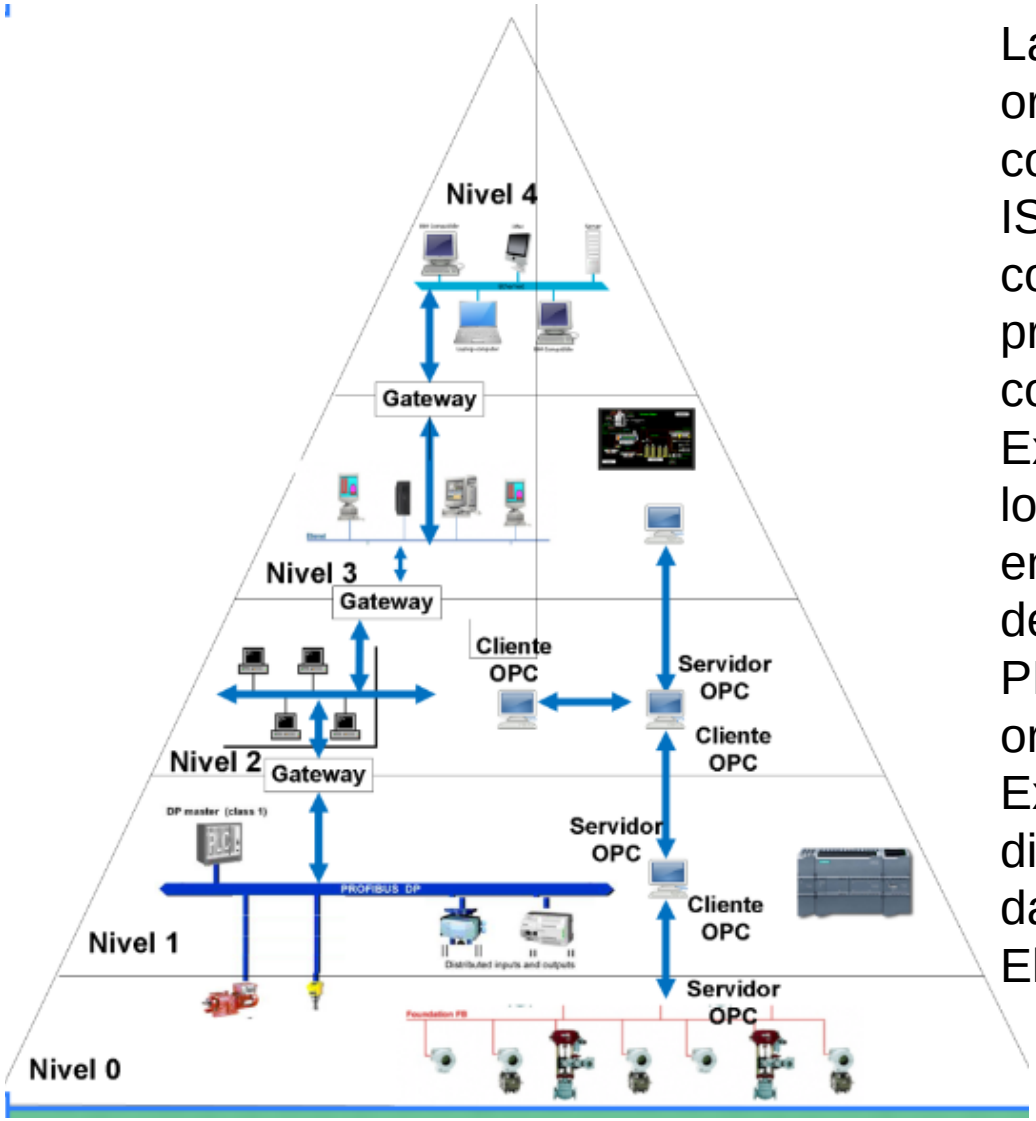
La interfaz de Acceso a Datos de OPC (clásica)

