Plataforma CASE Basada en Componentes para la Docencia de Ingeniería del Software

Francisco J. García Peñalvo, Iván Álvarez Navia

***Resumen*—La Ingeniería del Software es una de las disciplinas más importantes en la formación de un ingeniero informático, porque le confiere los conocimientos necesarios para el desarrollo de software de calidad y económicamente viable.**

Una parte fundamental dentro de la Ingeniería del Software es el manejo de herramientas que automaticen el proceso de desarrollo de aplicaciones software a lo largo de las distintas fases del ciclo de vida del proyecto. Estas herramientas reciben el nombre de herramientas CASE.

En este artículo se presenta Left-CASE, una plataforma CASE basada en componentes, que ofrece un amplio soporte a las técnicas de modelado del software que se estudian en la Ingeniería Informática, fundamentalmente en las disciplinas relacionadas con la Ingeniería del Software.

Esta herramienta cumple dos objetivos fundamentales: a) Permite futuras ampliaciones en cuanto al tipo de técnicas de modelado soportadas, siendo esto completamente transparente al usuario de la herramienta; y b) Todas las técnicas de modelado están integradas en el mismo entorno, presentando la misma filosofía de interacción con el usuario.

***Palabras claves*— Desarrollo basado en componentes, Herramientas CASE, Ingeniería del Software, Técnicas de modelado**

I. Introducción

El desarrollo de las clases prácticas de las asignaturas relacionadas con la disciplina de Ingeniería del Software, suele estar ligado a disponer de herramientas que automaticen en la mayor medida posible las diversas tareas del ciclo de vida del software, esto es, la integración de la tecnología CASE (*Computer Aided/Assisted Software/System Engineering*) en el currículo de un Ingeniero Informático [3].

La diversidad de posibilidades en cuanto a técnicas y herramientas que aparecen o pueden aparecer en un proceso software, dispara las necesidades de herramientas ligadas a esta disciplina, lo que se traduce en una demanda imposible de atender en términos de licencias comerciales para los laboratorios que dan cobertura a esta disciplina.

Una solución bastante extendida suele ser combinar soluciones comerciales, como por ejemplo **Designer** o **Developer** de **Oracle Inc.**, con herramientas de libre distribución o de demostración, como por ejemplo **ArgoUML** (http://argouml.tigris.org/) o **COCOMO II** (http://sunset.usc.edu/available_tools/index). De esta forma una organización (Escuela, Facultad, Departamento...) puede controlar su gasto en licencias software, sin limitar sustancialmente el soporte docente.

No obstante, esta situación tampoco es perfecta; así, por un lado, las soluciones comerciales suelen ser muy poco flexibles y pueden estar sujetas a contratos de mantenimiento de alto coste, mientras que por otro lado, las soluciones de libre distribución pueden no cubrir todo el espectro de necesidades de una propuesta docente (como sucede con las técnicas de modelado propias del paradigma estructurado, por ejemplo los diagramas de flujo de datos) y las versiones de demostración ponen ciertas limitaciones (temporales, de número de elementos o de funcionalidad...) que no las hacen aconsejables para un uso exhaustivo en su versión limitada, como por ejemplo **Rational Rose** (http://www.rational.com).

A todo esto se debe unir el hecho de que trabajar con múltiples herramientas no favorece una visión integrada del proceso de desarrollo de software, así como que el tiempo que el docente debe dedicar a buscar, evaluar y adquirir herramientas CASE es bastante elevado.

Por todos estos motivos, en el Departamento de Informática y Automática de la Universidad de Salamanca se ha optado por desarrollar un conjunto de herramientas CASE que se ajustasen a las necesidades propias de las asignaturas relacionadas con la Ingeniería del Software en las titulaciones de Ingeniería Informática que se imparten en esta Universidad.

Dichas herramientas están fundamentalmente ligadas a las técnicas de modelado propias de las fases de análisis y diseño, aunque también se han intentado cubrir aspectos básicos de la gestión de proyectos informáticos.

La forma de afrontar el desarrollo de estas herramientas ha dado lugar a dos familias de aplicaciones. La primera familia está formada por herramientas independientes, cada una de las cuales ofrece soporte a un número reducido de técnicas de modelado o de gestión de proyectos. Con estas herramientas se abordan los problemas de las licencias y de la adecuación de las mismas a los contenidos docentes impartidos en la Universidad de Salamanca, pero se falla en dar una visión integradora del proceso software.

Surge así una segunda familia de herramientas, que está basada en un entorno CASE extensible, donde las técnicas soportadas puedan crecer, pero siempre bajo las restricciones de compatibilidad con el entorno. Para lograr este objetivo se

Dr. F. J. García Peñalvo. Departamento de Informática y Automática. Universidad de Salamanca. fgarcia@usal.es.

I. Álvarez Navia. Departamento de Informática y Automática. Universidad de Salamanca. inavia@usal.es.

diseña una arquitectura formada por un *framework* base o contenedor y un conjunto de componentes específicos. El contenedor hace las veces de "anfitrión" para los componentes, ofreciéndoles los servicios comunes necesarios, así como el conjunto de interfaces necesarias para que dichos componentes puedan incorporarse al entorno construido, mientras que cada uno de los componentes debe proporcionar todas las características propias de una técnica de modelado concreta.

Con este segundo enfoque se logran los objetivos buscados en cuanto a licencias, adecuación a la titulación e integración del proceso software, pero además se permite una fácil evolución de la herramienta, dado que al desacoplar el entorno de trabajo (o plataforma) de las técnicas software concretas que se soportan, se pueden incorporar nuevas técnicas sin tener que modificar el trabajo ya realizado. Además, contar con un elemento anfitrión o contenedor ofrece una filosofía de manejo común para todas las técnicas, lo que facilita en gran medida el aprendizaje y manejo de la herramienta.

Este artículo se centra en presentar esta plataforma CASE basada en componentes y que recibe el nombre de Left CASE. El resto del artículo se organiza como sigue: en la sección dos se repasan brevemente las herramientas desarrolladas en el Departamento de Informática y Automática de la Universidad de Salamanca con anterioridad a Left CASE. La sección tercera introduce los aspectos técnicos de Left CASE, haciendo un especial hincapié en su arquitectura basada en componentes. La cuarta sección presenta a Left CASE desde una perspectiva funcional. Por último, la sección cinco cierra el artículo presentado las conclusiones del mismo e introduciendo los siguientes pasos a realizar.

II. Antecedentes de Left CASE

Como se ha comentado en el apartado anterior, en una primera aproximación se abordó el desarrollo de una serie de herramientas independientes entre sí que daban cobertura a las técnicas más representativas que, en relación con el modelado de sistemas software y la gestión de proyectos, se imparten en la Ingeniería Informática (tomando como referente los Planes de Estudio en vigor de las titulaciones Ingeniería Técnica en Informática de Sistemas e Ingeniería Informática, ambas impartidas en la Universidad de Salamanca), contemplando tanto al paradigma estructurado como al paradigma objetual.

Concretamente, se construyeron cuatro herramientas CASE, ADAM CASE, ER CASE, CRC CASE y Gestor de Proyectos, las cuales están disponibles en la dirección web http://tejo.usal.es/~fgarcia/docencia/isoftware/case/casetools.html.

Las tres primeras ofrecen soporte al modelado en análisis y diseño, soportando las siguientes técnicas: Diagramas de Flujo de Datos (DFD) [11], Diagramas Entidad-Relación (DER) [1], Diagramas de Clase (DC) con notación UML (*Unified Modeling Language*) [7], y Tarjetas CRC (*Class, Responsability and Collaboration*) [10]. La cuarta se centra en la gestión de proyectos, permitiendo el control de calendarios, actividades, estimaciones, costes y personal.

A continuación se repasan someramente las características de cada una de estas herramientas, para concluir esta sección con las lecciones aprendidas de estos desarrollos, lecciones que dieron lugar a Left CASE.

