes de tiempo real debe estar soportada por herramientas CASE.

En la figura 6 se muestran los principales registros de información que se manejan en el entorno, así como las principales herramientas que las procesan.

El soporte de la información de especificación de los componentes y de la aplicación son básicamente modelos UML y ficheros de código Ada. Dado que la tecnología de componentes que se utiliza no corresponde a una tecnología estándar, ha sido necesario desarrollar las herramientas para soportar el desarrollo de nuevos componentes, así como para la construcción de la aplicación a partir de su especificación como agregación de componentes. Para la gestión de los modelos UML se ha utilizado la herramienta comercial ROSE 2001 de Rational. Sin embargo, se ha necesitado desarrollar herramientas para generar el código fuente Ada de los componentes a partir de sus propias especificaciones UML y de las especificaciones de las interfaces que oferta. Así mismo, se está desarrollado una aplicación que genera el código fuente Ada de la aplicación a partir del modelo UML y de los códigos asociados a los componentes. Ambas herramientas siguen la estrategia de generar automáticamente los ficheros de especificación Ada y los esqueletos de los ficheros correspondientes a los cuerpos Ada , y dejan al diseñador que a través de un editor complete el código que implementa la parte algorítmica.

La herramienta de generación del modelo de tiempo real de la aplicación procesa los modelos de tiempo real de la parte especifica de la aplicación, de la distribución de ésta sobre la plataforma, de los modelos de tiempo real de la propia plataforma y de los modelos de los componentes que intervienen. Todas estas informaciones son modelos UML que deben ser compilados para generar el modelo Mast que describe cada una de la situaciones de tiempo real que deban analizarse. Este proceso es algorítmicamente complejo, necesita concentrar datos

dispersos en muchos modelos y en consecuencia requiere estar automatizado por una herramienta CASE.

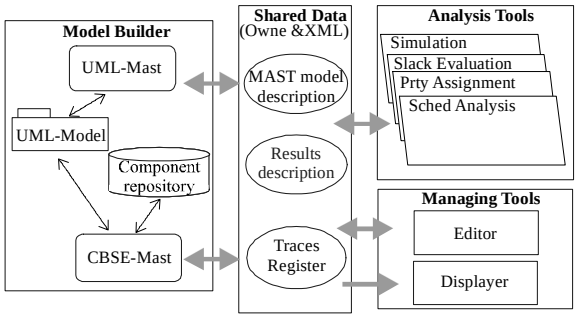

Figura 7. Estructura del entorno Mast.

En la figura 7 se muestra la estructura del entorno Mast y de las herramientas de análisis y diseño de sistemas de tiempo real. Se organiza alrededor de las estructuras de datos que representan el modelo y los resultados y trazas generados por las herramientas. Las estructuras de datos están formuladas en XML y formalizadas mediante las correspondientes definiciones Schema. Esto representa una considerable ayuda a la validación, interpretación y transformación de la información ya que habilita para la extensión del entorno todos los recursos que proporciona la tecnología XML.

El entorno está dotado con tres tipos de herramientas:

- Herramientas de gestión del modelo y de los resultados: Compuestas por interfaces gráficas para la introducción, edición y visualización de los modelos y de los resultados y trazas generadas por las herramientas.
- Herramientas de análisis y diseño de tiempo real: que procesan el modelo de tiempo real del sistema y generan información útil para el análisis y diseño del sistema que se desarrolla.
- Herramientas de generación del modelo: que ayudan a generar el modelo de tiempo real a partir de la especificación, de la descripción funcional, lógica y de despliegue o del código que se generan en el proceso de desarrollo.

4 COMPONENTES PARA AUTOMATIZACIÓN INDUSTRIAL.

Aunque la metodología que se presenta está aún en desarrollo, con el objetivo de validar los resultados que se van obteniendo, se han formulado una serie de dominios todos ellos relacionados con el campo de la automatización industrial. Para ellos, se han definido familias de interfaces que posibilitan su gestión y así mismo, para su validación, se han implementado en objetos concretos y con ellos se han construido aplicaciones de demostración de la tecnología. El objetivo de esta experiencia ha sido la verificación de la tecnología y por ello, no se ha cuidado especialmente la completitud en el desarrollo de los dominios.

La metodología va fundamentalmente orientada al desarrollo de componentes para diseño de aplicaciones de tiempo real, y por ello se ha desarrollado una tecnología compatible éstas. Se ha utilizado el lenguaje Ada como medio de implementación de los componentes, y los entornos de ejecución han sido principalmente sistemas basados en el núcleo de tiempo real MaRTE OS, que al haber sido construido por nuestro grupo, nos facilita un control absoluto sobre ellos, y en algunos casos, también se ha utilizado Windows NT, aunque mantener la predictibilidad para esta plataforma requiere condicionamientos no siempre soportables.

Los dominios, interfaces y componentes desarrollados han sido:

DOMINIO: Adquisición de señales analógicas y digitales (Adq).

Tiene como objetivo la adquisición y generación de señales analógicas y digitales a través de tarjetas de IO de propósito general instaladas en computadores con arquitectura PC.

