**Extensible Markup Language**

|  |  |
| --- | --- |
| **Extensible Markup Language (XML)** | |
| [http://upload.wikimedia.org/wikipedia/commons/thumb/6/68/XML.svg/200px-XML.svg.png](http://es.wikipedia.org/wiki/Archivo:XML.svg) | |
| [**Extensión de archivo**](http://es.wikipedia.org/wiki/Extensi%C3%B3n_de_archivo) | .xml |
| [**Tipo de MIME**](http://es.wikipedia.org/wiki/MIME) | application/xml, text/xml |
| **Desarrollado por** | [World Wide Web Consortium](http://es.wikipedia.org/wiki/World_Wide_Web_Consortium) |
| **Tipo de formato** | [Lenguaje de marcado](http://es.wikipedia.org/wiki/Lenguaje_de_marcado) |
| **Extendido de** | [SGML](http://es.wikipedia.org/wiki/Standard_Generalized_Markup_Language) |
| **Extendido a** | [XHTML](http://es.wikipedia.org/wiki/XHTML), [RSS](http://es.wikipedia.org/wiki/RSS), [Atom](http://es.wikipedia.org/wiki/Atom), ... |
| [**Estándar(es)**](http://es.wikipedia.org/wiki/Est%C3%A1ndar_internacional) | [1.0 (Fourth Edition)](http://www.w3.org/TR/2006/REC-xml-20060816/) [1.1 (Second Edition)](http://www.w3.org/TR/2006/REC-xml11-20060816/) |

**XML**, siglas en [inglés](http://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s) de *E****x****tensible* ***M****arkup* ***L****anguage* («[lenguaje de marcas](http://es.wikipedia.org/wiki/Lenguaje_de_marcas) ), es un [metalenguaje](http://es.wikipedia.org/wiki/Metalenguaje) extensible de etiquetas desarrollado por el [World Wide Web Consortium](http://es.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C). Es una simplificación y adaptación del [SGML](http://es.wikipedia.org/wiki/SGML) y permite definir la gramática de lenguajes específicos (de la misma manera que [HTML](http://es.wikipedia.org/wiki/HTML) es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son [XHTML](http://es.wikipedia.org/wiki/XHTML), [SVG](http://es.wikipedia.org/wiki/SVG), [MathML](http://es.wikipedia.org/wiki/MathML).

XML no ha nacido sólo para su aplicación en [Internet](http://es.wikipedia.org/wiki/Internet), sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una [tecnología](http://es.wikipedia.org/wiki/Tecnolog%C3%ADa) sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

## Ventajas del XML

|  |
| --- |
| * Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna. * El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan *bugs* y se acelera el desarrollo de aplicaciones. * Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicacio |

## Estructura de un documento XML

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de pedazos de información. Ejemplos son un tema musical, que se compone de compases, que están formados a su vez por notas. Estas partes se llaman *elementos*, y se las señala mediante [etiquetas](http://es.wikipedia.org/wiki/Lenguaje_de_marcado).

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma **<nombre>**, donde *nombre* es el nombre del elemento que se está señalando.

A continuación se muestra un ejemplo para entender la estructura de un documento XML:

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE Edit\_Mensaje SYSTEM "Lista\_datos\_mensaje.dtd"

[<!ELEMENT Edit\_Mensaje (Mensaje)\*>]>

<Edit\_Mensaje>

<Mensaje>

<Remitente>

<Nombre>Nombre del remitente</Nombre>

<Mail> Correo del remitente </Mail>

</Remitente>

<Destinatario>

<Nombre>Nombre del destinatario</Nombre>

<Mail>Correo del destinatario</Mail>

</Destinatario>

<Texto>

<Asunto>

Este es mi documento con una estructura muy sencilla

no contiene atributos ni entidades....

</Asunto>

<Parrafo>

Este es mi documento con una estructura muy sencilla

no contiene atributos ni entidades....

</Parrafo>

</Texto>

</Mensaje>

</Edit\_Mensaje>

Aquí está el ejemplo de código del DTD del documento "Edit\_Mensaje":

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- Este es el DTD de Edit\_Mensaje -->

<!ELEMENT Mensaje (Remitente, Destinatario, Texto)\*>

<!ELEMENT Remitente (Nombre, Mail)>

<!ELEMENT Nombre (#PCDATA)>

<!ELEMENT Mail (#PCDATA)>

<!ELEMENT Destinatario (Nombre, Mail)>

<!ELEMENT Nombre (#PCDATA)>

<!ELEMENT Mail (#PCDATA)>

<!ELEMENT Texto (Asunto, Parrafo)>

<!ELEMENT Asunto (#PCDATA)>

<!ELEMENT Parrafo (#PCDATA)>

### Documentos XML bien formados

Los documentos denominados como "bien formados" (del inglés *well formed*) son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier [analizador sintáctico](http://es.wikipedia.org/wiki/Parser) (*parser*) que cumpla con la norma. Se separa esto del concepto de validez que se explica más adelante.

|  |
| --- |
| * Los **documentos han de seguir una estructura** estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados. * Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial. * Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles. * El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML. * Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común. * Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos "entendibles" por las personas. |

### Partes de un documento XML

Un documento XML está formado por el prólogo y por el cuerpo del documento.

#### Prólogo

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.

El prólogo de un documento XML contiene:

* Una declaración XML. Es la sentencia que declara al documento como un documento XML.
* Una declaración de tipo de documento. Enlaza el documento con su [DTD](http://es.wikipedia.org/wiki/DTD) (definición de tipo de documento), o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
* Uno o más comentarios e instrucciones de procesamiento.