A. *ADAM CASE*

ADAMCASE V1.1 [2] es una herramienta CASE frontal para plataformas **MS Windows**, que facilita el desarrollo de proyectos estructurados y orientados a objetos (utilizando la notación UML). Inicialmente, ADAMCASE soporta un tipo de técnica de modelado para cada tipo de proyecto:

- Diagramas de Flujo de Datos (DFD), siguiendo la notación propuesta por Edward Yourdon [11] en proyectos estructurados.
- Diagramas de clases, siguiendo la notación propuesta por UML [7], en proyectos orientados a objetos.

En ADAM CASE el proyecto es la unidad básica de trabajo. En torno a ella se organizan los distintos elementos a utilizar: los diagramas, los componentes y, en el caso de proyectos estructurados, el diccionario de datos.

La arquitectura de la herramienta desarrollada, entendiendo como arquitectura la estructura global con sus subsistemas, sus interrelaciones y su correspondiente documentación, está basada en el modelo propuesto por [5] para un entorno CASE orientado al objeto. Dicho modelo propone tres capas: *interfaz*, *semántica* e *integración*.

Al diseñar la interfaz de usuario se buscó que el aspecto de ésta no variara en exceso de otras herramientas comerciales, eligiéndose como modelo la interfaz ofrecida por la herramienta Rational Rose 98 (http://www.rational.com) por ser la herramienta CASE más difundida para el trabajo con UML. El aspecto de la interfaz de ADAM CASE 1.1 se puede apreciar en la Figura 1.

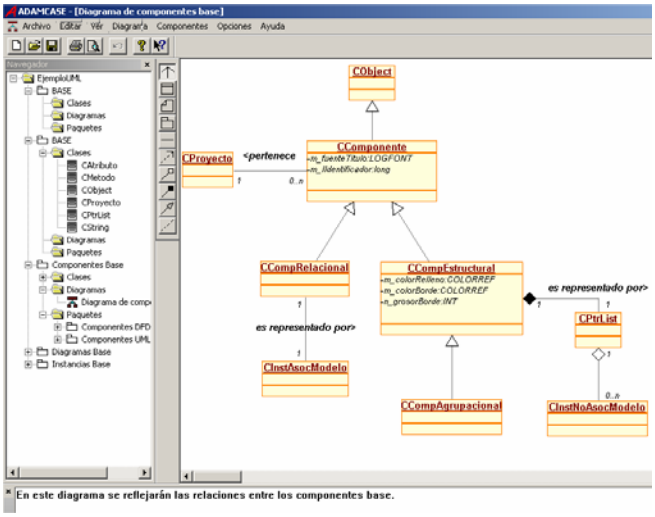

Fig. 1. ADAM CASE 1.1

B. *ER CASE*

ER CASE V1.1 es una herramienta CASE frontal para plataformas **MS Windows**, que permite la creación de Diagramas Entidad Relación (DER), utilizando la notación de

Chen [1], así como su posterior paso a un modelo lógico de datos basado en el modelo relacional.

Sobre los diagramas se efectúan comprobaciones básicas para la detección de errores. Además, se genera automáticamente un diccionario de datos, a partir del diagrama entidad-relación realizado.

El usuario tiene en todo momento conocimiento del proyecto en el que está trabajando y de todos los componentes existentes en el DER, gracias al navegador. Esta herramienta utiliza unos conceptos y unas metáforas similares a las empleadas en ADAM CASE, como se puede apreciar en la Figura 2.

Los proyectos pueden ser almacenados y se permite al usuario imprimir cualquier parte del proyecto. Adicionalmente, se puede generar un informe con toda la información.

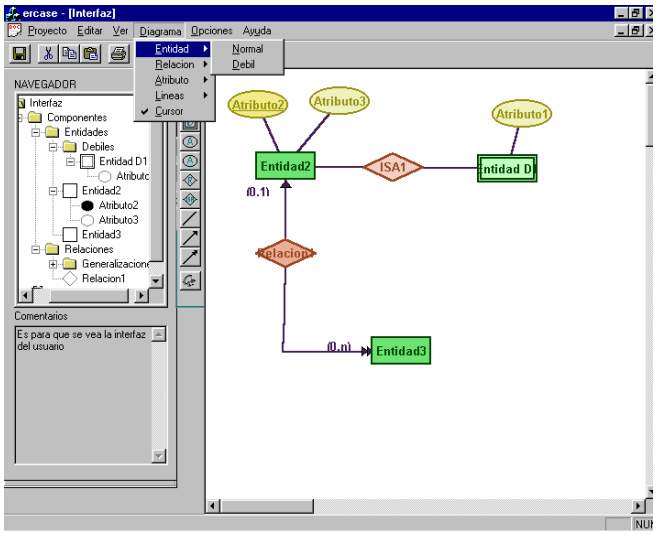

Fig. 2. ER CASE 1.1

C. *CRC CASE*

CRC CASE es una herramienta cuyo cometido consiste en asistir al desarrollador en una de las tareas más críticas, el descubrimiento de clases. Para ello se basa en una técnica muy intuitiva que es conocida como Tarjetas de Clase o Tarjetas CRC [10], donde se van repartiendo las responsabilidades funcionales entre las clases e identificando que clases colaboradoras necesitan para llevar a cabo dichas responsabilidades.

El problema planteado en el descubrimiento de clases es que marca justo el cambio hacia la orientación al objeto, a partir de este momento, los proyectos que utilicen este tipo de técnica, comenzarán a distinguirse con claridad de aquéllos que utilizan técnicas estructuradas.

A pesar de que la barrera existente entre el análisis y el diseño en los desarrollos orientados a objetos es algo poco perceptible, y en muchas ocasiones da lugar a cierta confusión, se puede considerar que CRC CASE se encuentra dentro de las denominadas CASE frontales o superiores, ya que éstas abarcan las primeras fases de análisis y diseño, lugar donde se encuentra localizado el descubrimiento de clases.

CRC CASE se caracteriza por los siguientes aspectos:

- Interfaz de usuario sencilla e intuitiva (Figura 3).
- Comprobación de errores.
- Enlace con ADAM CASE, de forma que se pueda continuar trabajando con los datos generados.
- Generación de informes.
- Posibilidad de ser explotada en cualquier tipo de plataforma al estar desarrollada en Java.

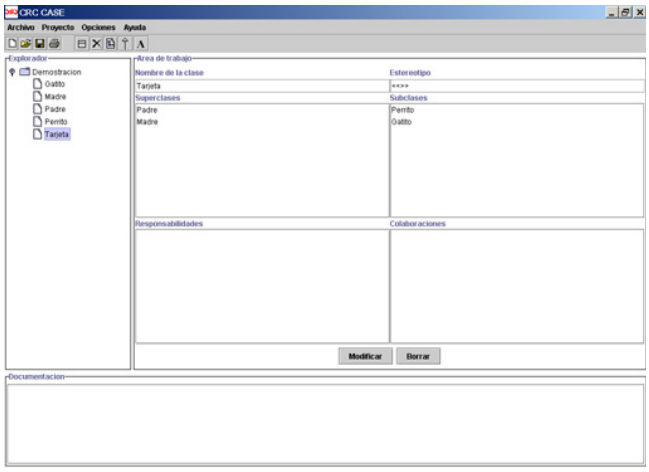

Fig. 3. CRC CASE 1.0

D. *Gestor de Proyectos*

El Gestor de Proyectos es una herramienta desarrollada para plataformas **MS Windows**, cuyo objetivo fundamental, dentro de la gestión de proyectos software, se centra en la planificación, lo que incluye obtención de datos, tratamiento y mantenimiento de los mismos y generación de informes.

Esta herramienta presenta una interfaz basada en formularios para la obtención y posterior tratamiento de los datos necesarios en un proyecto software. Como ejemplo de este tipo de escenarios de interacción se presenta en la Figura 4 el diálogo para el establecimiento de tareas y su estimación de recursos y tiempos.