Rogelio Mazaeda
Eusebio de la Fuente

Objetivo

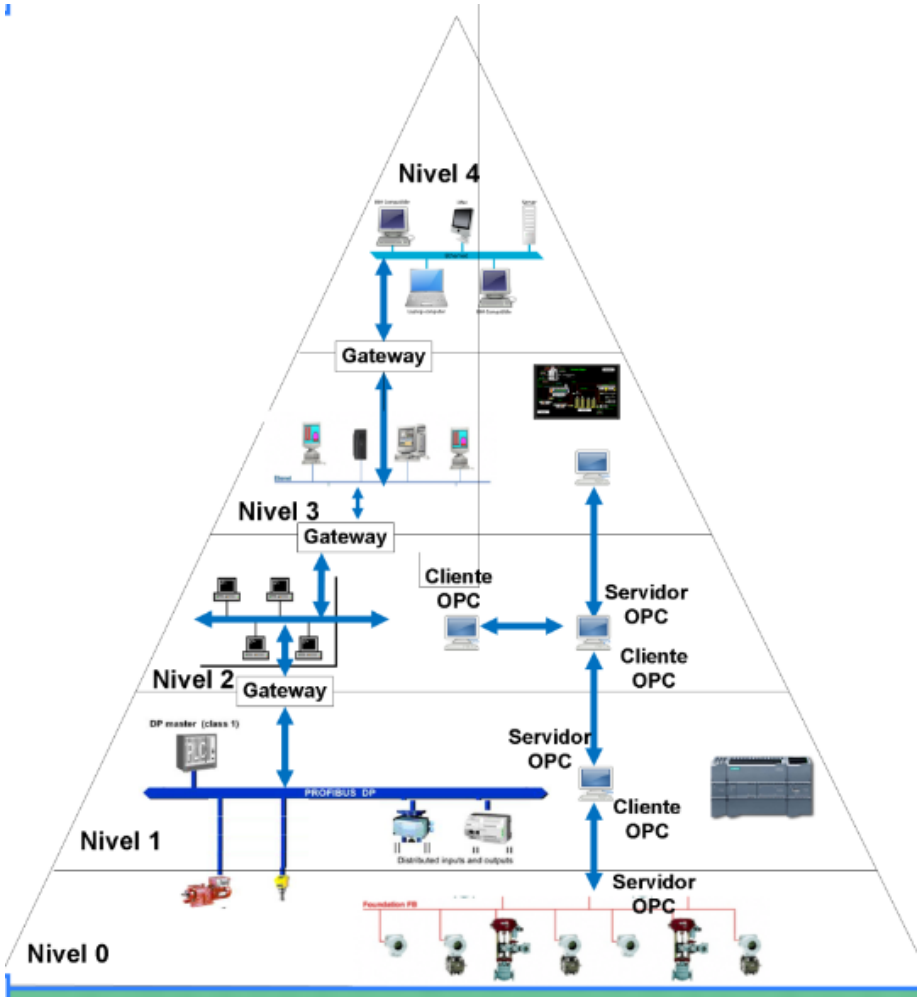
Dotar a las aplicaciones de un mecanismo estándar (OPC) para acceder a datos de fábrica.

Introducción



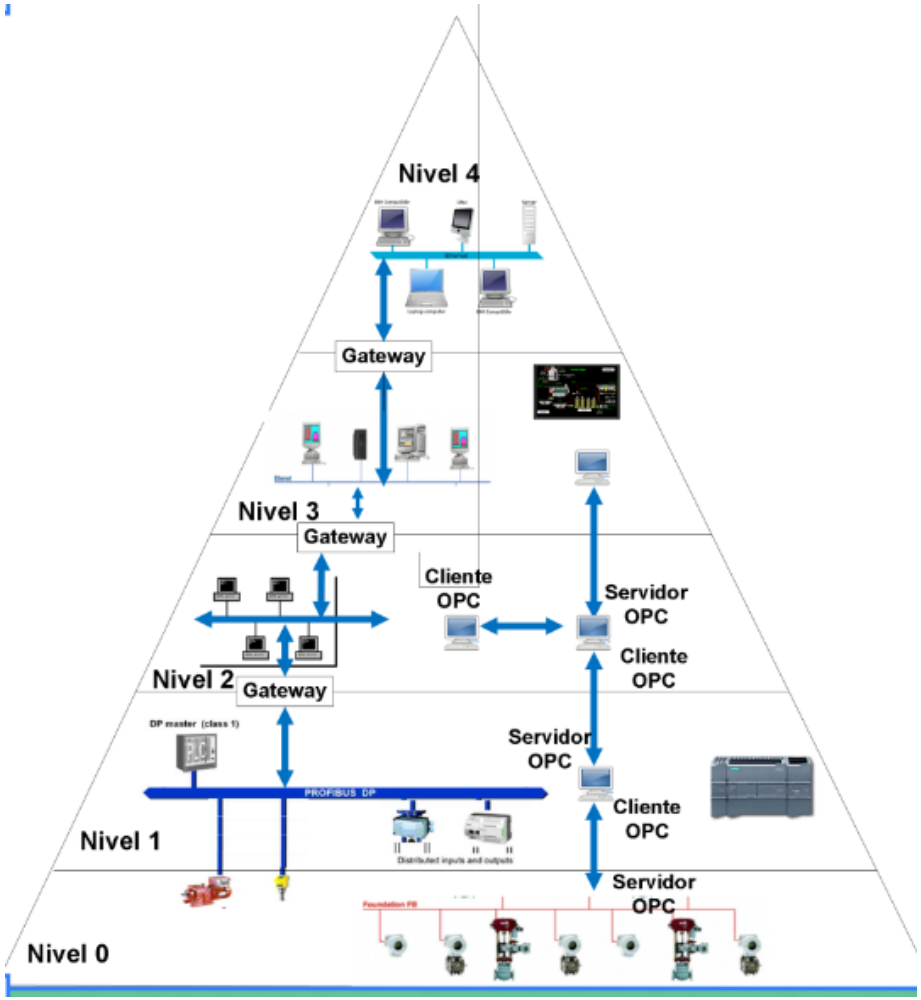
Las fábricas contemporáneas se organizan en estructuras jerárquicas como la refrendada por el estándar ISA-95. Nivel de gestión (ERP), de control a medio plazo de la producción (MES), nivel supervisor, control avanzado, de campo, etc.
Existen aplicaciones informáticas a los diferentes niveles y desplegados en diferentes dispositivo de computo: desde aplicaciones de control en PLC, a control avanzado y en ordenadores convencionales.
Existe la necesidad de que las diferentes herramientas intercambien datos de manera estándar.
El interfaz OPC cumple esta función.