#### Cuerpo

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener un y solo un elemento raíz, característica indispensable también para que el documento esté bien formado.

### Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

### Atributos

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Deben ir entre comillas.

### Entidades predefinidas

Entidades para representar caracteres especiales para que, de esta forma, no sean interpretados como marcado en el procesador XML.

### Secciones CDATA

Es una construcción en XML para especificar datos utilizando cualquier carácter sin que se interprete como marcado XML. Solo se utiliza en los atributos. No confundir con 2(#PCDATA) que es para los elementos. Permite que caracteres especiales no rompan la estructura. Ej:

<![CDATA[ contenido especial: áéíóúñ& ]]>

### Comentarios

Comentarios a modo informativo para el programador que han de ser ignorados por el procesador.

Los comentarios en XML tienen el siguiente formato:

<!--- Esto es un comentario --->

<!-- Otro comentario -->

## Validez

Que un documento esté "bien formado" solamente se refiere a su estructura sintáctica básica, es decir, que se componga de elementos, atributos y comentarios como XML especifica que se escriban. Ahora bien, cada aplicación de XML, es decir, cada lenguaje definido con esta tecnología, necesitará especificar cuál es exactamente la relación que debe verificarse entre los distintos elementos presentes en el documento.

Esta relación entre elementos se especifica en un documento externo o definición (expresada como [DTD](http://es.wikipedia.org/wiki/DTD) *(Document Type Definition = Definición de Tipo de Documento)* o como [XSchema](http://es.wikipedia.org/wiki/XSchema)). Crear una definición equivale a crear un nuevo lenguaje de marcado, para una aplicación específica.

### Document type definition (DTD)

La DTD define los tipos de elementos, atributos y entidades permitidas, y puede expresar algunas limitaciones para combinarlos. Los documentos XML que se ajustan a su DTD son denominados válidos.

#### Declaraciones tipo elemento

Los elementos deben ajustarse a un tipo de documento declarado en una DTD para que el documento sea considerado como válido.

#### Modelos de contenido

Un modelo de contenido es un patrón que establece los subelementos aceptados, y el orden en que se aceptan.

#### Declaraciones de lista de atributos

Los atributos se usan para añadir información adicional a los elementos de un documento.

#### Tipos de atributos

* Atributos [CDATA](http://es.wikipedia.org/w/index.php?title=CDATA&action=edit&redlink=1) y [NMTOKEN](http://es.wikipedia.org/w/index.php?title=NMTOKEN&action=edit&redlink=1)
* Atributos enumerados y notaciones
* Atributos [ID](http://es.wikipedia.org/w/index.php?title=ID&action=edit&redlink=1) e [IDREF](http://es.wikipedia.org/w/index.php?title=IDREF&action=edit&redlink=1)

#### Declaración de entidades

XML hace referencia a objetos que no deben ser analizados sintácticamente según las reglas XML, mediante el uso de entidades. Las entidades pueden ser:

* Internas o externas
* Analizadas o no analizadas
* Generales o parametrizadas

#### Espacios de nombres

Los [espacios de nombres XML](http://es.wikipedia.org/wiki/Espacios_de_nombres_XML) permiten separar semánticamente los elementos que forman un documento XML.

### XML Schemas ([[XSD] XML\_Schema])

Un [Schema](http://es.wikipedia.org/wiki/Schema) es algo similar a un DTD. Define qué elementos puede contener un documento XML, cómo están organizados y qué atributos y de qué tipo pueden tener sus elementos.

#### Ventajas de los Schemas frente a los DTDs

* Usan sintaxis de XML, al contrario que los DTDs.
* Permiten especificar los tipos de datos.
* Son extensibles.

## Herramientas para trabajar con documentos XML

De hecho cualquier procesador de texto, que sea capaz de producir archivos txt es capaz de generar XML, aunque en los entornos de desarrollo como [Eclipse](http://es.wikipedia.org/wiki/Eclipse_%28software%29) o Visual Studio, se facilita, ya que reconoce los formatos y ayuda a generar un XML bien formado.

# DTD

Siglas en inglés de **Document Type Definition**. La definición de tipo de documento (DTD) es una descripción de estructura y sintaxis de un documento [XML](http://es.wikipedia.org/wiki/XML) o [SGML](http://es.wikipedia.org/wiki/SGML). Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD. De esta forma, dichos documentos, pueden ser validados, conocen la estructura de los elementos y la descripción de los datos que trae consigo cada documento, y pueden además compartir la misma descripción y forma de validación dentro de un grupo de trabajo que usa el mismo tipo de información.

## Definición

La DTD es una definición, en un documento [SGML](http://es.wikipedia.org/wiki/SGML) o [XML](http://es.wikipedia.org/wiki/XML), que especifica restricciones en la estructura y sintaxis del mismo. La DTD se puede incluir dentro del archivo del documento, pero normalmente se almacena en un fichero [ASCII](http://es.wikipedia.org/wiki/ASCII) de texto separado. La sintaxis de las DTD para SGML y XML es similar pero no idéntica.

La definición de una DTD especifica la sintaxis de una aplicación de SGML o XML, que puede ser un estándar ampliamente utilizado como [XHTML](http://es.wikipedia.org/wiki/XHTML) o una aplicación local.