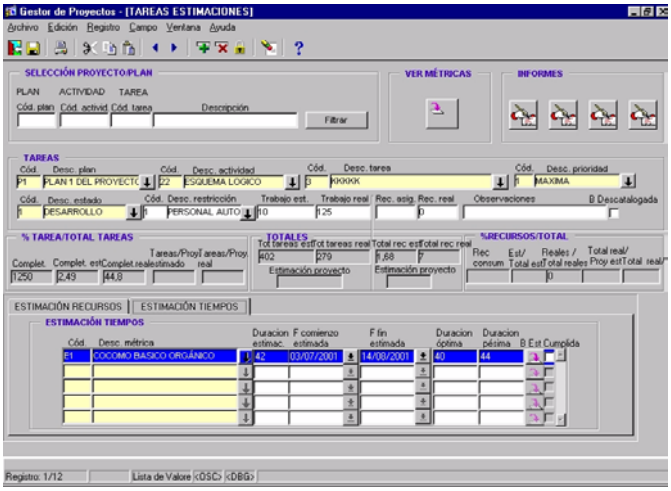

Fig. 4. Gestor de Proyectos: Diálogo para el establecimiento de tareas y su estimación de recursos y tiempos

Para controlar todos los apartados que confluyen en la planificación del proyecto software se ha utilizado la metáfora

del navegador como se puede apreciar en la Figura 5.

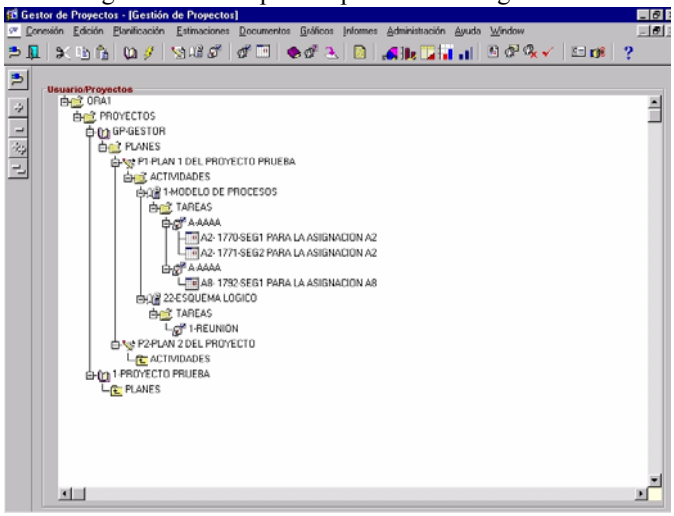

Fig. 5. Navegador del Gestor de Proyectos

E. Lecciones aprendidas

Las herramientas presentadas en esta sección han cumplido, y siguen cumpliendo, la misión docente para la que fueron desarrolladas, tanto en la Universidad de Salamanca como fuera de ella.

Desde el curso 2000-2001 se utilizan de forma continuada en las asignaturas relacionadas con la Ingeniería del Software y en la realización de los Proyectos de Fin de Carrera, especialmente ADAM CASE y ER CASE.

No obstante, con la experiencia obtenida en el desarrollo de dichas herramientas y desde una perspectiva más crítica, se detectan dos problemas fundamentales:

- Cada herramienta es independiente del resto, no habiendo integración entre ellas, ni en el ámbito de los modelos ni en el ámbito de la interfaz de usuario.
- Si se quiere ampliar el número de técnicas soportadas, se deben desarrollar nuevas herramientas, o bien ampliar las existentes, modificando su código.

Por estos motivos, se abre una nueva línea de trabajo consistente en desarrollar una plataforma CASE basada en componentes en la que exista un contenedor en el que se puedan alojar diversos componentes, cada uno de los cuales representa una técnica diferente. De esta forma, introducir nuevas técnicas significa desarrollar nuevos componentes, sin tener que modificar el trabajo ya existente. Además, gracias a la existencia de ese contenedor común, la interfaz de la herramienta es consistente y familiar a sus usuarios, salvando, lógicamente, las diferencias que vienen impuestas por las características intrínsecas de cada técnica.

Con esta filosofía nace Left CASE, cuya arquitectura y funcionalidad serán objeto de estudio en los siguientes apartados de este artículo.

III. Arquitectura de Left CASE

Como ya se ha venido introduciendo en las secciones previas, Left CASE es una herramienta basada en componentes en la que desacopla el entorno de trabajo o plataforma (*framework* o anfitrión) de las técnicas concretas que se soportan (componentes), de forma que se pueden incorporar nuevas técnicas sin tener que tocar el trabajo ya realizado.

Una de las primeras decisiones que se tomaron una vez definido el alcance y filosofía de la plataforma fue la elección del modelo de componentes sobre el que implementar Left CASE. En relación con el modelo de componentes existen diferentes opciones, algunas con un marcado carácter propietario y otras ligadas al denominado software libre.

Dadas las características del proyecto, realizado por y para miembros de la comunidad académica, se optó, desde un principio, por trabajar con herramientas de libre distribución. Dentro de éstas, dos son las alternativas más destacadas: GNOME (http://www.gnome.org) con su modelo de componentes Bonobo, y KDE (http://www.kde.org) con su modelo de componentes Kparts.

La decisión final del equipo de trabajo fue decantarse por GNOME debido a varias razones, entre las que cabe destacar las siguientes:

- Empieza a contar con un importante apoyo, no sólo de los clásicos del desarrollo del software libre, sino también con el de algunos sectores importantes de la industria (SUN, HP).
- El mecanismo de interoperabilidad entre componentes se basa en CORBA [8], estándar en auge y con gran aceptación tanto en el mundo académico como en el empresarial.
- El modelo de componentes Bonobo [4] permite unos niveles de integración entre aplicaciones, dentro del entorno de escritorio GNOME, que, hasta ahora, tan sólo existía en algunas soluciones propietarias.

Una vez fijado el contexto en el que se desarrolla el trabajo, se va a presentar la arquitectura de la plataforma Left CASE.

En primer lugar, y como es obvio en una arquitectura de estas características, se ha diferenciado claramente la parte que va a hacer las veces de un *framework* de componentes y de los componentes en sí mismos. Esta estructura queda esquematizada en las figuras 6 y 7, donde se muestra, de forma genérica, la inclusión de componentes dentro de un contenedor utilizando Bonobo. Concretamente en la Figura 6 se describe un contenedor que tiene en su interior dos componentes. El contenedor ofrece a los componentes una serie de interfaces: `Bonobo::ClientSite` y `Bonobo::ViewFrame`. Estas interfaces se definen en Bonobo utilizando IDL (*Interface Definition Language*), y son las encargadas de permitir que un componente pueda existir dentro del contenedor. Par incrustar un componente dentro de un contenedor, se debe crear un *ClientSite* donde se almacena. Este componente puede tener varias vistas que comparten datos, pero que pueden mostrar diferentes aspectos de éstos. Cada instancia de un componente necesita que dentro del contenedor exista un *ViewFrame,* es decir, una vista para la

instancia.

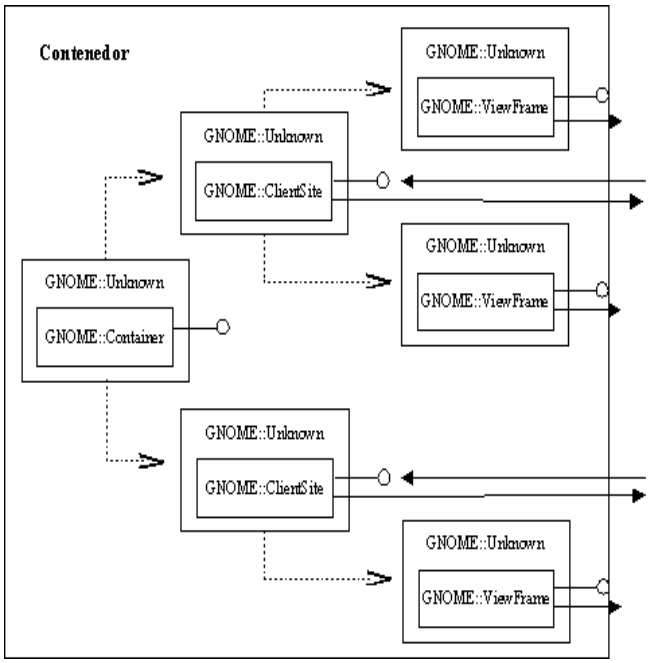

Fig. 6. Relación entre componentes y contenedores en Bonobo: detalle del contenedor