Introducción



OPC: Es un estándar de interoperabilidad para el intercambio confiable y seguro de datos en la industrias Es útil para crear aplicaciones distribuidas en diferentes medios de cómputo conectadas mediante redes informáticas. Las siglas actualmente significan *Open Platform Communications* destacando la actual énfasis en permitir la **interoperabilidad** en plataformas diferentes.

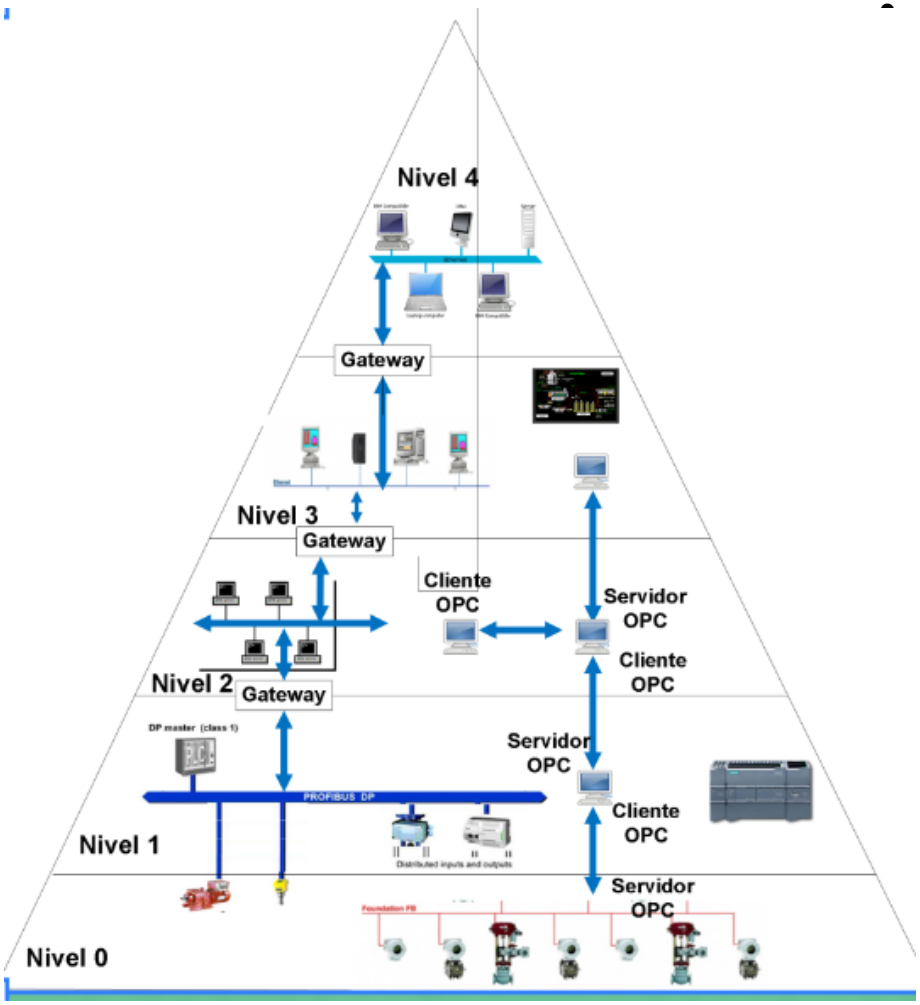
Introducción



Necesidad de OPC:

Brinda procedimientos y herramientas estándares para acceder a los datos desde diferentes fuentes procedentes de distintos fabricantes, sustituyendo la situación previa en la que una proliferación de protocolos y tecnologías propietarias incompatibles hacía muy difícil la necesaria integración.