## ¿Qué describe una DTD?

Las DTD se emplean generalmente para determinar la estructura de un documento mediante etiquetas (en inglés *tags*) [XML](http://es.wikipedia.org/wiki/XML) o [SGML](http://es.wikipedia.org/wiki/SGML). Una DTD describe:

* **Elementos**: indican qué etiquetas son permitidas y el contenido de dichas etiquetas.
* **Estructura**: indica el orden en que van las etiquetas en el documento.
* **Anidamiento**: indica qué etiquetas van dentro de otras.

## Ejemplos

* Un ejemplo de una DTD XML muy simple, para describir una lista de personas:

<!ELEMENT lista\_de\_personas (persona\*)>

<!ELEMENT persona (nombre, fechanacimiento?, sexo?, numeroseguridadsocial?)>

<!ELEMENT nombre (#PCDATA) >

<!ELEMENT fechanacimiento (#PCDATA) >

<!ELEMENT sexo (#PCDATA) >

<!ELEMENT numeroseguridadsocial (#PCDATA)>

Observándolo línea a línea nos dice:

1. <lista\_de\_personas> es un nombre de elemento válido. El \* indica que puede haber 0 o más elementos de persona.
2. <persona> es un nombre de elemento válido. Éste contiene obligatoriamente el elemento nombre mientras que el resto son opcionales. Y lo son porque nos lo indica el símbolo "?".
3. <nombre> es un nombre de elemento válido. Contiene caracteres.
4. <sexo> es un nombre de elemento válido. Contiene caracteres.
5. <fechanacimiento> es un nombre de elemento válido.
6. <numeroseguridadsocial> es un nombre de elemento válido.

* Un ejemplo de un fichero [XML](http://es.wikipedia.org/wiki/XML) que hace uso de esta DTD:

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE lista\_de\_personas SYSTEM "ejemplo.dtd">

<lista\_de\_personas>

<persona>

<nombre>José García</nombre>

<fechanacimiento>25/04/1984</fechanacimiento>

<sexo>Varón</sexo>

</persona>

</lista\_de\_personas>

La DTD mostrada más arriba requiere un elemento "nombre" dentro de cada elemento "persona"; el elemento "lista\_de\_personas" es también obligatorio, pero el resto son opcionales.

Es posible renderizar este documento en un navegador habilitado para [XML](http://es.wikipedia.org/wiki/XML) (como [IE5](http://es.wikipedia.org/wiki/Internet_Explorer) o [Mozilla](http://es.wikipedia.org/wiki/Mozilla_Application_Suite)) pegando y guardando la DTD de más arriba en un archivo de texto llamado *ejemplo.dtd* y el fichero.xml a un fichero de texto denominado de forma diferente, y abriendo el archivo.xml con el navegador. Ambos ficheros deben estar guardados en el mismo directorio (o carpeta). No obstante, algunos navegadores no comprueban que un documento [XML](http://es.wikipedia.org/wiki/XML) sigue las reglas de la DTD; solamente se requieren para comprobar que la DTD es sintácticamente correcta.

## Limitaciones de la DTD

Un esquema basado en una DTD tiene bastantes limitaciones. Una DTD no permite definir elementos locales que sólo sean válidos dentro de otros elementos. Por ejemplo, si queremos tener un elemento <Manager> que describa al gestor de una compañía o al de una delegación, y la definición de Manager es diferente en cada caso, con una DTD tendríamos que crear los elementos “CompanyManager” y “DelegationManager” para evitar el conflicto de nombres. Es decir, la falta de jerarquía en una DTD obliga a introducir una jerarquía a base de guiones o puntos en el espacio de nombres ([Namespace](http://es.wikipedia.org/wiki/Namespace" \o "Namespace)). En una DTD es poco flexible la definición de elementos con contenido mixto, es decir, que incluyan otros elementos además de texto. Además no es posible indicar a qué tipo de dato (número, fecha, moneda) ha de corresponder un atributo o el texto de un elemento.

La necesidad de superar estas limitaciones propicia la aparición de otros lenguajes de esquema como [XML Schema](http://es.wikipedia.org/wiki/XML_Schema), herramientas más completas de descripción que son una alternativa a las DTD.

# XML Schema

**XML Schema** es un [lenguaje de esquema](http://es.wikipedia.org/wiki/Lenguaje_de_esquema) utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje [XML](http://es.wikipedia.org/wiki/XML). Se consigue así una percepción del tipo de documento con un nivel alto de abstracción. Fue desarrollado por el [World Wide Web Consortium](http://es.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C) y alcanzó el nivel de recomendación en mayo de 2001.

## Componentes

XML Schema es un lenguaje de esquema escrito en [XML](http://es.wikipedia.org/wiki/XML), basado en la gramática y pensado para proporcionar una mayor potencia expresiva que las [DTD](http://es.wikipedia.org/wiki/DTD), menos capaces al describir los documentos a nivel formal.

Los documentos esquema (usualmente con extensión .xsd de *XML Schema Definition* (XSD)) se concibieron como una alternativa a las DTD, más complejas, intentando superar sus puntos débiles y buscar nuevas capacidades a la hora de definir estructuras para documentos XML. La principal aportación de XML Schema es el gran número de [tipos de datos](http://es.wikipedia.org/wiki/Tipos_de_datos) que incorpora. De esta manera, XML Schema aumenta las posibilidades y funcionalidades de aplicaciones de procesado de datos, incluyendo tipos de datos complejos como fechas, números y strings.