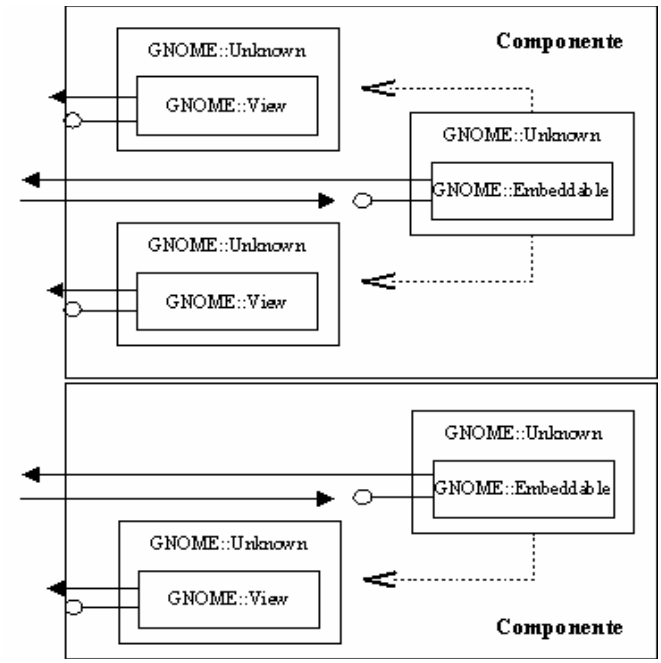

Fig. 7. Relación entre componentes y contenedores en Bonobo: detalle de los componentes

En cuanto a los componentes, ver Figura 7, existe una interfaz fundamental: `Bonobo::View`. Mediante esta interfaz el componente se comunica con el contenedor, utilizando la interfaz del contenedor *ViewFrame*, siendo también el lugar por el que se pasan al componente los sucesos que puedan ocurrir en el contenedor.

Teniendo en cuenta esta estructura de aplicación que impone el uso de Bonobo, el diseño arquitectónico propuesto es el mostrado en la Figura 8, formado por tres paquetes bien diferenciados: **GUI**, **Gestión Administrativa** y **Componente**.

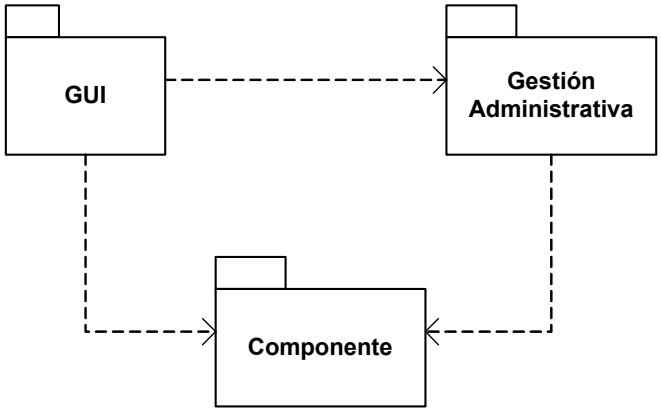

Fig. 8. Diagrama de paquetes del sistema

El paquete **GUI** representa la interacción del sistema con el usuario, intercambiando los datos con él y encargándose de responder a los eventos. También se incluye en esta parte los diálogos de la aplicación. Tiene dos clases:

- **Contenedor**, que proporciona el espacio para alojar el componente y define una serie de métodos para su manejo, y
- **Ventana**, que representa la ventana que se muestra al iniciar la aplicación y va a englobar los menús, los aceleradores de teclado...

El paquete **Gestión Administrativa** gestiona el almacenamiento de los diagramas creados por el sistema cuyo componente haya sido descargado. Además, almacena todos los datos del proyecto en el que se está trabajando, así como la interfaz para poder interactuar con él. Las tres clases que lo componen son las siguientes:

- **Proyecto**, clase que guarda toda la información del proyecto en estudio y ofrece la interfaz para manejar todos los datos del proyecto.
- **Diagrama**, que especifica a que componente pertenece dicho diagrama y otros datos de interés a la hora de recuperar el componente.
- **Instancia de modelado**, que representa los datos de cada uno de los elementos que pueda tener el diagrama, y que la aplicación guarda por si fuera necesario volver a cargar el componente.

El paquete **Componente** es una entidad independiente que se incrusta en el contenedor, pero se muestra en el diagrama de la arquitectura global del sistema (Figura 8) para mejorar la comprensión y análisis del mismo. En lo que respecta a su diseño, depende del tipo de funcionalidad específica que ofrece cada componente en concreto. Sin embargo, debido a la característica por la que todo componente se empotra en el contenedor, todos parten de una base arquitectónica común, formada por tres paquetes: **GUI-Componente**, **Gestión Componente** y **Elementos de modelado del componente**, como se puede apreciar en la Figura 9.

El paquete **GUI-Componente** abarca toda la interfaz gráfica de usuario y gestiona la comunicación con el mismo.

El paquete **Gestión Componente** engloba los elementos

que tiene el componente. En general, todos los componentes cuentan con tres clases fundamentales, a las que se pueden añadir otras, dependiendo del componente concreto. Las tres clases comunes son:

- **Navegador**, que gestiona el comportamiento del navegador y cómo éste debe ser actualizado según las modificaciones que se efectúen en el panel de dibujo.
- **Barra de herramientas**, que ofrece la interfaz necesaria para que se pueda cambiar el tipo de dibujo seleccionado en el sistema. Cuando sucede algo en la barra de herramientas ésta se encarga de actualizar el panel.
- **Panel**, que representa el lugar donde se hacen los dibujos. Gestiona los sucesos que se producen en el área de dibujo para que se puedan crear y modificar los elementos.

El paquete **Elementos de modelado del componente** incluye los conceptos propios de la técnica representada por el componente.

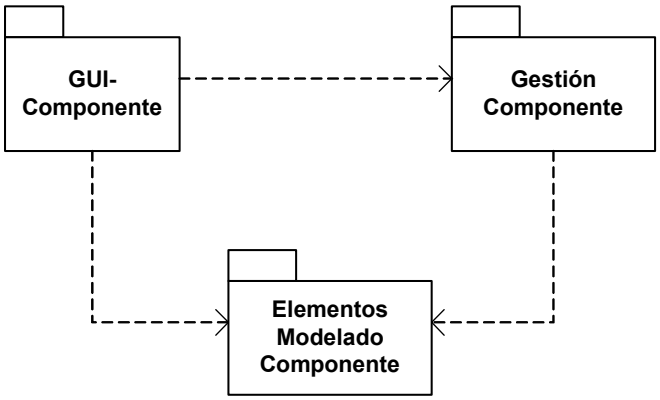

Fig. 9. Diagrama de paquetes del componente

Otro aspecto que merece la pena destacar en relación con la arquitectura del sistema es cómo se produce la activación de los componentes. Para este cometido se ha utilizado OAF, la Infraestructura de Activación de Objetos (*Object Activation Framework*). OAF utiliza un sistema muy sencillo de configuración consistente en una serie de archivos XML situados en un directorio conocido por todas las aplicaciones (generalmente `/usr/share/oaf`), en los cuales se incluye una entrada por cada servidor CORBA instalado en el sistema.

El uso de esta arquitectura separa el desarrollo del contenedor del desarrollo de los componentes, de manera tal que estos últimos son instancias de componentes que pueden ser activados en cualquier momento, sin que el usuario tenga conocer nada sobre su localización.

IV. Funcionalidad de Left CASE

Left CASE es una herramienta CASE que permite al usuario la creación, modificación, verificación y almacenamiento de diferentes tipos de diagramas, correspondientes a otras tantas técnicas de modelado, ofreciendo además un soporte a la generación automática de la documentación técnica asociada. Integrándose todo ello en un producto software que presenta una interfaz gráfica común e intuitiva, que resulta fácil de manejar, ejecutándose en plataformas UNIX/GNOME como se puede apreciar en las figuras 10 y 11, en las que respectivamente se muestra Left CASE ejecutándose en un PC con sistema operativo Linux/Debian y en una estación SUN Enterprise 250-Server con sistema operativo Solaris.

