

Generación automática de documentación: XML

Víctor Manuel Ruiz Penichet

Facultad de Informática - Universidad Politécnica de Albacete

vpenichet@yahoo.es

Resumen

Cualquier programa de peso, cualquier gran sistema software, ha de ser documentado. Esto es algo que no se aplica siempre debido a diversos motivos: el programador piensa que pierde el tiempo, piensa que siempre entenderá su código, ... sin embargo, y arropado por la programación orientada a objetos o por la programación modular en general, cada vez se lleva más a la práctica para mejorar el entendimiento del sistema y la comunicación entre los diferentes miembros de la platilla.

En este trabajo veremos qué se entiende por documentación, cuáles son las partes más importantes que se deberían documentar y algunas aproximaciones para llevarlo a cabo.

Nos centraremos después en la documentación generada en XML, concretamente gracias a componentes y herramientas de última generación como Visual Studio .NET, que nos permite su generación de forma automática.

El hecho de que el trabajo se haya centrado en la generación en XML es porque gracias a esta tecnología, la documentación generada será entendible por humanos (mediante hojas de estilo XSL) y por aplicaciones.

1. Introducción

La generación de la documentación siempre ha sido una ardua tarea, que si bien es muy pesada de realizar, es muy necesaria para que los programas de gran envergadura sean fácilmente entendibles por todo un equipo de desarrollo y facilitar así la comunicación entre los distintos miembros del equipo, haciendo que todos hablen un mismo “lenguaje”.

Es también mucho más fácil para una persona que se acaba de introducir en el equipo de desarrollo. Entendiendo una buena documentación, pronto podrá ser productivo y quitará menos tiempo al resto del personal con sus lógicas preguntas sobre lo que no entiende.

En una programación basada en componentes se convierte en algo absolutamente indispensable para poder utilizar los componentes desarrollados por otros trabajadores e incluso para poder entender aquellos hechos por uno mismo pero hace ya mucho tiempo.

Pero, ¿quién genera toda esa documentación? Un programador piensa que “pierde” el tiempo documentando cada módulo, función, programa, ... que realiza y tiene tendencia a dejárselo para el final, cuando ya apenas entiende lo que hizo y es ahora cuando sí que pierde tiempo de verdad.

Gracias a programas que realizan esta pesada tarea a partir de los comentarios en el código fuente o de otra manera, la documentación sobre los sistemas se agiliza, unifica y lo más importante, se hace.

En este trabajo se hace hincapié en la generación automática de documentación en XML a partir de los comentarios escritos en el código fuente C#. El compilador de C# genera este tipo de documentación fácilmente y de forma casi simultánea a los fuentes evitando así errores e inconsistencias que podría aparecer de hacerse por un lado la documentación y por otro los ficheros fuente. Debido a que la idea es que la documentación sea fácilmente legible para los humanos, se añadirá una hoja de estilos XSL para mostrarla de forma más cómoda.

La generación de documentación en XML con C# se puede hacer desde la línea de comandos o desde Visual Studio.NET.

2. Principios en la generación de documentación

2.1 Qué se entiende por generación de documentación: qué documentar

En primer lugar deberemos tener claro lo que es exactamente la documentación de un programa de peso en Ingeniería del Software, desde dónde empezamos, o qué es lo que debemos documentar.

La documentación corre paralela con el desarrollo del sistema. Cuando defines o capturas los requerimientos del sistema, generas las hojas de especificaciones, que son la base de toda la documentación. En ellas describes a nivel funcional lo que el sistema hace.

Tomando como base las especificaciones, vas identificando clases. Cada clase debe ser documentada indicando su nombre, su utilidad (responsabilidades) y sus métodos, propiedades y atributos. Para cada uno de ellos se debe incluir su nombre, el protocolo para su uso, su función y documentar claramente cualquier efecto adicional.

Puedes agrupar una o más clases en un componente de código (una DLL, por ejemplo). En este momento debes especificar cuales son los componentes de código que empleas, y para cada uno de ellos incluir toda la información, tanto de las clases que agrupa como de los archivos fuente de los cuales depende.