### Tipos de componentes

los esquemas.

XML Schema supera muchas de las limitaciones y debilidades de las DTDs. Fue diseñado completamente alrededor de [namespaces](http://es.wikipedia.org/wiki/XML_Schema#Namespaces) y soporta tipos de datos típicos de los [lenguajes de programación](http://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n), como también tipos personalizados simples y complejos. Un esquema se define pensando en su uso final.

## Namespaces

La programación en Schema XML se basa en [Namespaces](http://es.wikipedia.org/wiki/Namespace). Podemos encontrar una analogía entre éstos y los llamados *packages* en [Java](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java). Cada Namespace contiene elementos y atributos que están estrechamente relacionados con el Namespace. Así, a la hora de definir un elemento o un atributo de un Namespace, siempre se creará una conexión entre los diferentes campos de éste. Además, esta forma de trabajar nos permite relacionar elementos que no están en el mismo Namespace.

Después de escribir un Schema XML se puede confirmar la correcta realización mediante la validación de esquemas XML: [Validación XML](http://es.wikipedia.org/wiki/Validaci%C3%B3n_XML).

## Ejemplo

Un ejemplo de la estructura de un documento esquema vacío sería el siguiente:

<?xml version="1.0" encoding="ISO-8859-1"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="0.1" xml:lang="es">

</xsd:schema>

Un ejemplo de definición con XML Schema sería el siguiente:

<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3c.org/2001/XMLSchema">

<xsd:element name="Libro">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Título" type="xsd:string"/>

<xsd:element name="Autores" type="xsd:string" maxOccurs="10"/>

<xsd:element name="Editorial" type="xsd:string"/>

</xsd:sequence>

<xsd:attribute name="precio" type="xsd:double"/>

</xsd:complexType>

</xsd:element>

</xsd:schema>

Podemos ver como en ambos casos se inician las declaraciones indicando la versión de [XML](http://es.wikipedia.org/wiki/XML) que se va a utilizar y la codificación que se usa. Estos dos campos son necesarios para poder interpretar el esquema. Además, en la siguiente línea de código podemos ver como se redirecciona al usuario a la página que ofrece las pautas de creación de XML Schema en las que se basará la descripción del esquema.

El elemento raíz se llama “Libro” y tiene tres hijos (elementos anidados) y un atributo. Los hijos son “Titulo”,”Editorial” que deben aparecer una vez y “Autores” que puede aparecer de una a diez veces. El hecho de que estén agrupados en una secuencia indica que los elementos deben aparecer en orden, es decir, primero el “Titulo”, luego los “Autores” y por último la “Editorial”. Los tres elementos son de tipo *string*. El atributo de libro se llama “precio” y es de tipo *double*.

# Simple API for XML

**SAX** (**Simple API for XML**) is a [serial access](http://en.wikipedia.org/wiki/Serial_access) [parser](http://en.wikipedia.org/wiki/Parser) [API](http://en.wikipedia.org/wiki/Application_programming_interface) for [XML](http://en.wikipedia.org/wiki/XML). SAX provides a mechanism for reading data from an XML document. It is a popular alternative to the [Document Object Model](http://en.wikipedia.org/wiki/Document_Object_Model) (DOM).

## XML processing with SAX

A parser which implements SAX (ie, *a SAX Parser*) functions as a stream parser, with an [event-driven](http://en.wikipedia.org/wiki/Event-driven) API. The user defines a number of [callback methods](http://en.wikipedia.org/wiki/Callback_%28computer_science%29) that will be called when events occur during parsing. The SAX events include:

* XML Text nodes
* XML Element nodes
* XML Processing Instructions
* XML Comments

Events are fired when each of these XML features are encountered, and again when the end of them is encountered. XML attributes are provided as part of the data passed to element events.

SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

## Example

Given the following XML document:

<?xml version="1.0" encoding="UTF-8"?>

<RootElement param="value">

<FirstElement>

Some Text

</FirstElement>

<SecondElement param2="something">

Pre-Text <Inline>Inlined text</Inline> Post-text.

</SecondElement>

</RootElement>

This XML document, when passed through a SAX parser, will generate a sequence of events like the following:

* XML Processing Instruction, named *xml*, with attributes *version* equal to "1.0" and *encoding* equal to "UTF-8"
* XML Element start, named *RootElement*, with an attribute *param* equal to "value"
* XML Element start, named *FirstElement*
* XML Text node, with data equal to "Some Text" (note: text processing, with regard to spaces, can be changed)
* XML Element end, named *FirstElement*
* XML Element start, named *SecondElement*, with an attribute *param2* equal to "something"
* XML Text node, with data equal to "Pre-Text"
* XML Element start, named *Inline*
* XML Text node, with data equal to "Inlined text"
* XML Element end, named *Inline*
* XML Text node, with data equal to "Post-text."
* XML Element end, named *SecondElement*
* XML Element end, named *RootElement*

In fact, this may vary: the SAX specification deliberately states that a given section of text may be reported as multiple sequential text events. Thus in the example above, a SAX parser may generate a different series of events, part of which might include:

* XML Element start, named *FirstElement*
* XML Text node, with data equal to "Some "
* XML Text node, with data equal to "Text"
* XML Element end, named *FirstElement*

## Definition

Unlike [DOM](http://en.wikipedia.org/wiki/Document_Object_Model), there is no *formal* specification for SAX. The [Java](http://en.wikipedia.org/wiki/Java_%28programming_language%29) implementation of SAX is considered to be [normative](http://en.wikipedia.org/wiki/Norm_%28sociology%29)[[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)], and implementations in other languages attempt to follow the rules laid down in that implementation, adjusting for the differences in language where necessary.