La versión actual de Left CASE soporta cuatro tipos de técnicas de modelado, que se agrupan en dos tipos de proyectos: estructurados y orientados a objetos.

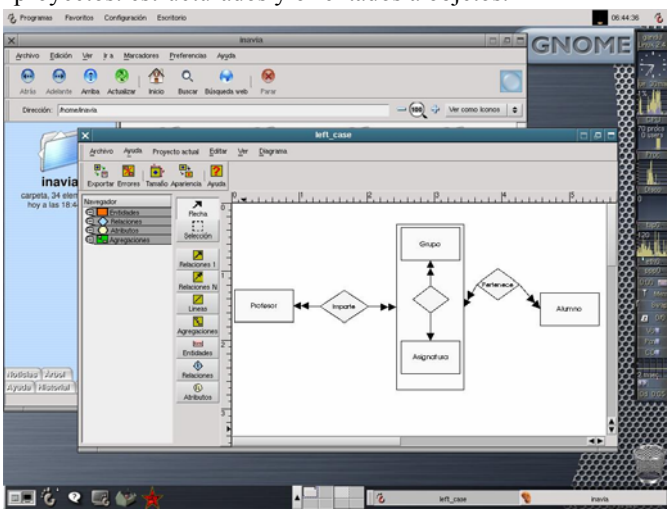

Fig. 10. Left CASE ejecutándose en un PC con sistema operativo Linux/Debian

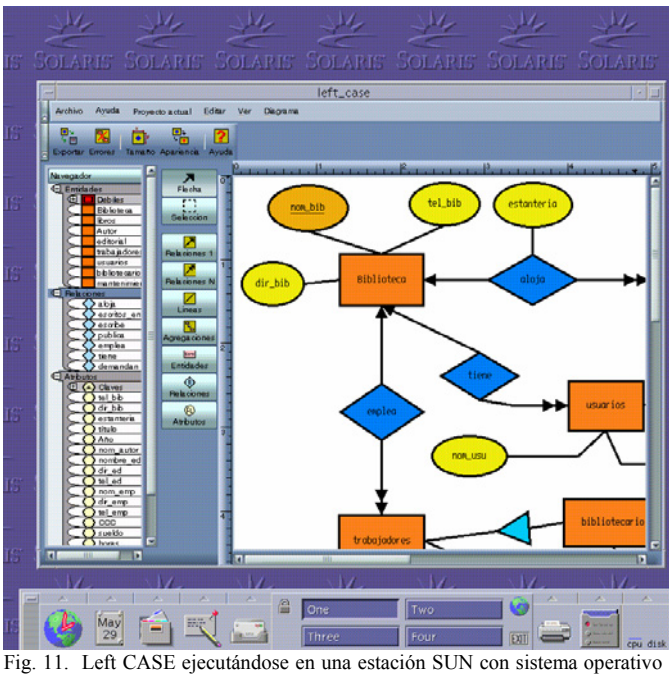

Fig. 11. Left CASE ejecutándose en una estación SUN con sistema operativo Solaris

Los proyectos estructurados permiten desarrollar modelos formados por diagramas entidad-relación (DER), diagramas de flujo de datos (DFD) y diagramas de transición de estados (DTE), estos dos últimos siguiendo la notación propuesta por Edward Yourdon [11].

Los proyectos orientados a objetos permiten desarrollo modelos de clases, utilizando para ello la notación propuesta en el lenguaje de modelado UML 1.4 [7].

La herramienta en su versión actual satisface una parte significativa de las necesidades de soporte automatizado a las técnicas de modelado en las titulaciones de Informática de Sistemas e Ingeniería en Informática de la Universidad de Salamanca.

En esta sección se va a hacer un recorrido por la funcionalidad ofrecida por Left CASE, abordándose aspectos como la selección del tipo de proyecto desde el contenedor, las técnicas soportadas tanto en un proyecto estructurado como en un proyecto orientado a objetos y la capacidad de generación automática de la documentación técnica de un proyecto.

A. Operaciones con el contenedor

Como se puede apreciar en la Figura 12, al arrancar la aplicación no aparece ninguna característica asociada a cualquiera de las técnicas soportadas, correspondiéndose este esqueleto con la noción de contenedor. Es el momento en el que se crea un nuevo proyecto o se selecciona uno existente, ver Figura 13, cuando se determina que componentes van a poder utilizarse (dependiendo de si se elige un proyecto estructurado o UML), cargándose los controles propios del componente elegido.

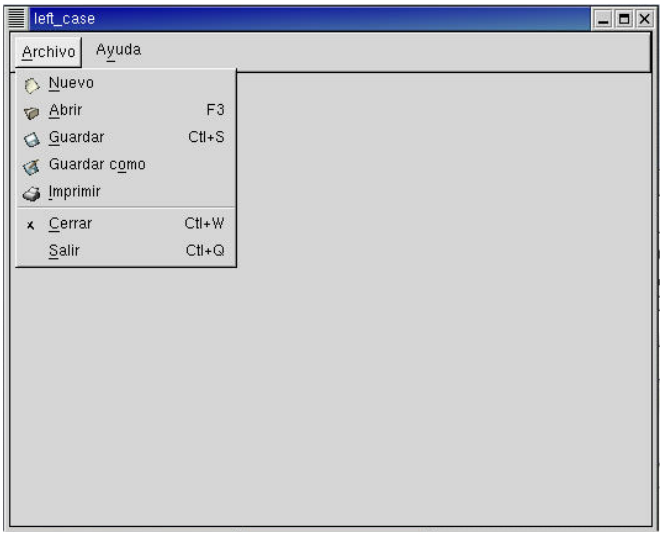

Fig. 12. Contenedor de componentes Left CASE

Así, si como se muestra en la Figura 13, se ha elegido un *proyecto estructurado*, actualmente se podría elegir entre realizar un diagrama entidad-relación, un diagrama de flujo de datos o un diagrama de transición de estados, tal y como se muestra en la Figura 14. Sin embargo, si se hubiera elegido la opción de *proyecto UML*, con la versión actual sólo se hubieran podido realizar diagramas de clases.

No obstante, la ventaja que tiene la arquitectura de esta herramienta es que en el momento que haya nuevos componentes disponibles e instalados, éstos serán reconocidos por el contenedor y ofrecidos según corresponda, bien entre los componentes asociados con proyectos estructurados, bien

Fig. 13. Selección de tipo de proyecto

con los componentes asociados con los proyectos UML.

B. Proyectos estructurados

Como se ha venido poniendo de manifiesto a lo largo de este artículo, Left CASE fue creada con el objetivo de cubrir los aspectos prácticos del modelado de sistemas software, fundamentalmente en las asignaturas relacionadas con la Ingeniería del Software.

Hoy en día, cada vez se le está dando una mayor relevancia al paradigma objetual dentro del mundo del desarrollo del software, siendo UML el estándar dentro de los lenguajes de modelado. Sin embargo, la realidad del desarrollo del software en España es que la orientación a objetos convive con los métodos clásicos de desarrollo, típicamente referenciados como métodos estructurados, como corroboran muchos Planes de Estudios en las Universidades españolas o la propia metodología Métrica Versión 3 [6], desarrollada para el Ministerio de Administraciones Públicas.