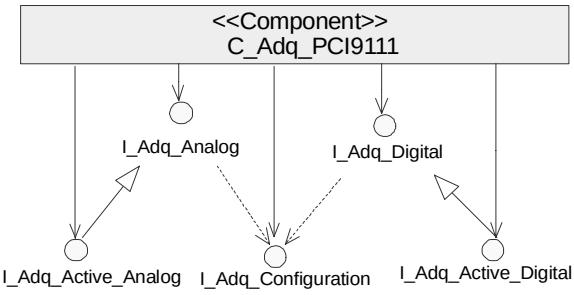

Figura 7. Componente del dominio ADQ

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

Interfaz **I_Adq_Analog**: Conjunto de operaciones de adquisición y generación de tensiones analógicas a través de convertidores A/D y D/A.

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

Interfaz **I_Adq_Digital**: Conjunto de operaciones de lectura y escritura en líneas y grupos de líneas de entrada y de salida digitales.

Interfaz **I_Adq_Active_Digital**: Interfaz activa que añade a la interfaz básica I_Adq_Digital 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_Adq_PCI9111**: Componente basado en la tarjeta de adquisición PCI 9111 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 y 16 bits digitales de salida. Este componente implementa las 5 interfaces del dominio Adq antes citadas. El componente se ha desarrollado para los entornos MaRTE OS y Windows NT.

DOMINIO: Adquisición y digitalización de imágenes de vídeo (IG).

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 del PC.

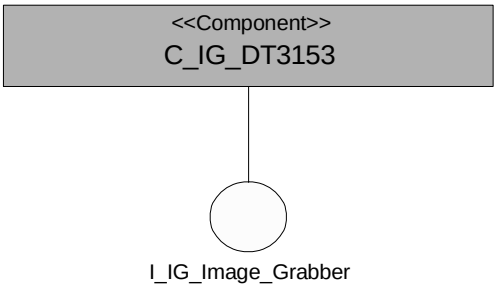

Figura 8. Componente del dominio IG.

Interfaz **I_IG_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 . Esta desarrollado en el entorno Windows NT.

DOMINIO: Procesado digital de imágenes (Img).

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

Interfaz **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, los procedimientos de almacenamiento y recuperación y los diferentes formatos de codificación de la misma. Es utilizada por las restantes interfaces de visión.

Interfaz **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 las características asociadas a los pixels.

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

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

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

Interfaz **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.

Interfaz **I_Img_Draw**: Ofrece un conjunto de operaciones que permiten dibujar e insertar distintos elementos en una imagen.

Componente **C_Img_Ada_Vision**: Este componente ofrece los recursos relativos a gestión, procesado de imágenes y análisis de imágenes. Su funcionalidad corresponde a las librerias IPL y Open_CV 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. Está desarrollado para el entorno Windows NT.

DOMINIO: Graphic Panel (GP)

Define el conjunto de conceptos y recursos informáticos relacionados con el control y la supervisión de un panel gráfico sensible al tacto.

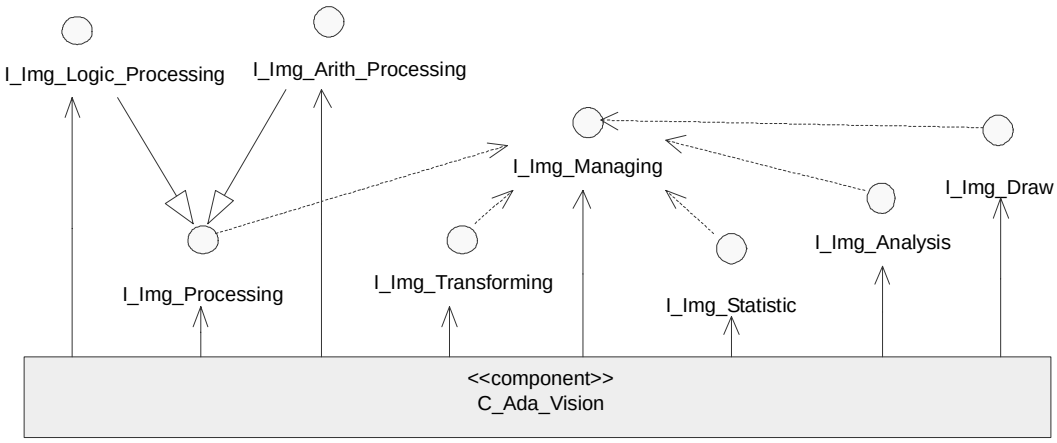

Figura 9. Componente del dominio Img.

Interfaz **GP_Driver**: Ofrece los servicios para la gestión de un componente que ha sido programado en el panel.

Interfaz **GP_Component**: Interfaz abstracta de un componente, con las propiedades y funciones básicas (cargar, actualizar, borrar, mostrar, etc.) de cualquier componente que pueda ser presentado en el panel. Esta interfaz se especializa en un conjunto abierto de interfaces concretas que permiten la gestión de cada tipo de componente programable en el panel.

Interfaz **GP_Reactive_Component**: Interfaz abstracta especialización de la interfaz GP_Component, que corresponde a los elementos que pueden desencadenar asíncronamente acciones cuando son pulsados o cuando cambia su estado.