## Benefits

SAX parsers have certain benefits over DOM-style parsers. The quantity of [memory](http://en.wikipedia.org/wiki/Memory_%28computers%29) that a SAX parser must use in order to function is typically much smaller than that of a DOM parser. DOM parsers must have the entire tree in memory before any processing can begin, so the amount of memory used by a DOM parser depends entirely on the size of the input data. The memory footprint of a SAX parser, by contrast, is based only on the maximum depth of the XML file (the maximum depth of the XML tree) and the maximum data stored in XML attributes on a single XML element. Both of these are always smaller than the size of the parsed tree itself.

Because of the event-driven nature of SAX, processing documents can often be faster than DOM-style parsers. Memory allocation takes time, so the larger memory footprint of the DOM is also a performance issue.

Due to the nature of DOM, streamed reading from disk is impossible. Processing XML documents larger than main memory is also impossible with DOM parsers but can be done with SAX parsers. However, DOM parsers may make use of [disk space as memory](http://en.wikipedia.org/wiki/Virtual_Memory) to evade this limit.

# Document Object Model

[](http://es.wikipedia.org/wiki/Archivo:DocumentObjectModelES.svg)

[http://es.wikipedia.org/skins-1.5/common/images/magnify-clip.png](http://es.wikipedia.org/wiki/Archivo:DocumentObjectModelES.svg)

Jerarquía del Modelo en Objetos para la representación de Documentos

El ***Document Object Model*** (una traducción al español no literal, pero apropiada, podría ser ***Modelo en Objetos para la representación de Documentos***), abreviado DOM, es esencialmente una [interfaz de programación de aplicaciones](http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) que proporciona un conjunto estándar de objetos para representar documentos [HTML](http://es.wikipedia.org/wiki/HTML) y [XML](http://es.wikipedia.org/wiki/XML), un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

A través del DOM, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente.

El responsable del DOM es el consorcio [W3C](http://es.wikipedia.org/wiki/W3C) (*World Wide Web Consortium*).

En efecto, el DOM es una [API](http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) para acceder, añadir y cambiar dinámicamente contenido estructurado en documentos con lenguajes como [ECMAScript](http://es.wikipedia.org/wiki/ECMAScript) ([Javascript](http://es.wikipedia.org/wiki/Javascript" \o "Javascript)).

## Estableciendo referencias a objetos

El DOM define la manera en que [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) y elementos se relacionan entre sí en el navegador y en el documento. Cualquier [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) adecuado para el [desarrollo](http://es.wikipedia.org/wiki/Dise%C3%B1o_web) de [página web](http://es.wikipedia.org/wiki/P%C3%A1gina_web) puede ser utilizado. En el caso de [javascript](http://es.wikipedia.org/wiki/Javascript), cada **objeto** tiene un nombre, el cual es exclusivo y único. Cuando existen más de un [objeto](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) del mismo tipo en un documento web, estos se organizan en un vector. Algunos nombres de objetos comunes son los siguientes:

|  |  |
| --- | --- |
| **Nombre** | **Traducción** |
| window | ventana |
| document | documento |
| body | cuerpo |
| div | division |
| p | parrafo |

Se le puede asignar una identificación a los [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) la cual se puede utilizar para hacer referencia a estos, por ejemplo:

....

Para hacer referencia a elementos del mismo tipo, los cuales, como se ha dicho, están organizados en un vector, se pueden utilizar puntos de la siguiente manera.

document.div[0]

document.div["Juan"]

document.div.Juan

Donde el elemento "Juan" es el primer elemento del vector de elementos del tipo <div>

En forma alternativa se puede únicamente usar el identificador del elemento

Juan

También se puede usar la función "getElementById"

document.getElementById("Juan")

## Manipulando las propiedades y funciones de objetos

Los **objetos computacionales** de la misma forma que cualquier objeto de la vida real, tienen propiedades. Algunos ejemplos de propiedades de objetos de la vida real son dimensiones, color y peso.

En la mayoría de los [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) computacionales algunas propiedades se pueden determinar de la siguiente manera:

Objeto.propiedad = valor;

//por ejemplo para el objeto "Juan"

Juan.color = rojo;

La manipulación de [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) sigue los mismos principios que en el [lenguaje de programación](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) que se esté utilizando. Una de las características de estos [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) es la **función** para la cual están diseñados, de hecho la mayoría de las ocasiones tienen más de una [función](http://es.wikipedia.org/wiki/Funci%C3%B3n). En [javascript](http://es.wikipedia.org/wiki/Javascript) muchas [funciones](http://es.wikipedia.org/wiki/Funciones) para cada uno de los [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) incluyendo el [navegador](http://es.wikipedia.org/wiki/Navegador_web) y la [ventana](http://es.wikipedia.org/wiki/Ventana) que lo contiene han sido definidas previamente, adicionalmente el usuario puede definir [funciones](http://es.wikipedia.org/wiki/Funciones) de acuerdo a sus necesidades, por ejemplo el código:

function comeLaLetraA(Texto){

var TextoNuevo = "";

while(letras en el Texto recibido){

//lee la siguiente letra

//si esta letra no es "a" añadela al nuevo texto

}

return TextoNuevo;

}

Añade una nueva función al documento utilizado para crear una página web.