Las operaciones. Para cada una de ellas su función (normalmente basta con el nombre de la operación). Si se quiere ser exhaustivo habría que definir las clases involucradas en la operación y (si no es evidente) la función que cumplen en ella. A nivel de la operación es conveniente también especificar el nivel de seguridad de la misma. Si una operación tiene algún prerrequisito, o si se debe hacer algo inmediatamente después de que termine. Estas particularidades tienen que estar claramente documentadas.

También será necesario documentar todo comentario.

Si la documentación se realiza en paralelo con el proceso de desarrollo (es conveniente que la documentación se genere durante el diseño del componente) es un trabajo natural y simplifica el proceso completo. Si se deja para el final es un dolor de cabeza espantoso, y casi siempre se hace mal. En nuestro caso pretendemos que se haga automáticamente.

2.2 Algunas posibilidades en la generación automática de documentación

A la hora de decidir implementar un módulo que nos genere la documentación de partes del sistema o del sistema en su totalidad se debe tener en cuenta qué es lo que analizaremos. Básicamente tenemos dos posibilidades:

- El código fuente del módulo a documentar.
- Un diagrama si hacemos uso de una herramienta CASE: el repositorio.

2.2.1 Desde el código fuente

Aprovecharemos el hecho de que los lenguajes de programación siempre siguen la misma estructura y la misma construcción, es decir, los comentarios se pondrán tras un determinado símbolo, el nombre de la función o procedimiento puede estar después del tipo y sus parámetros detrás del mismo, ...

Por tanto es fácil ir siguiendo el texto he ir sacando la información que nos interese conforme vamos avanzando por él.

Lo difícil puede aparecer a la hora de establecer relaciones con las otras clases pero en todo lenguaje de programación hay forma de saber con lo que se relaciona. Por ejemplo en una relación de herencia, tras el nombre de la clase suele venir el o los de las superclases de las cuales hereda atributos y funcionalidad,

así sólo tendremos que seguir recorriendo el fichero y recoger el o los nombres de dichas superclases. En cuanto a relaciones de asociación aparecen al descubrir dentro de una clase una referencia a otra.

2.2.2 Desde un diagrama de una herramienta CASE

Las ventajas que nos ofrece el uso del significado de los diagramas que se generan en un proyecto con una herramienta CASE son que la información que nos ofrece un diagrama de este estilo es mucho más clara, relaciones con otras clases (asociación, agregación) jerarquía de clases (herencia), atributos, funciones, todo esto nos lo da directamente y lo sacamos del repositorio. La grandísima ventaja que nos ofrece el "tirar" del repositorio es el hecho de que cualquier modificación que realicemos sobre nuestro modelo, gracias a características propias del repositorio, quedará reflejado automáticamente en la documentación generada con lo que nos libera del peso de tener que hacerlo nosotros manualmente. Imagina la cantidad de referencias que tiene un proyecto grande y el trabajo y dolor de cabeza que podemos evitarnos... Siempre haremos uso del repositorio para poder acceder a la información que queremos.

Del repositorio obtenemos datos:

- Nombre del autor, fecha, nombre del programa, ...
- Nombres de las clases.
- Atributos definidos en las mismas (públicos y privados).
- Operaciones tanto públicas como privadas.
- Relaciones de asociación y agregación. Grado de multiplicidad. Dependencias con otras clases.
- Herencia, grado de profundidad, número y nombre de descendientes y ascendientes, descendientes directo.

- Cualquier comentario que se haga al código.
- Dependencias de partes de código o variables con otras (esto no saldrá del repositorio porque si no lo dice el programador es difícil de saber, pero sería interesante contemplarlo).
- Incluso modificaciones posteriores.
- En definitiva todo lo que sea susceptible de ser documentado automáticamente.

Y lo mejor de todo esto es que cualquier cambio queda reflejado directamente sobre el repositorio con lo que se refleja también directamente en la generación automática de la documentación.