Componente **GP_37W2_PBIII**: Está basado en el panel gráfico GP_37W2 y el software de desarrollo GP_PRO_PBIII. Es un componente concreto que implementa todas las interfaces del dominio GP. Ha sido implementado para los entornos MaRTE OS y Windows NT.

DOMINIO: Ada Speaker Sound (ASS)

Ofrece recursos para interpretar sonidos y melodías musicales a través del altavoz de un PC. Así mismo ofrece recursos para componer melodías a partir de la información habitual en un pentagrama.

Interfaz **ASS_Composer**: Ofrece los recursos para componer una melodía a partir de la información contenida en un pentagrama.

Interfaz **ASS_Player**: Ofrece procedimientos para interpretar una melodía en el altavoz de un PC.

Componente **ASS_MarteSpeakerSound**:

Componente que realiza el dominio ASS, en el entorno MaRTE OS.

6 CONCLUSIONES.

La tecnología de componentes es considerada actualmente como la solución mas prometedora de la ingeniería software para acortar los tiempos de desarrollo y generar aplicaciones fiables dentro de la tendencia de incremento continuado de la complejidad.

Sin embargo el principio básico de esta metodología de independencia entre la especificación y la implementación están en cierto modo en contradicción con la característica de predictibilidad que requieren las aplicaciones de tiempo real, en las que el comportamiento temporal de la respuesta, que es parte fundamental de su especificación, depende claramente de su estructura interna y de la plataforma sobre la que opera.

La línea de trabajo que se ha presentado y que aún está en desarrollo propone asociar una metodología de modelado del comportamiento temporal de los servicios al contrato de uso del componente. Estos modelos tienen las necesarias características de componibilidad para permitir construir un modelo fiable de una aplicación a partir de los modelos de los componentes que utiliza y de la plataforma sobre la que se ejecutan. Este proceso es complejo por el volumen de información que requiere y por la complejidad algorítmica que conlleva, pero puede ser automatizado dentro de un entorno CASE.

Si el entorno en el que se ejecuta la aplicación de tiempo real basada en componentes es abierto y pueden existir otras aplicaciones no bien conocidas que interfieran con ella, se propone complementar el contrato de instalación de los componentes de tiempo real con un contrato de servicio, que será negociado con el sistema operativo en la fase de instanciación y que en caso de ser aceptado, garantice la respuesta temporal ofertada por el componente en sus servicios. Para hacer operativa esta propuesta se está desarrollando una nueva extensión de los sistemas operativos basados en el estándar POSIX de tiempo real a fin de que soporten los servicios de uso basados en contrato.

A fin de finalizar la metodología se han desarrollado un conjunto de dominios, interfaces y componentes concretos que permiten construir aplicaciones de demostración de la validez y ventajas de esta tecnología.

Agradecimientos.

El presente trabajo ha sido financiado por el Programa Nacional de Tecnologías de la Información y de las Comunicaciones, Proyecto TRECOM (TIC 2002-04123-C03-02).

Referencias.

[1] Chessman J. y Daniels J.: (2000) "UML Components Software: A simple process for specifying component-based software", Addison-Wesley.

[2] Cmkovic I. y otros, (May, 2000) "A case of study: Demands on component based development", Proc. of 22nd Int. Conf. of Software Engineering, Cannes (France).

[3] Drake J.M., Medina J.L. y González Harbour, M., (2000) "Entorno para el diseño de sistemas basados en componentes". X Jornadas de Concurrencia, Jaca (España).

[4] González Harbour M, Gutierrez J.J.,Palencia J.C. y Drake J.M. (June, 2001) "MAST: Modeling and Analysis Suite for Real-Time Applications" 13th Euromicro Conference on Real Time Systems Delf(Netherlands).

[5] Isovic D. y Norström C., (2000) "Component on real-time systems" The 8th Int. Conf. On Real-Time Computing System and Applications (RTCSA-2002), Tokyo (Japan).

[6] Larsson M. y otros, (2000) "Development experiences of a component-based systems" Proc. of 7th IEEE Int. Conf. on Engineering of Computer Based Systems (ECBS00).

[7] Lüders F. y Crnkovic I., (2001) "Experiences with component-based software development in industrial control" Mälardalen Real-Time Research Center.

[8] Medina J.L., González Harbour M. y Drake J.M., (December, 2001) "MAST Real Time View: A graphical UML tool for modeling object-oriented real-time applications" 22th IEEE Real-Time Systems Symposium (RTSS), London (England).

[9] Medina J.L. y Drake J.M., (2002) "Modelado y análisis de tiempo real de aplicaciones basadas en componentes" V Jornadas de Tiempo Real, Cartagena, (España).

[10] Norström C. y otros, (April, 2001) "Experiences from introducing state-of-the-art real-time techniques in the automotive industry" Proc. of 8th IEEE Int. Conf. on Engineering of Computer Based Systems (ECBS01) Washington (USA).

[11] Yaw S.S. y Xia B., (April, 1998) "An approach to distributed component-based real-time application software development" IEEE Int. Simp. On Object-Oriented Real-Time Distributed Computing (ISORT-98).