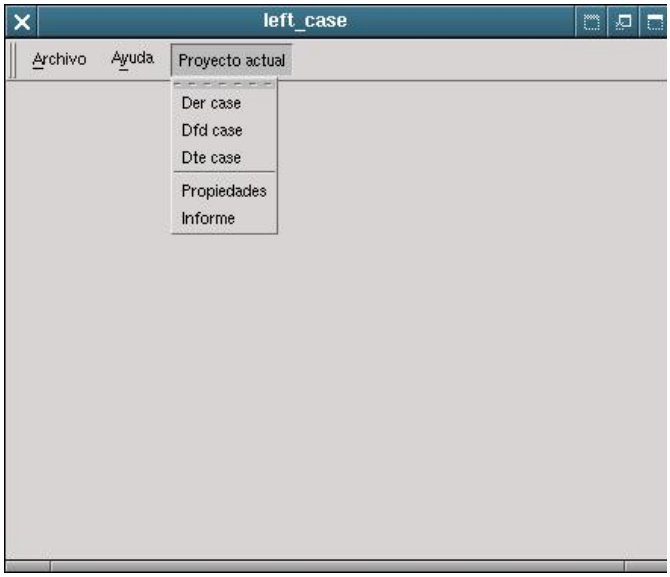

Fig. 14. Técnicas estructuradas en Left CASE

Por todo ello, no nos podíamos olvidar de las técnicas más tradicionales del modelado de sistemas software ni en el temario de la asignatura de Ingeniería del Software ni en sus

prácticas. Así, Left CASE da soporte, como se muestra en la Figura 14, a los diagramas entidad/relación extendidos (DER) con base en la notación de Chen [1], para la realización de modelos conceptuales de datos, a los diagramas de flujo de datos (DFD) con notación de Yourdon [11], para la realización de modelos funcionales, y a diagramas de transición de estados (DTE) también con notación de Yourdon [11], para la realización de especificaciones de control y modelos de comportamiento.

El componente para modelos entidad relación presenta la funcionalidad necesaria para el desarrollo de modelos conceptuales de datos. Cuando se carga este componente (al igual que sucede con el resto), la interfaz del contenedor se ve modificada con la interfaz propia del componente. Esta interfaz se muestra en la Figura 15, donde se distinguen claramente las siguientes zonas:

- **Área de diagrama**: Zona donde se desarrolla el diagrama entidad-relación, arrastrando los elementos que aparecen en la barra de diseño.
- **Barra de diseño**: Botones que representan los diferentes elementos que componen un diagrama entidad-relación.
- **Navegador**: Zona en la que aparecen representados todos los elementos del diagrama pero de una forma más compacta que facilita su gestión.
- **Barra de herramientas**: Accesos rápidos a funciones comúnmente usadas en el componente.

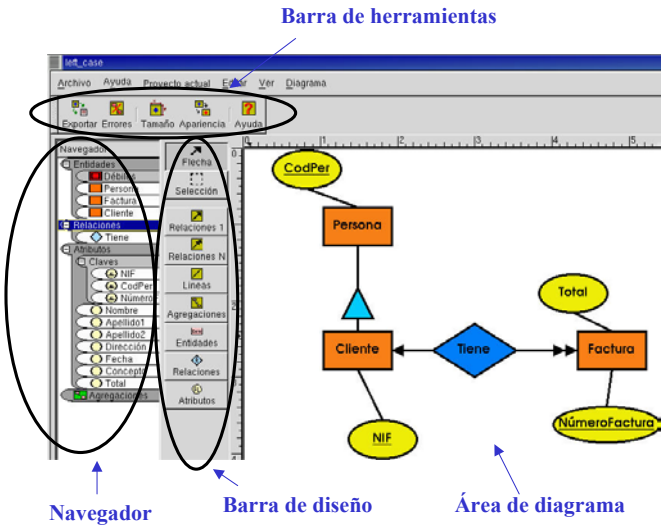

Fig. 15. Componente DER en Left CASE

Cabe destacar la opción que presenta este componente para realizar automáticamente el paso a un modelo lógico relacional desde el modelo conceptual realizado. Para que esta transformación se produzca, el diagrama entidad-relación no debe presentar ningún error; en caso de existir errores se detectarían y el usuario sería informado para que los subsanase previamente a la generación del modelo relacional. En la Figura 16 se presenta el modelo relacional equivalente al modelo de la Figura 15. Además, se ofrece la posibilidad de la generación automática de un diccionario de datos que se comparte con el componente de creación de DFDs.

Fig. 16. Modelo relacional automáticamente generado a partir del modelo de la Figura 15

El componente para la creación de diagramas de flujos de datos sigue una filosofía similar a la presentada por el componente para la creación de diagramas entidad-relación, pero con la dificultad añadida de presentar un número mayor de elementos de diseño y de soportar la creación de diagramas de flujos de datos por niveles, como se puede apreciar en las figuras 17 y 18.

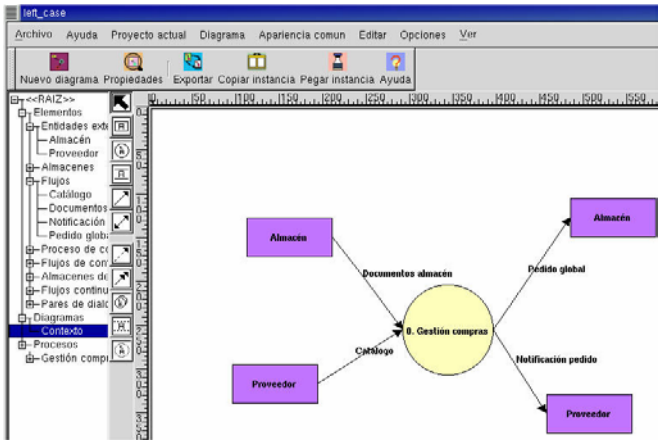

Fig. 17. Diagrama de contexto realizado con el componente DFD en Left CASE

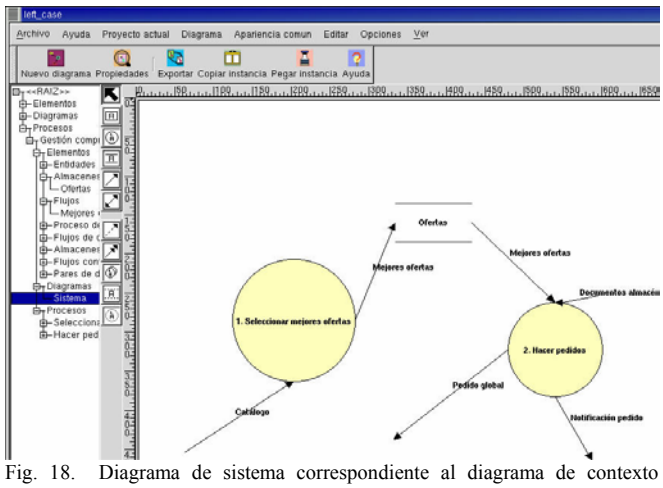

Fig. 18. Diagrama de sistema correspondiente al diagrama de contexto presentado en la Figura 17

Es importante mencionar que el componente DFD no sólo se soporta la notación de Yourdon, sino que también soporta las extensiones para sistemas de tiempo real propuestas por Ward y Mellor [9], como se puede comprobar en el ejemplo de la Figura 19.

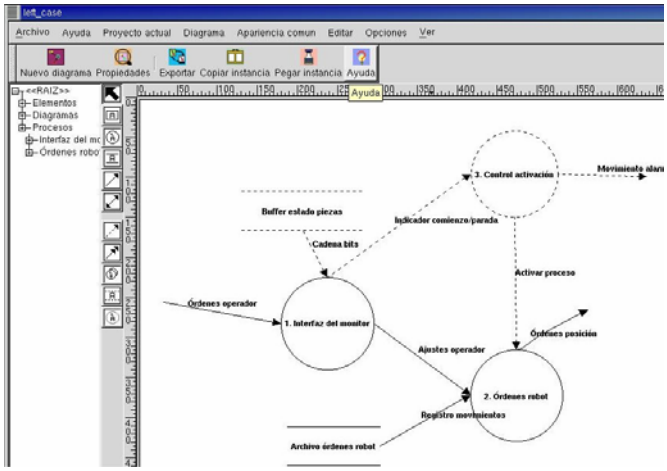

Fig. 19. DFD que emplea la notación de Ward y Mellor

Uno de los aspectos más tediosos en el proceso de creación de un DFD niveles es la gestión del diccionario de datos donde se recogen las definiciones de los flujos de datos y de los almacenes. El componente DFD dentro de Left CASE facilita en gran medida la gestión de los ítems que aparecen en el diccionario de datos, incluyendo automáticamente todas aquellas definiciones que se han introducido durante el modelado de los componentes del DFD. En la Figura 20 se muestra el diccionario de datos generado a partir de la información introducida al crear los niveles del DFD que se muestra en las figuras 17 y 18. Cabe destacar que posteriormente, a través de la interfaz mostrada en la Figura 20, el usuario puede completar la información referente a las entradas del diccionario de datos.