## Manipulando Eventos

Un **evento** desde del punto de vista computacional ocurre cuando alguna situación cambia en la computadora, como por ejemplo, la posición del ratón, la opresión de alguna tecla, los contenidos de alguna de las memorias, la condición de la pantalla, etc. En la creación de [páginas web](http://es.wikipedia.org/wiki/P%C3%A1gina_web) estos eventos representan la interacción de la computadora con el usuario.

Cuando algunos de estos eventos ocurren, como por ejemplo la presión de algún botón del ratón, es deseable que la computadora responda de alguna manera. Esa es la razón por la que existen **"event handlers"** (*encargados de manejar eventos*)los cuales son [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) que responden a eventos. Una manera de añadir eventos en el DOM utilizando [javascript](http://es.wikipedia.org/wiki/Javascript) es:

<element onevent="script">....</element>

Por ejemplo:

<div id="midivision" onclick=comeLaLetraA("bar")>

Aqui va otro texto

<div>

Otra forma de manipular eventos en [javascript](http://es.wikipedia.org/wiki/Javascript) al crear [páginas para la web](http://es.wikipedia.org/wiki/P%C3%A1gina_web) es tratándolos como propiedades de los elementos que forman la página, por ejemplo:

object.event = funcion;

//como puede ser:

document.mydivision.onclick = hazAlgo;

// tambien:

document.getElementById("mydivision").onclick = hazAlgo;

En el DOM se considera que un evento se origina en el exterior de la página web y se propaga de alguna manera hasta los elementos internos de la página. Un posible ejemplo de esta propagación es:

EVENTO → Ventana → Document → HTML → BODY → DIV → DESTINO

RESPUESTA → DIV → BODY → HTML → Document → Window → EVENTO

Siguiendo esta idea, se establecen tres etapas, **captura** la cual es cuando el evento se esta trasladando a su destino. **Blanco** que es cuando llega al blanco, o sea que llega a su destino. Este destino es el objeto en el cual se va a crear una reacción a este evento. Finalmente la etapa de **burbujeo** que es cuando el evento "regresa" a su posición original.

Ciertos objetos pueden estar al pendiente de ciertos eventos. Para hacer esto el objeto añade un "oyente de eventos" con la función **addEventListener**. Cuando el evento ocurra, alguna función determinada se lleva a cabo, en este proceso se indica en que momento la función se lleva a cabo, ya sea en la etapa de *captura* o en la etapa de *burbujeo*. Este momento se indica con la palabra **true** si debe ocurrir en la etapa de captura o **false** si debe ocurrir en la etapa de burbujeo. En [javascript](http://es.wikipedia.org/wiki/Javascript) se escribe de la siguiente manera:

objeto.addEventListener(evento, funcion, momento);

por ejemplo:

document.getElementById("mydivision").addEventListener("click", hazAlgo, false);

# Javadoc

**Javadoc** es una utilidad de [Sun Microsystems](http://es.wikipedia.org/wiki/Sun_Microsystems) para la generación de documentación de [APIs](http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) en formato [HTML](http://es.wikipedia.org/wiki/HTML) a partir de código fuente [Java](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java).

Javadoc es el estándar de la industria para documentar clases de [Java](http://es.wikipedia.org/wiki/Java). La mayoría de los [IDEs](http://es.wikipedia.org/wiki/IDE) los generan automáticamente.

## Etiquetas Javadoc

Para generar [APIs](http://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) con Javadoc han de usarse etiquetas (tag's) de [HTML](http://es.wikipedia.org/wiki/HTML) o ciertas palabras reservadas precedidas por el caracter "@".  
Estas etiquetas se escriben al principio de cada clase, miembro o método, dependiendo de qué objeto se desee describir, mediante un comentario iniciado con "/\*\*" y acabado con "\*/".  
A continuación se explican algunas de las palabras reservadas - puede verse una lista completa de las tags con su correpondiente uso en [sun.com](http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/javadoc.html#javadoctags)

***Nota 1:*** En uso explica la semántica del texto tras el tag.  
***Nota 2:*** Versión indica desde qué versión de Javadoc es válida.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag** | **Descripción** | **Uso** | **Versión** |
| @author | Nombre del desarrollador. | nombre\_autor | 1.0 |
| @deprecated | Indica que el método o clase es antigua y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores. | descripción | 1.0 |
| @param | Definición de un parámetro de un método, es requerido para todos los parámetros del método. | nombre\_parametro descripción | 1.0 |
| @return | Informa de lo que devuelve el método, no se puede usar en constructores o métodos "void". | descripción | 1.0 |
| @see | Asocia con otro método o clase. | referencia (#método(); clase#método(); paquete.clase; paquete.clase#método()). | 1.0 |
| @throws | Excepción lanzada por el método | nombre\_clase descripción | 1.2 |
| @version | Versión del método o clase. | versión | 1.0 |

## Ejemplo

Un ejemplo de un Javadoc de un método.

/\*\*

\* Inserta un título en la clase descripción.

\* Al ser el título obligatorio, si es nulo o vacío se lanzará

\* una excepción.

\*

\* @param titulo El nuevo título de la descripción.

\* @throws IllegalArgumentException Si titulo es null, está vacío o contiene solo espacios.