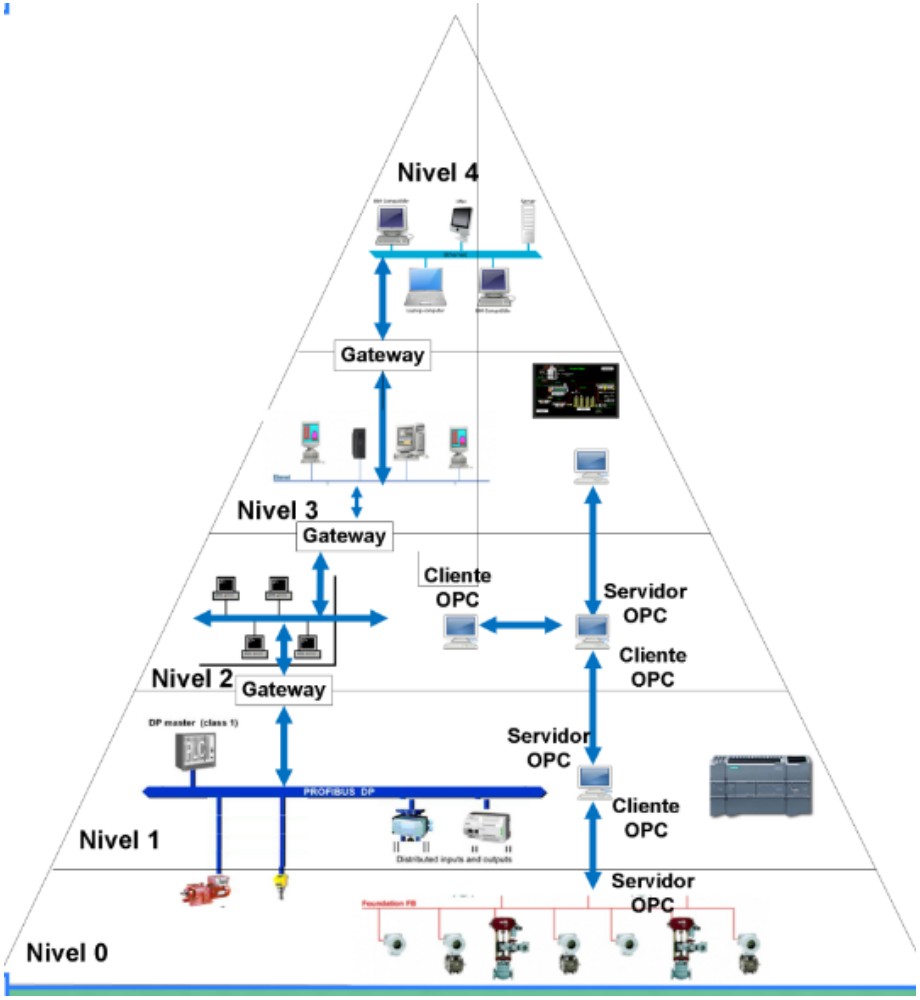
Introducción

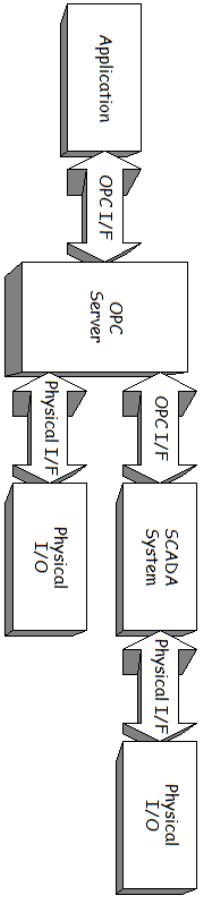


Ventajas de OPC:

- Facilita la labor del integrador de soluciones de automatización. Ej: Se pueden desplegar soluciones comunes a problemas similares. Las soluciones pueden ser probadas en simuladores que implementen el estándar OPC.
  Las industrias a automatizar se benefician porque pueden elegir las mejores soluciones (de diferentes proveedores) para cada caso sin estar obligadas a adquirir de un solo suministrador por razones de compatibilidad.
  Todo lo anterior tiende al abaratamiento de la tecnología de automatización: los proveedores equipos de automatización, a pesar de lo que inicialmente pudiera pensarse, también se beneficiarían de la existencia de una industria más dinámica.