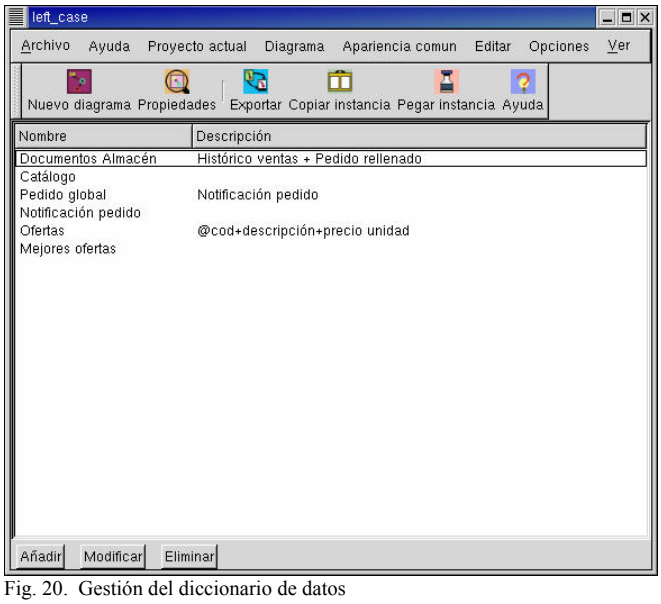

Fig. 20. Gestión del diccionario de datos

El último de los componentes que actualmente se incluyen dentro del paradigma estructurado es aquél que ofrece la posibilidad de realizar diagramas de transición de estados, utilizando para ello la notación propuesta por Edward Yourdon [11].

Este componente presenta las propiedades de edición que se han visto en los otros componentes explicados, compartiendo con el componente DFD la capacidad de crear niveles donde un estado se ve refinado por otro diagrama de transición de estados. La Figura 21 recoge un ejemplo de este tipo de diagramas realizado con Left CASE.

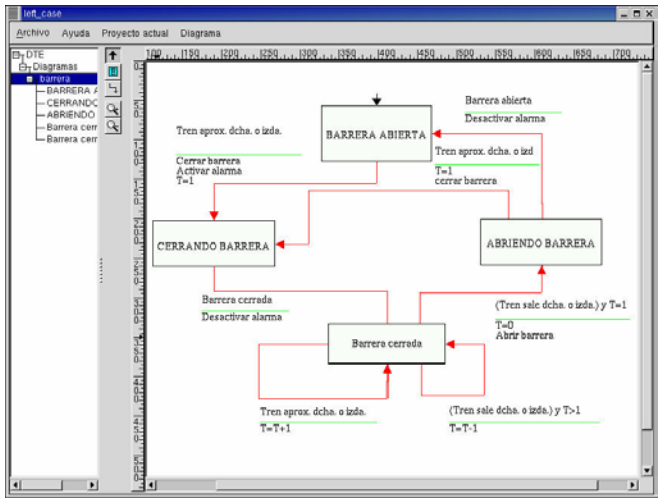

Fig. 21. Ejemplo de diagrama de transición de estados

En este componente DTE es importante destacar la interfaz que permite la asociación de acciones y condiciones a las transiciones que suponen el cambio de estado en el sistema modelado. Esta interfaz se puede observar en la Figura 22.

Fig. 22. Interfaz para la introducción de las condiciones y de las acciones asociadas a una transición

C. *Proyectos UML*

UML [7] se ha convertido en el lenguaje de modelado estándar dentro del paradigma objetual y, por tanto, su estudio es un aspecto fundamental en la Ingeniería del Software.

Así pues, queda justificada su presencia en Left CASE, pero dada su amplia diversidad de diagramas no era viable soportarlos todos desde un principio, por lo que se decidió incluir inicialmente sólo el diagrama de clases. Los motivos de esta decisión quedan justificados por las siguientes razones:

- El diagrama de clases es una de las técnicas más representativas del modelado orientado a objetos, y

la primera que se enseña a los alumnos.

- Se prefirió dar prioridad a la obtención de componentes que dieran soporte a las técnicas estructuradas frente a las técnicas de UML por estar las técnicas estructuradas menos soportadas por herramientas de libre distribución, mientras que era relativamente sencillo conseguir buenas herramientas que dieran soporte a UML.
- Las características de extensión de Left CASE no comprometían la presencia a medio plazo de nuevos componentes que soportasen más técnicas de modelado de UML.

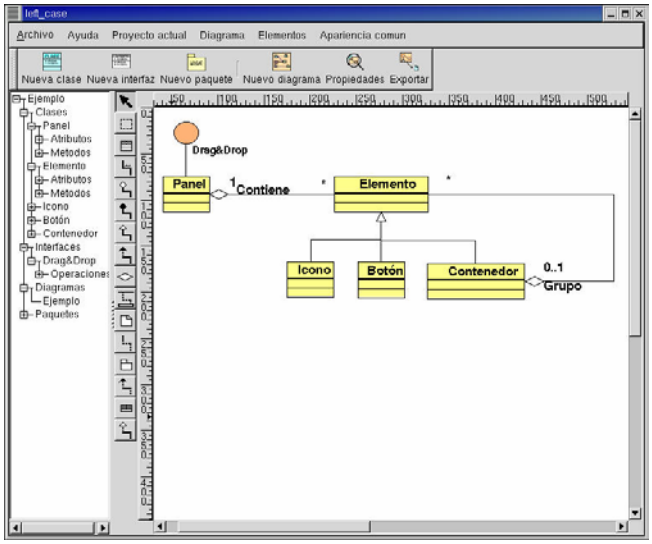

Fig. 23. Ejemplo de diagrama de clases

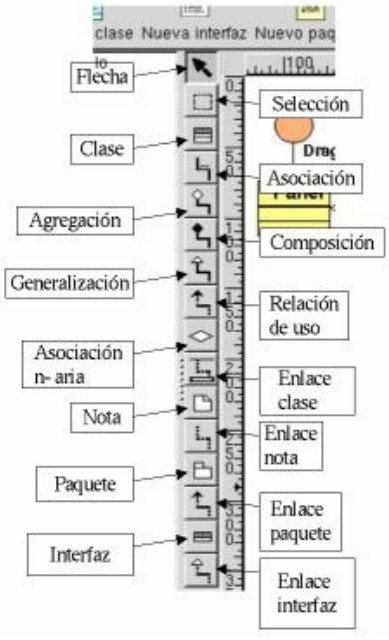

Fig. 24. Elementos de modelado soportados por el componente para la creación de diagramas de clases

El componente para la creación de diagramas de clases sigue la filosofía de edición de los componentes ya presentados, permitiendo el anidamiento a través del concepto de paquete, que es un contenedor que puede incorporar los elementos típicos de un diagrama de clases además de otros paquetes. La Figura 23 presenta un ejemplo desarrollado con este componente.

Es importante subrayar la gran variedad de elementos de modelado que incorpora este tipo de diagrama (ver Figura 24), así como la enorme cantidad de connotaciones semánticas que se asocian a cada uno de dichos elementos. Como ejemplo de esto último se presenta en la Figura 25 el cuadro de diálogo asociado a las propiedades de una relación de agregación.