\*/

public void setTitulo (String titulo) throws IllegalArgumentException

{

if (titulo == null || titulo.trim().equals(""))

{

throw new Exception ("El título no puede ser nulo o vacío");

}

else

{

this.titulo = titulo;

}

}

# Java Remote Method Invocation

**RMI** (***Java Remote Method Invocation***) es un mecanismo ofrecido por [Java](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java) para invocar un [método](http://es.wikipedia.org/wiki/M%C3%A9todos_%28programaci%C3%B3n_orientada_a_objetos%29) de manera remota. Forma parte del entorno estándar de ejecución de Java y provee de un mecanismo simple para la comunicación de servidores en aplicaciones distribuidas basadas exclusivamente en Java. Si se requiere comunicación entre otras tecnologías debe utilizarse [CORBA](http://es.wikipedia.org/wiki/CORBA) o [SOAP](http://es.wikipedia.org/wiki/SOAP) en lugar de RMI.

RMI se caracteriza por la facilidad de su uso en la programación por estar específicamente diseñado para Java; proporciona paso de objetos por referencia (no permitido por SOAP), [recolección de basura](http://es.wikipedia.org/wiki/Recolecci%C3%B3n_de_basura) distribuida (Garbage Collector distribuido) y paso de tipos arbitrarios (funcionalidad no provista por CORBA).

Por medio de RMI, un programa Java puede exportar un [objeto](http://es.wikipedia.org/wiki/Objeto_%28programaci%C3%B3n%29), lo que significa que éste queda accesible a través de la red y el programa permanece a la espera de peticiones en un [puerto TCP](http://es.wikipedia.org/wiki/Transmission_Control_Protocol). A partir de este momento, un cliente puede conectarse e invocar los métodos proporcionados por el objeto.

La invocación se compone de los siguientes pasos:

* Encapsulado (marshalling) de los parámetros (utilizando la funcionalidad de [serialización](http://es.wikipedia.org/wiki/Serializaci%C3%B3n) de Java).
* Invocación del método (del cliente sobre el servidor). El invocador se queda esperando una respuesta.
* Al terminar la ejecución, el servidor serializa el valor de retorno (si lo hay) y lo envía al cliente.
* El código cliente recibe la respuesta y continúa como si la invocación hubiera sido local.

## Contexto

Desde la versión 1.1 de JDK, Java tiene su propio ORB: RMI (Remote Method Invocation). A pesar de que RMI es un ORB en el sentido general, no es un modelo compatible con CORBA. RMI es nativo de Java, es decir, es una extensión al núcleo del lenguaje. RMI depende totalmente del núcleo de la Serialización de Objetos de Java, así como de la implementación tanto de la portabilidad como de los mecanismos de carga y descarga de objetos en otros sistemas, etc.

El uso de RMI resulta muy natural para todo aquel programador de Java ya que éste no tiene que aprender una nueva tecnología completamente distinta de aquella con la cual desarrollará. Sin embargo, RMI tiene algunas limitaciones debido a su estrecha integración con Java, la principal de ellas es que esta tecnología no permite la interacción con aplicaciones escritas en otro lenguaje.

RMI como extensión de Java, es una tecnología de programación, fue diseñada para resolver problemas escribiendo y organizando código ejecutable. Así RMI constituye un punto específico en el espacio de las tecnologías de programación junto con C, C++, Smalltalk, etc

## Arquitectura [[editar](http://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&action=edit&section=2)]

La arquitectura RMI puede verse como un modelo de cuatro capas:

### Primera capa [[editar](http://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&action=edit&section=3)]

La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. Cualquier aplicación que quiera que sus métodos estén disponibles para su acceso por clientes remotos debe declarar dichos métodos en una [interfaz](http://es.wikipedia.org/wiki/Interfaz_%28Java%29) que extienda java.rmi.Remote. Dicha interfaz se usa básicamente para "marcar" un objeto como remotamente accesible. Una vez que los métodos han sido implementados, el objeto debe ser exportado. Esto puede hacerse de forma implícita si el objeto extiende la clase UnicastRemoteObject (paquete java.rmi.server), o puede hacerse de forma explícita con una llamada al método exportObject() del mismo paquete.

### Segunda capa [[editar](http://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&action=edit&section=4)]

La capa 2 es la capa proxy, o capa stub-skeleton. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.

### Tercera capa [[editar](http://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&action=edit&section=5)]

La capa 3 es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo stream (stream-oriented connection) desde la capa de transporte.

### Cuarta Capa [[editar](http://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&action=edit&section=6)]

La capa 4 es la de transporte. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (Java Remote Method Protocol), que solamente es "comprendido" por programas Java.

## Elementos [[editar](http://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&action=edit&section=7)]

Toda aplicación RMI normalmente se descompone en 2 partes:

* Un servidor, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
* Un cliente, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

## Ejemplo [[editar](http://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&action=edit&section=8)]

Un servidor RMI consiste en definir un objeto remoto que va a ser utilizado por los clientes. Para crear un objeto remoto, se define una [interfaz](http://es.wikipedia.org/wiki/Interfaz_%28Java%29), y el objeto remoto será una clase que implemente dicha interfaz. Veamos como crear un servidor de ejemplo mediante 3 pasos:

* Definir el interfaz remoto. Cuando se crea un interfaz remoto:
  + El interfaz debe ser público.
  + Debe extender (heredar de) el interfaz java.rmi.Remote, para indicar que puede llamarse desde cualquier máquina virtual Java.
  + Cada método remoto debe lanzar la excepción java.rmi.RemoteException en su cláusula throws, además de las excepciones que pueda manejar.