Introducción





Introducción

Desarrollo histórico

1990-1998 (OPC: OLE for Process Control)

OPC clásico v1:
Para plataformas Microsoft
Basado en tecnología DCOM.
OPC DA (Data Access)
Surge OPC Foundation para
Cubrir necesidad de certidicación
y garantizar interoperabilidad

1998-2003 (OPC: OLE for Process Control)

OPC clásico v1:
Se añaden las interfases
OPC A&E (alarmas y eventos)
Y OPC para datos históricos

2003-2009 (especificación OPC- UA)

OPC Unified Arachitecture:
Independiente de Microsoft
Transporte binario o texto
Semántica compleja
Cíberseguridad más elaborada

2010- (consolidación OPC- UA)

OPC UA (IEC 65 541):
Aparece el primer dispositivo
OPC UA empotrado en PLC que
Cumple estánda IEC 61131
2013-Se laza OPC-UA para ISA95

Conviven el OPC clásico (con una gran base instalada industrial) con el OPC-UA, mucho más poderoso pero más complejo. Una cierta inercia para hacer la transición aunque en los últimos años el proceso se acelera. Existen métodos para convertir OPC clásico en UA con unas mínimas funcionalidades

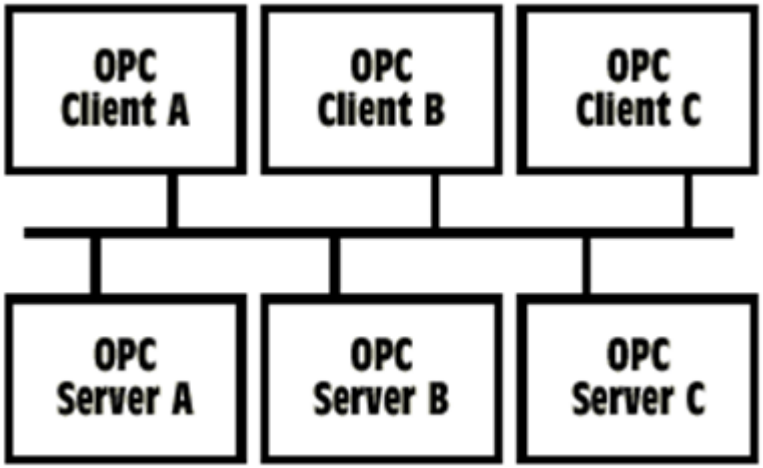
Introducción

Principales estándares OPC

Denominación	Funcionalidad
OPC DA*	Acceso a datos actuales codificados en binario
OPC AE	Tratamiento de alarmas y Eventos
OPC HDA	Acceso a datos históricos (series temporales)
OPC BATCH	Tiene en cuenta peculiaridades de procesos por lotes
OPC Security	Diferentes niveles y políticas de cíber-seguridad
OPC DX	Intercambio directo de datos entre servidores
OPC XML-DA	Acceso a datos formato texto XML (a diferencia del original formato binario)
OPC UA*	Arquitectura Unificado: intención de integrar y superar los estándares previos

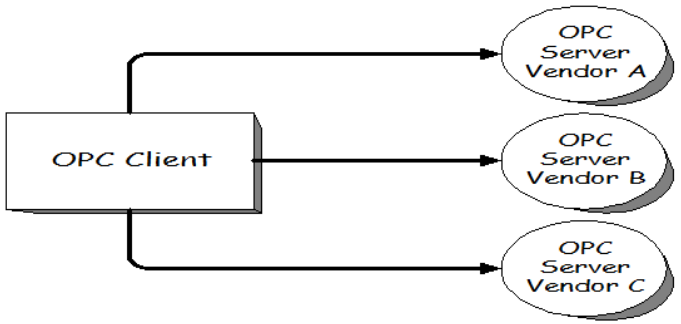
*No todos han tendido igual repercusión. OPC DA (clásico) y OPC UA ha sido las más importantes.