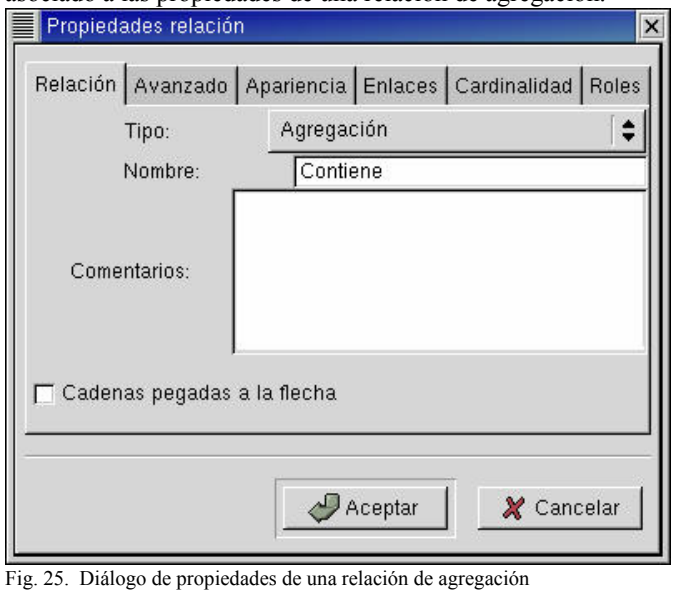

Fig. 25. Diálogo de propiedades de una relación de agregación

D. Generación de documentación

Todos los componentes desarrollados ofrecen un importante soporte a la hora de desarrollar la documentación técnica del proyecto que se esté realizando con Left CASE. Así, todos ellos permiten la impresión de los diagramas realizados, así como la exportación de los mismos a un formato gráfico *bitmap*, en este caso PNG (*Portable Network Graphics*).

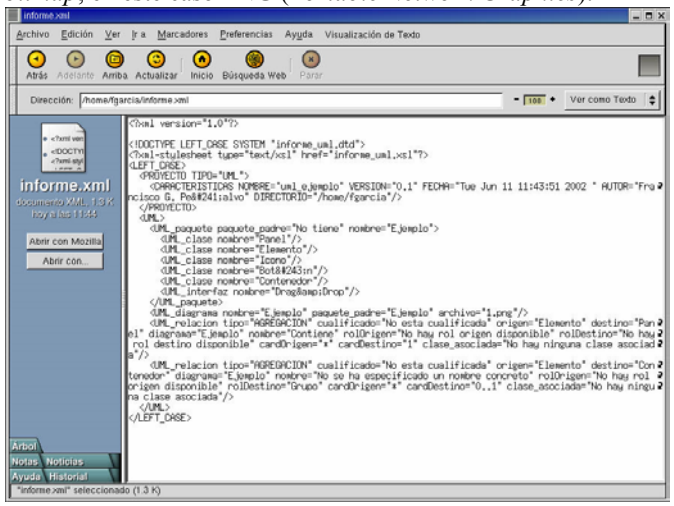

Fig. 26. Informe generado en formato XML

Sin embargo, en este sentido, la opción más interesante es la generación automática de un informe con los datos propios de los modelos desarrollados.

Los informes se generan en XML, lo que posibilita que puedan ser presentados de otras formas mediante hojas de estilo XSL. Cada componente provee a los usuarios con una de estilo adecuada para la perfecta visualización del mismo.

Como ejemplo de esta característica se muestra en la Figura 26 el resultado en XML del informe generado correspondiente al modelo de la Figura 23.

V. Conclusiones

En el presente artículo se ha presentado Left CASE, una herramienta CASE con capacidad para crear diferentes tipos de diagramas, utilizando una arquitectura basada en componentes, lo que le confiere unas características de extensibilidad y flexibilidad no disponibles en otras herramientas. Se proporciona un marco en el que resulta sencillo la adición de nuevas características, sin más que crear el correspondiente componente.

En la arquitectura de Left CASE queda claramente diferenciado, por una parte, un elemento que permite recibir otros componentes, el contenedor o *framework* de la plataforma, y por otra parte los componentes que se alojan en dicho contenedor, donde cada uno de los cuales soporta una técnica dentro del ciclo de vida del software.

Las características básicas que se ofrecen en la versión actual se pueden resumir en la creación, edición, verificación y almacenamiento de diagramas DER, DFD, DTE y diagramas de clase UML. Así mismo, se ofrece la posibilidad de imprimir los diagramas creados, exportarlos a ficheros en formato gráfico (PNG) y generar automáticamente la documentación técnica asociada a cada componente.

En cuanto a las herramientas utilizadas, cabe destacar el uso del modelo de componentes Bonobo (proyecto GNOME 1.4) que, por un lado, ha facilitado el desarrollo de la aplicación por la arquitectura utilizada y los servicios que ofrece, pero por otro lado, la falta de documentación y los fallos debidos a que se trata de sus primeras versiones estables, ha complicado el desarrollo de la aplicación.

Desde el punto de vista docente, esta plataforma CASE supone un importante soporte, en lo tocante a los aspectos de modelado, a las clases prácticas de las asignaturas relacionadas con la disciplina de Ingeniería del Software, aunque también se utiliza para la realización de los proyectos de final de carrera. Sus puntos más fuertes son su independencia de licencias comerciales, su ajuste a los contenidos actualmente impartidos en estas asignaturas y su capacidad de extensión y evolución para soportar nuevas técnicas de modelado, cuya presencian ayudará a que esta herramienta evolucione en el tiempo al igual que los contenidos de las mismas.

Durante el curso 2001-2002 esta herramienta ha sido utilizada opcionalmente por los alumnos de Ingeniería del Software para la realización de sus prácticas. Con las mejoras acontecidas, esta herramienta pasa a ser de utilización obligatoria para la realización de dichas prácticas en el presente curso académico (2002-2003), aprovechándose esta circunstancia para realizar *tests* de usabilidad para mejorar su aspecto de interacción con el usuario, además de servir para la localización de errores y su corrección.

Las líneas de trabajo futuro se centran en el desarrollo de nuevos componentes que ofrezcan soporte para nuevas técnicas de modelado, satisfaciendo así nuevas necesidades del ingeniero de software que use este entorno. Así, actualmente se está trabajando en técnicas que completen la parte relacionada con los proyectos estructurados, desarrollando nuevos componentes para la edición de casos de uso, diagramas de secuencia, diagramas de colaboración y diagramas de transición de estados, todos ellos cumpliendo las directrices marcadas por el lenguaje de modelado UML.

Otro aspecto a tener en cuenta es, desde el punto de vista de la arquitectura, la adaptación de la misma y de los componentes a la especificación GNOME 2.0.

Agradecimientos

Este trabajo ha sido parcialmente subvencionado por la Consejería de Educación y Cultura de la Junta de Castilla y León mediante el proyecto US23/02.

Referencias

[1] P. Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9-36, March 1976.

[2] F. J. García, Mª N. Moreno, A. Mª Moreno, G. González, B. Curto and F. J. Blanco, "ADAM CASE. Using upper CASE tools in Software Engineering Laboratory," in *Computers and Education: Towards an Interconnected Society*, M. Ortega and J. Bravo (Eds.). Kluwer Academic Publishers. 2001, pp. 167-175.

[3] M. J. Granger and J. C. Little, "Integrating CASE tools into the CS/CIS curriculum," in *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE '96,* ACM, 1996, pp. 130-132.

[4] M. de Icaza (1999). Bonobo. Ximian White Paper. Available: http://www.ximian.com/devzone/tech/bonobo.html.

[5] F. Losavio, A. Matteo and M. Pérez, "An Architecture for an Object-Oriented CASE Environment," *Journal of Object-Oriented Programming (JOOP)*, vol. 12, no. 6, pp. 49-54, 1999.

[6] Ministerio de Administraciones Públicas (2001). *MÉTRICA 3.0*, Volúmenes 1-3.

[7] OMG (2001, September). OMG Unified Modeling Language Specification Version 1.4. Available: http://www.celigent.com/omg/umlrtf/artifacts.htm.

[8] OMG (2002, May). Complete CORBA 2.6.1 Specification. Document formal/02-05-15, Available: http://www.omg.org.

[9] P. Ward and S. Mellor (1985). *Structured Development for Real-Time Systems*. Vols. 1-3. Yourdon Press.

[10] R. Wirfs-Brock, B. Wilkerson and L. Wiener, *Designing Object-Oriented Software*. Prentice-Hall, 1990.

[11] E. Yourdon, *Modern Structured Analysis*. Prentice Hall, 1989.