public interface MiInterfazRemoto extends java.rmi.Remote

{

public void miMetodo1() throws java.rmi.RemoteException;

public int miMetodo2() throws java.rmi.RemoteException;

}

* Implementar el interfaz remoto

public class MiClaseRemota

extends java.rmi.server.UnicastRemoteObject

implements MiInterfazRemoto

{

public MiClaseRemota() throws java.rmi.RemoteException

{

// Código del constructor

}

public void miMetodo1() throws java.rmi.RemoteException

{

// Aquí ponemos el código que queramos

System.out.println("Estoy en miMetodo1()");

}

public int miMetodo2() throws java.rmi.RemoteException

{

return 5; // Aquí ponemos el código que queramos

}

public void otroMetodo()

{

// Si definimos otro método, éste no podría llamarse

// remotamente al no ser del interfaz remoto

}

public static void main(String[] args)

{

try

{

MiInterfazRemoto mir = new MiClaseRemota();

java.rmi.Naming.rebind("//" + java.net.InetAddress.getLocalHost().getHostAddress() +

":" + args[0] + "/PruebaRMI", mir);

}

catch (Exception e)

{

}

}

}

* Como se puede observar, la clase MiClaseRemota implementa el interfaz MiInterfazRemoto que hemos definido previamente. Además, hereda de UnicastRemoteObject, que es una clase de Java que podemos utilizar como superclase para implementar objetos remotos.
* Luego, dentro de la clase, definimos un constructor (que lanza la excepción RemoteException porque también la lanza la superclase UnicastRemoteObject), y los métodos de la/las interfaz/interfaces que implemente.
* Finalmente, en el método main, definimos el código para crear el objeto remoto que se quiere compartir y hacer el objeto remoto visible para los clientes, mediante la clase Naming y su método rebind(...).

Nota: Hemos puesto el método main() dentro de la misma clase por comodidad. Podría definirse otra clase aparte que fuera la encargada de registrar el objeto remoto.

* Compilar y ejecutar el servidor

Ya tenemos definido el servidor. Ahora tenemos que compilar sus clases mediante los siguientes pasos:

* + Compilamos el interfaz remoto. Además lo agrupamos en un fichero JAR para tenerlo presente tanto en el cliente como en el servidor:

javac MiInterfazRemoto.java

jar cvf objRemotos.jar MiInterfazRemoto.class

* + Luego, compilamos las clases que implementen los interfaces. Y para cada una de ellas generamos los ficheros Stub y Skeleton para mantener la referencia con el objeto remoto, mediante el comando rmic:

set CLASSPATH=%CLASSPATH%:.\objRemotos.jar:.

javac MiClaseRemota.java

rmic -d . MiClaseRemota

Observamos en nuestro directorio de trabajo que se han generado automáticamente dos ficheros.class (MiClaseRemota\_Skel.class y MiClaseRemota\_Stub.class) correspondientes a la capa stub-skeleton de la arquitectura RMI.

* Para ejecutar el servidor, seguimos los siguientes pasos:
  + Se arranca el registro de RMI para permitir registrar y buscar objetos remotos. El registro se encarga de gestionar un conjunto de objetos remotos a compartir, y buscarlos ante las peticiones de los clientes. Se ejecuta con la aplicación rmiregistry distribuida con Java, a la que le podemos pasar opcionalmente el puerto por el que conectar (por defecto, el 1099).

En el caso de Windows, se debe ejecutar:

start rmiregistry 1234

Y en el caso de Linux:

rmiregistry &

* + Por último, se lanza el servidor:

java -Djava.rmi.server.hostname=127.0.0.1 MiClaseRemota 1234

* Crear un cliente RMI

Vamos ahora a definir un cliente que accederá a el/los objeto/s remoto/s que creemos. Para ello seguimos los siguientes pasos:

* + Definir la clase para obtener los objetos remotos necesarios

La siguiente clase obtiene un objeto de tipo MiInterfazRemoto, implementado en nuestro servidor:

public class MiClienteRMI

{

private MiClienteRMI(){};

public static void main(String[] args)

{

try

{

MiInterfazRemoto mir = (MiInterfazRemoto)java.rmi.Naming.lookup("//" +

args[0] + ":" + args[1] + "/PruebaRMI");

// Imprimimos miMetodo1() tantas veces como devuelva miMetodo2()

for (int i=1;i<=mir.miMetodo2();i++)

mir.miMetodo1();

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

Como se puede observar, simplemente consiste en buscar el objeto remoto en el registro RMI de la máquina remota. Para ello usamos la clase Naming y su método lookup(...).

* + Compilar y ejecutar el cliente

Una vez que ya tenemos definido el cliente, para compilarlo hacemos:

set CLASSPATH=%CLASSPATH%;.\objRemotos.jar;.

javac MiClienteRMI.java

Luego, para ejecutar el cliente hacemos:

java MiClienteRMI 127.0.0.1 1234

Se debe poder acceder al fichero Stub de la clase remota. Para ello, o bien lo copiamos al cliente y lo incluimos en su CLASSPATH, o lo eliminamos del CLASSPATH del servidor e incluimos su ruta en el java.rmi.codebase del servidor (si no se elimina del CLASSPATH del servidor, se ignorará la opción java.rmi.codebase, y el cliente no podrá acceder al Stub).. Si echamos un vistazo a la ventana donde está ejecutándose el servidor RMI, veremos como se ha encontrado el objeto remoto y ejecutado sus métodos.