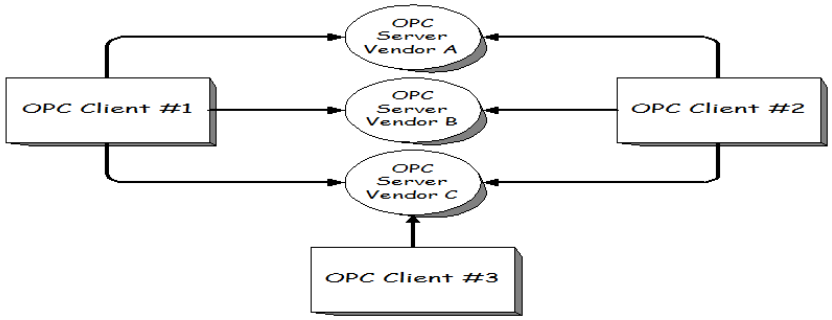
OPC. Modelo Cliente-Servidor



Varias aplicaciones clientes se conectan a uno o varios servidores OPC, que pueden estar en el mismo u otro dispositivo de cómputo accesible mediante red informática, para obtener los datos que estos últimos brindan.

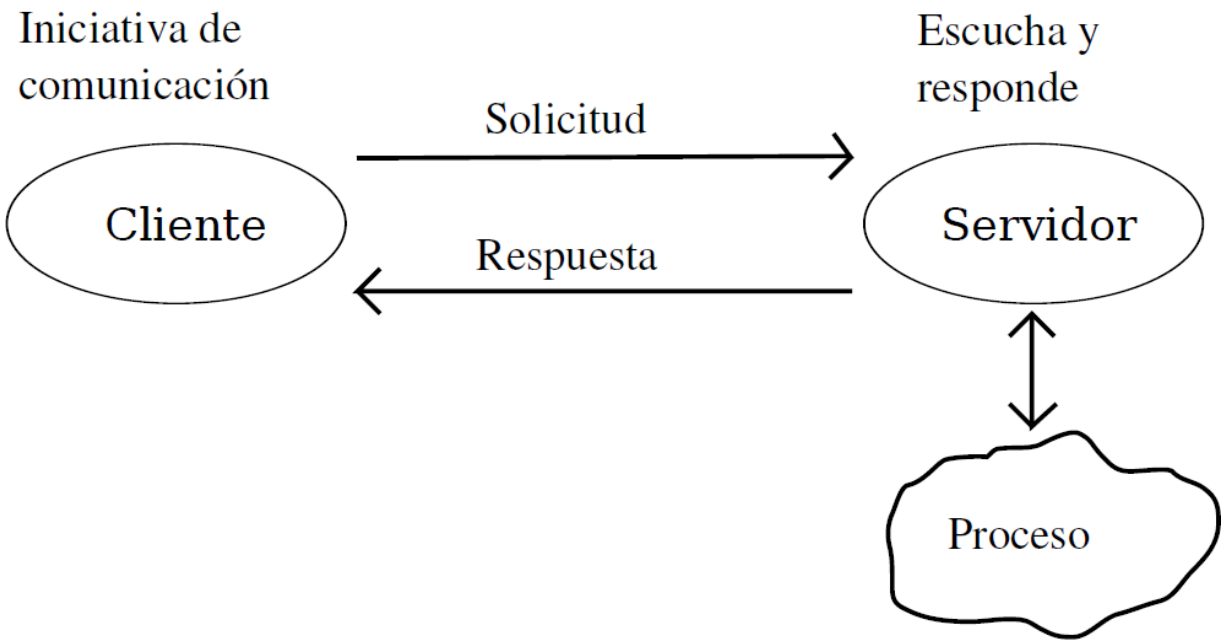
OPC. Modelo Cliente-Servidor





En el caso más elemental el servidor obtiene los datos desde los dispositivos físicos (ej: a través de interface propietaria) o pueden ser datos generados por el propio servidor (un cálculo a partir de medidas físicas) o el servidor a su vez puede ser un cliente de otro servidor OPC, creado jerarquía complejas, que refleja por ejemplo la estructura de la pirámide de ISA-95.