2.2.3 El formato de las salidas: XML

Una vez que tenemos clara la forma de acceder a lo que queremos documentar, bien sea desde el código fuente, o bien desde los diagramas generados mediante una CASE, debemos saber el formato en el que obtendremos la salida de la generación de documentación.

Usualmente las salidas son en formato .doc o .rtf que al fin y al cabo son archivos de texto con explicaciones sobre cada parte del programa y de forma bien estructurada y ordenada. Y también hay salidas en formato HTML que dan forma a un documento de hipertexto lleno de enlaces a zonas del propio documento con información relacionada de manera que queda mucho más organizado, claro, ...

El hecho de generar la documentación en formato HTML permite una navegación fácil y rápida por los contenidos creados pudiendo llegar rápidamente a la información que buscamos, por ejemplo sobre una determinada función o un determinado módulo.

Sin embargo la posibilidad que se estudia aquí es la generación de documentación en XML pues es un lenguaje

que posibilita la integración con otras herramientas o con otras aplicaciones, incluida la salida en HTML mediante hojas de estilo XSL para la presentación de los datos.

XML (eXtensible Markup Language) no es exactamente un lenguaje sino un meta-lenguaje que nos permite definir lenguajes de marcado adecuados a usos determinados. Es un meta-lenguaje diseñado para describir el contenido de lo que representa en lugar de sólo presentar la información como hace el HTML. Es decir, que HTML es un lenguaje ideal para que lo entiendan los humanos pero XML va más allá, facilitando la comprensión a los programas informáticos de lo que estamos queriendo decir.

Así se pueden crear, por ejemplo, motores de búsqueda más rápidos y eficientes, permite acceder a nuestras páginas preferidas desde un móvil, facilita el intercambio de información, el comercio electrónico, ... Es como si se quisiera poder hacerlo todo en la Web.

Ahora que tenemos etiquetado todo lo que queremos decir podemos, por ejemplo, preocuparnos del formato de la presentación para lo que el W3C está trabajando en la creación de unas hojas de estilo, las XSL (Extensible Style Lenguaje) mediante las que podemos transformar un documento XML en un HTML, RTF, PDF, ...

Según © SEEBURGER AG, XML (eXtensible Mark-up Language) es un meta lenguaje para datos estructurados. XML es la interfase del futuro. Es ideal describir o estructurar documentos de una forma que puedan ser intercambiados entre aplicaciones sin pérdida de información, especialmente en Internet. *"XML abre sus aplicaciones y provee de una forma eficiente para la integración B2B."*

En el este documento se pretende presentar un la generación automática de documentación en XML a través del compilador de C#, lenguaje desarrollado para la plataforma .NET Framework.

3. Generación de documentación XML

3.1 Escribir comentarios de documentación

Para poder generar los archivos de documentación en XML hay que ir comentando todo lo que queremos que salga en nuestro fichero fuente mediante comentarios en XML. El hecho de que se haga simultáneamente con la escritura del código hace que evitemos errores e inconsistencias como se apuntó anteriormente.

Si compiláramos un proyecto sin comentarios de documentación, pero sin embargo, generáramos la documentación oportuna, obtendríamos un archivo como se muestra en el párrafo de al lado.

```

/// <summary>
/// Clase de ejemplo de cómo escribir documentacion XML
/// </summary>
class A
{
    /// <summary>
    /// Método principal de ejemplo perteneciente a clase <see cref="A"/>
    /// </summary>
    /// <remarks>
    /// No hace nada
    /// </remarks>
    static void Main()
    {}
}

```

Y el documento XML generado automáticamente a partir de estos comentarios sería:

```

<?xml version="1.0"?>
<doc>
  <assembly>
    <name>A</name>
  </assembly>
  <members>
    <member name="T:A">
      <summary>
        Clase de ejemplo de cómo escribir documentacion XML
      </summary>
    </member>
    <member name="M:A.Main">
      <summary>
        Método principal de ejemplo perteneciente a clase <see cref="T:A"/>
      </summary>
      <remarks>
        No hace nada
      </remarks>
    </member>
  </members>
</doc>

```

```

<?xml version="1.0"?>
<doc>
  <members>
  </members>
</doc>

```

Entre las etiquetas <doc> y </doc> tendríamos los comentarios y concretamente entre <members> y </members> tendríamos comentarios de cada uno de los miembros comentados.