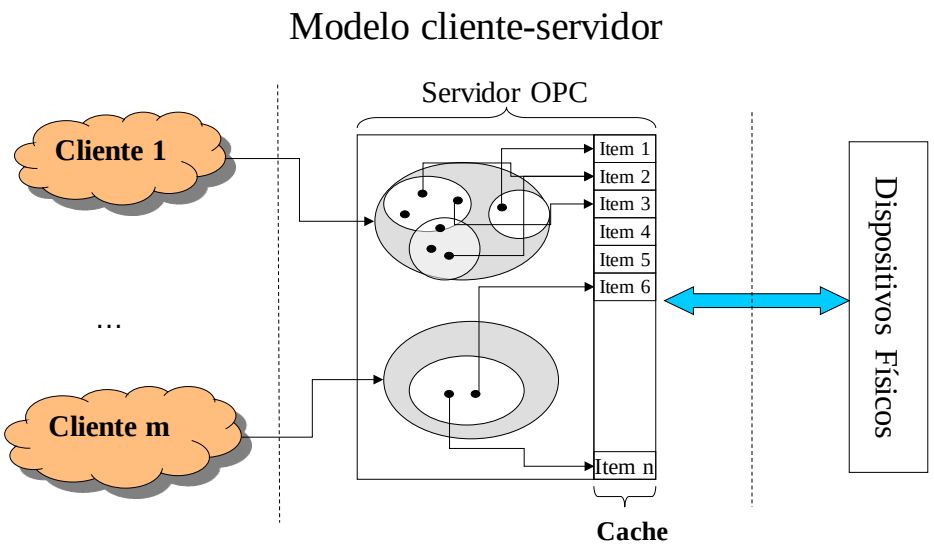
OPC. Modelo Cliente-Servidor



Modelo cliente-servidor, tiene determinadas características generales (compara por ejemplo, con el conocido modelos servidores web/navegadores internet (clientes)):

- El **servidor** expone el acceso un conjunto de datos y se pone de los requerimientos del(los) cliente(s) (**pasivo**).
- El **cliente** tiene un papel **activo**, establece su “interés” sobre un conjunto de datos y toma la iniciativa para leer o escribir sobre ellos, enviando las peticione necesarias al (los) servidor(es)

OPC. Modelo Cliente-Servidor



El estándar OPC (tanto el clásico, como OPC UA) cumple estos requerimientos del modelo cliente/servidor. Algunas particularidades:

- Los datos expuestos por el **servidor** (numéricos, cadenas de caracteres, arrays, etc) están **organizados de forma típicamente** jerárquica (**espacio de nombres: *namespace***) que en muchos casos puede ser interrogado, sobre la marcha, por el cliente.
- El **cliente** puede exhibir **diferentes requerimientos** sobre la frecuencia y la forma de acceder a esos datos, organizándolos en grupos con atributos diferentes.
- Aunque es cierto que el cliente tiene la iniciativa, para leer y escribir (de manera síncrona o asíncrona) sobre las fuentes de datos individuales, también puede establecer su interés en ser **notificado** (por el servidor) cuando existan cambios de suficiente consideración en algunos datos de interés.

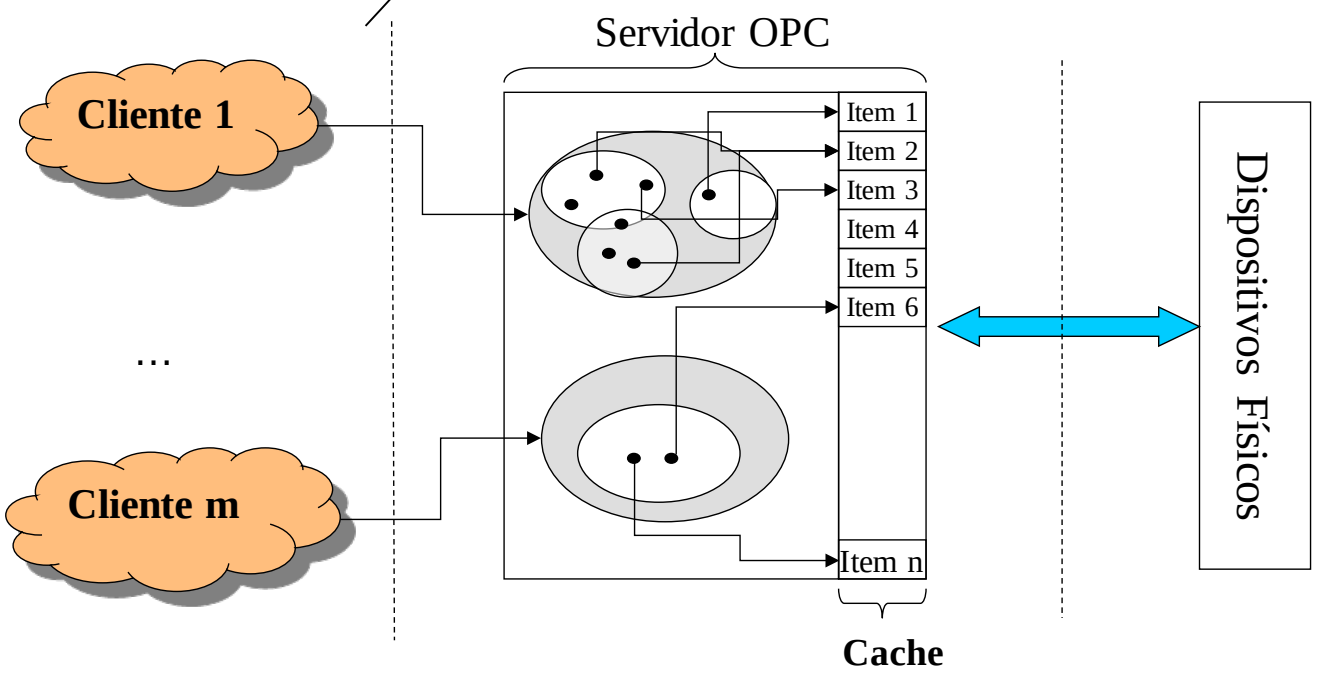
OPC. Mecanismo de interoperabilidad entre aplicaciones.