Si introdujéramos comentarios de documentación entonces el aspecto del archivo XML generado ya sí sería la documentación propiamente dicha.

Un trozo de código con comentarios de documentación tendría el aspecto siguiente:

La etiqueta

```
<member name = "<indicadorElemento>
: <nombreCompletamenteCalificado>">
```

nos da idea de qué tipo de elemento se documenta (<indicadorElemento>) y el nombre completamente calificado de dicho elemento (<nombreCompletamenteCalificado>). Los indicadores de elemento básicos pueden ser los que se muestran en la Figura 1.

| INDICADOR DE TIPO DE ELEMENTO | Tipo de elemento indicado |
|-------------------------------|---|
| T | Tipo de dato |
| F | Campo |
| P | Propiedad o indizador |
| M | Método (incluidos operadores y constructores) |
| E | Evento |

Figura 1. Indicadores del tipo de elemento.

Para un buen programador acostumbrado a poner comentarios en su código, introducirlos en formato de comentarios de documentación XML no supone un esfuerzo muy grande y, a cambio, se generará la documentación de forma automática con la consiguiente ventaja de ahorro de tiempo. Además usando Visual Studio.NET esta tarea es mucho más cómoda gracias a los asistentes que introducen mucho código por nosotros. Hemos pasado a la máquina la tarea de realizar la documentación, que tan pesada suele resultar.

3.2 Generación a través del compilador en línea de comandos

Para generar la documentación desde la línea de comandos a través del compilador necesitaremos usar la opción de compilación /doc:<fichero>. Un ejemplo sería el que sigue:

```
csc fich_ejemplo.cs
/doc:fich_ejemplo.xml
```

Al abrir el fichero XML generado con el navegador de Internet podremos ver un

conjunto de etiquetas que recogen toda la información de los comentarios en los archivos fuente compilados. De esta manera una persona no lee con facilidad este tipo de documentación pero podremos hacerlo mucho más legible si introducimos en el archivo XML generado una línea:

```
<?xml:stylesheet href = "<ficheroXSL>"
type="text/xsl"?>
```

El archivo lo podemos modificar desde cualquier editor de textos.

Así modificaremos la salida de nuestro archivo en XML con una hoja de estilos XSL (fichero XSL) de manera que sea más fácil de entender por una persona. Normalmente se pasa a un formato en HTML.

Como no todo el mundo sabe escribir hojas de estilo, el uso de Visual Studio.NET facilita toda esta tarea.

3.3 Generación a través de Visual Studio.NET

La utilización de Visual Studio.NET será mucho más común pues C# es base para las nuevas tecnologías .NET, es más, podríamos decir que es el lenguaje nativo de .NET Framework y Visual Studio.NET no es más que la herramienta que hace posible el uso de toda la potencia de .NET Framework.

Además, su aspecto visual hace el trabajo mucho más sencillo y agradable que la típica consola y la introducción de órdenes por la línea de comandos.

Para documentar un proyecto debemos realizar los siguientes pasos:

- I. Señalar desde el *Solution Explorer* el proyecto a documentar.
- II. Escribir el nombre del fichero XML a generar en el cuadro de texto *View* → *Property Pages* → *Configuration Properties* → *Build* → *XML Documentation File*
- III. El fichero XML generado se almacenará en la subcarpeta *bin* del proyecto y para

poder visualizarla desde el *Solution Explorer* deberemos activar el botón *Show All Files*.

- IV. Para generar de forma más legible los archivos XML podremos ir a Tools → Build Comments Web Pages donde se generarán las páginas en formato HTML.

3.4 Ejemplo de generación de código con Visual Studio.NET

En un proyecto sencillo donde pongamos un calendario en una página web y un botón sin apenas funcionalidad, el código sería el siguiente:

WebForm1.aspx.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace Calendario_Web
{
    /// <summary>
    /// Calendario para la Web
    /// </summary>
    public class WebForm1 : System.Web.UI.Page
    {
        protected System.Web.UI.WebControls.Label Label1;
        protected System.Web.UI.WebControls.TextBox TextBox1;
        protected System.Web.UI.WebControls.Button Button1;
        protected System.Web.UI.WebControls.Calendar Calendar1;

        private void Page_Load(object sender, System.EventArgs e)
        {
            // Introducir aquí el código de usuario para inicializar la
            página
        }

        #region Web Form Designer generated code
        override protected void OnInit(EventArgs e)
        {
            //
            // CODEGEN: llamada requerida por el Diseñador de Web Forms
            ASP.NET.
            //
            InitializeComponent();
            base.OnInit(e);
        }

        /// <summary>
        /// Método necesario para admitir el Diseñador, no se puede modificar
    }
}
```

```
    /// el contenido del método con el editor de código.
    /// </summary>
    private void InitializeComponent()
    {
        this.Button1.Click += new
System.EventHandler(this.Button1_Click);
        this.Load += new System.EventHandler(this.Page_Load);
    }
#endregion

    /// <summary>
    /// Botón que introduce el texto "Has pulsado el botón"
    /// </summary>
    /// <remarks>
    /// No hace nada más
    /// </remarks>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void Button1_Click(object sender, System.EventArgs e)
    {
        TextBox1.Text = "Has pulsado el botón";
    }
}
}
```

El aspecto web sería el que muestra la Figura 2:

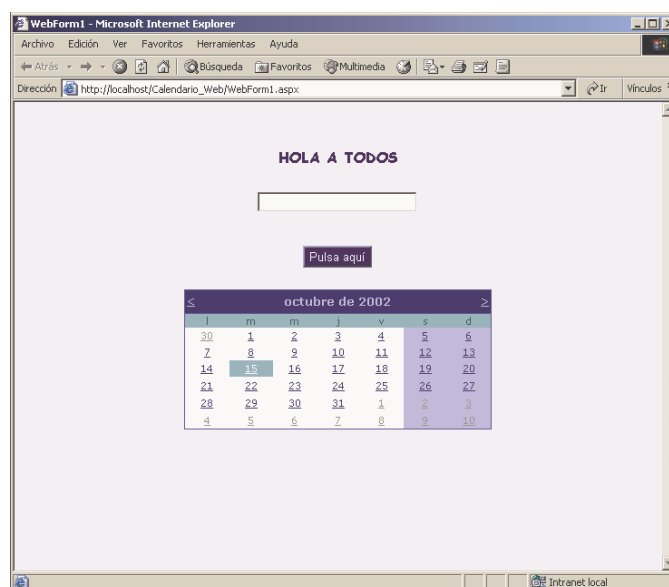


Figura 2. Aspecto de la aplicación

El archivo XML generado es el siguiente:

Calendario_Doc.xml

```
<?xml version="1.0"?>
<doc>
  <assembly>
    <name ?>Calendario_Web</name ?>
  </assembly>
  <members>
    <member name="T:Calendario_Web.Global">
      <summary>
        Descripción breve de Global.
      </summary>
    </member>
    <member name="M:Calendario_Web.Global.InitializeComponent">
      <summary>
        Método n_u99 ?e_u97 ?rio para admitir el Diseñador, no se puede
        modificar
        el contenido de_u32 ?método con el_editor de código.
      </summary>
    </member>
    <member name="T:Calendario_Web.W0bForm1">
      <summary>
        Calendario para la Web
      </summary>
    </member>
    <member name="M:Calendario_Web.WDbForm1.InitializeComponent">
      <summary>
        Método n_u99 ?e_u97 ?rio para admitir el Diseñador, no se puede
        modificar
        el contenido de_u32 ?método con el_editor de código.
      </summary>
    </member>
    <member name
    ?="M:Calendario_Web.W^bForm1.Button1_Click(System.Object,System.EventArgs)">
      <summary>
        Botón qu_u32 ?introduce el texto "Has pulsado el botón"
      </summary>
      <remarks>
        No hace nada más
      </remarks>
      <param name="sender"></param>
      <param name="e"></param>
    </member>
  </members>
</doc>
```

Y el aspecto en HTML desde el Explorador de Soluciones sería como muestra la Figura 3:

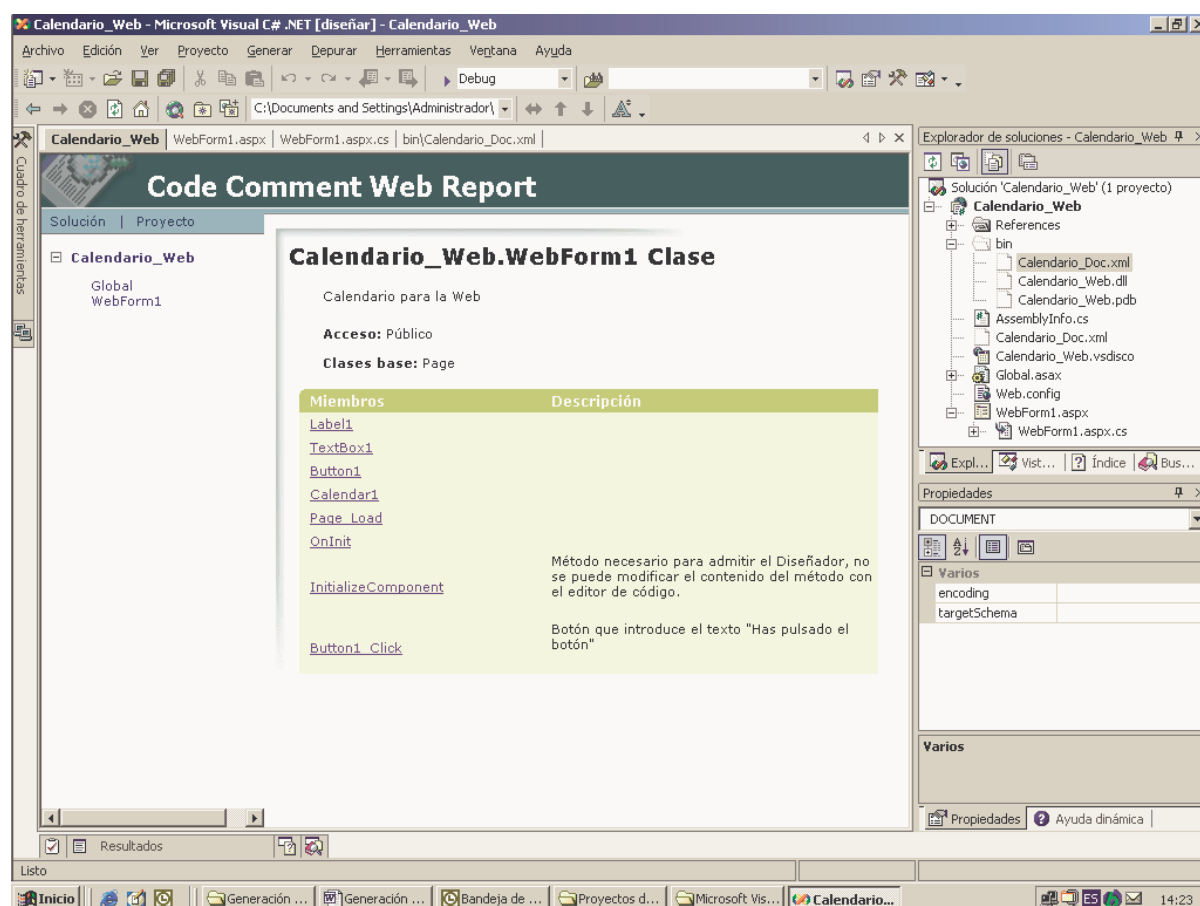


Figura 3. Formato HTML de la documentación desde el Explorador de Soluciones

4. Conclusiones

La generación automática de documentación ya es un hecho realidad que cualquier programador debería tener en cuenta para facilitar el entendimiento de código complejo tanto en un equipo de desarrollo como para sí mismo. Ya no es una excusa la incomodidad de documentar porque como hemos visto se puede hacer, sencillamente, sobre la marcha.

Un código limpio y sobre todo bien documentado ahorra mucho tiempo, esfuerzo y, sobre todo, dinero.

Referencias

[1] D. José Antonio González Seco, El lenguaje de programación C#.

[2] Manual de Visual Studio.NET

[3]
<http://edic.lsi.uniovi.es/isoft/prcase21/prcase21.htm>

Problema: usa EasyCASE (<http://www.idtoki.com/paginas/easycase.htm>) pero genera la documentación en word y tiene detalles internos de cómo interactúa con esa herramienta para generarla. En español.

[4] Doc-O-Matic - Source Code Documentation System - Doc-O-Matic home page

Con esta herramienta podremos crear documentación automáticamente a partir del código generado y de los comentarios asociados a ese código en los formatos de Ayuda de Windows y de HTML pero no en formato WORD.

Podemos ver un ejemplo aquí: <http://www.doc-o-matic.com/example.html>

[5]

http://argouml.tigris.org/project_download.html ArgoUML, versión estable.

[6] <http://argouml.tigris.org/v07/> Código fuente de ArgoUml y cómo puede ser utilizado

[7] System Architect® :

http://www.popkin.com/customers/customer_service_center/downloads/downloads.htm

[8] <http://sdt.cern.ch/Soda/> Hay que ir a las oficinas a por el programa. Componente completa que realiza la generación automática de documentación

[9]

<http://www.stack.nl/~dimitri/doxygen/index.html> (Doxygen es una herramienta para generar la documentación automáticamente pero falta lo que es la documentación (en modo de análisis). De todos modos no es una componente CASE.

[10]

<http://www.ibium.com/alf/xml/index.asp>, Alfredo Reino, alfredo@ibium.com

[11]

<http://www.programacion.com/html/xml/>

Algunas herramientas

GeneXus es una herramienta para desarrollo de aplicaciones que cubre todo el ciclo de vida: diseño, generación, modificación y documentación de bases de datos y programas.
<http://www.genexus.es/compania/descgx.htm>

Trans es el interfaz natural del programador con MultiBase. Un entorno de desarrollo de aplicaciones de muy alta productividad que ofrece al programador, en un único conjunto, todos los elementos necesarios para escribir el software más avanzado.

<http://www.transtools.com/producto/multiba1.htm>

TechWriter es una herramienta de análisis de sistemas y documentación que soporta todas las versiones de XBASE. Este producto le ayuda a documentar sus programas y le ahorra tiempo y dinero al automatizar los procesos de recogida, formateo, generación y visualización de la documentación del sistema.
<http://www.abox.com/productos.asp?pid=19>

CIAO Prolog, Potente compilador GNU, que interpreta el estándar ISO de Prolog y dispone de una cantidad ingente de librerías y una herramienta para la generación automática de documentación.
<http://www.programacion.com/visitar.php?id=46>

Oracle JDeveloper es uno de los entornos visuales existentes en el mercado que permiten desarrollar aplicaciones en Java. Este tipo de herramientas permite generar código de manera automática, con el consiguiente ahorro de tiempo y esfuerzo
http://siul02.si.ehu.es/~alfredo/iso/Laboratorio1/ISOLab1_2002.htm

LANSA es el principal entorno de desarrollo de aplicaciones AS/400. Es muy sencillo de aprender y utilizar. Puede ampliar fácilmente sus aplicaciones existentes y crear y diseñar nuevas aplicaciones con rapidez. Gracias a su potencia y su portabilidad, el lenguaje de cuarta generación de LANSAs consigue un desarrollo mucho más rápido que el RPG. El mantenimiento es mucho más sencillo, con definiciones centralizadas de lógica común en el Object Repository de LANSAs.
<http://www.redesis.com/lansa.htm>

A9001 TRISTATION 1131

RATIONAL,
<http://www.rational.com/>