Frontera puede indicar:

1) Diferente hardware digital unidos mediante red informática.
2) Diferentes procesos en un mismo ordenador (espacio de direcciones diferentes).
3) Servidor y cliente comparten el mismo espacio direcciones (dll).

OPC no es un estándar de comunicación informática. Tanto el cliente y el servidor pueden estar en el mismo medio de cómputo (típicamente en procesos separados). Si estuvieran en ordenadores separados, utilizaría estándares de comunicación para establecer la interacción: OPC (sólo en este caso) pudiera considerarse como una aplicación (en la correspondiente capa) del modelo OSI de comunicación subyacente.

Modelo cliente-servidor



OPC clásico. Fundamentos

OPC debe muchas de sus características a la forma en que se desarrollo originalmente, es por ello y por el hecho de su todavía amplia base desplegada industrialmente, que partiremos de la descripción del **OPC clásico**, para finalmente destacar las diferencias con el nuevo **OPC UA**.

Conceptos básicos para entender **OPC clásico**:

1. Programación Orientada a Objetos (POO) y programación basada en componentes.
2. Administrador de Componentes (Distribuidos) (broker) de Microsoft (D)COM.
3. Interfaz y Objetos (D)COM.
4. ¿Qué es el identificador único universal CLSID?
5. Las llamadas a procedimientos remotos (RPC). Los eventos.
6. Interfaces comunes y hechas a la medida (*custom*).
7. El OPC clásico como una interfaz (D)COM hecha a la medida.
8. El interfaz de Automation de OPC. Descripción detallada.
9. Creación de servidores y clientes OPC clásico.
10. Creación de servidores OPC simulados mediante EcosimPro y clientes de Matlab.

OPC clásico. Fundamentos

Programación Orientada a Objetos (POO) y programación basada en componentes.

La POO intenta cumplir el viejo sueño de la industria de software de crear aplicaciones informáticas complejas combinado diferentes objetos (piezas de código que combina datos + procedientos para actuar sobre los mismos) y que resultan instancias de clases (patrones) que definan ciertas características comunes.
De esta forma se podría crear un **aplicación de control**, por ejemplo, combinando de manera apropiada **instancias de clases** tales como **controlador**, **válvula**, **bomba**, **depósitos**, etc.
Hay lenguajes de programación que soportan mejor este paradigma de programación. Ej: Java, C++, entre otros.

OPC clásico. Fundamentos

Programación Orientada a Objetos (POO) y programación basada en componentes.

Entre las características más importantes de la POO están:

- **Encapsulamiento**: los usuarios de un objeto acceden mediante interfaz estable, bien definida y no tiene acceso a la forma interna en la que fue programada la clase.
- **Herencia**: Una clase de objetos puede heredar el comportamiento y los datos de una o varias clases padre. Se puede crear árboles de herencia muy profundos. Ej: las clases **bomba** y **válvula** podrían derivarse de una clase común **actuador**, y esta última a su vez de otra **dispositivo_campo**. Esto permite reutilizar el código y minimizar la oportunidad de ocurrencia de errores.
- **Polimorfismo**: Un mismo código puede tratar de manera idéntica a clases diferentes, apelando al elemento en común que las mismas posean. Ejemplo: se puede pedir el **identificador** a una **bomba** y **controlador**, diferentes objetos pero que son tratados de forma similar porque ambos son **dispositivos_campo**.
- **Agregación**: Un objeto complejo puede estar conformado por instancias de objetos más simples, en estructuras anidadas tan profundas como se quiera.

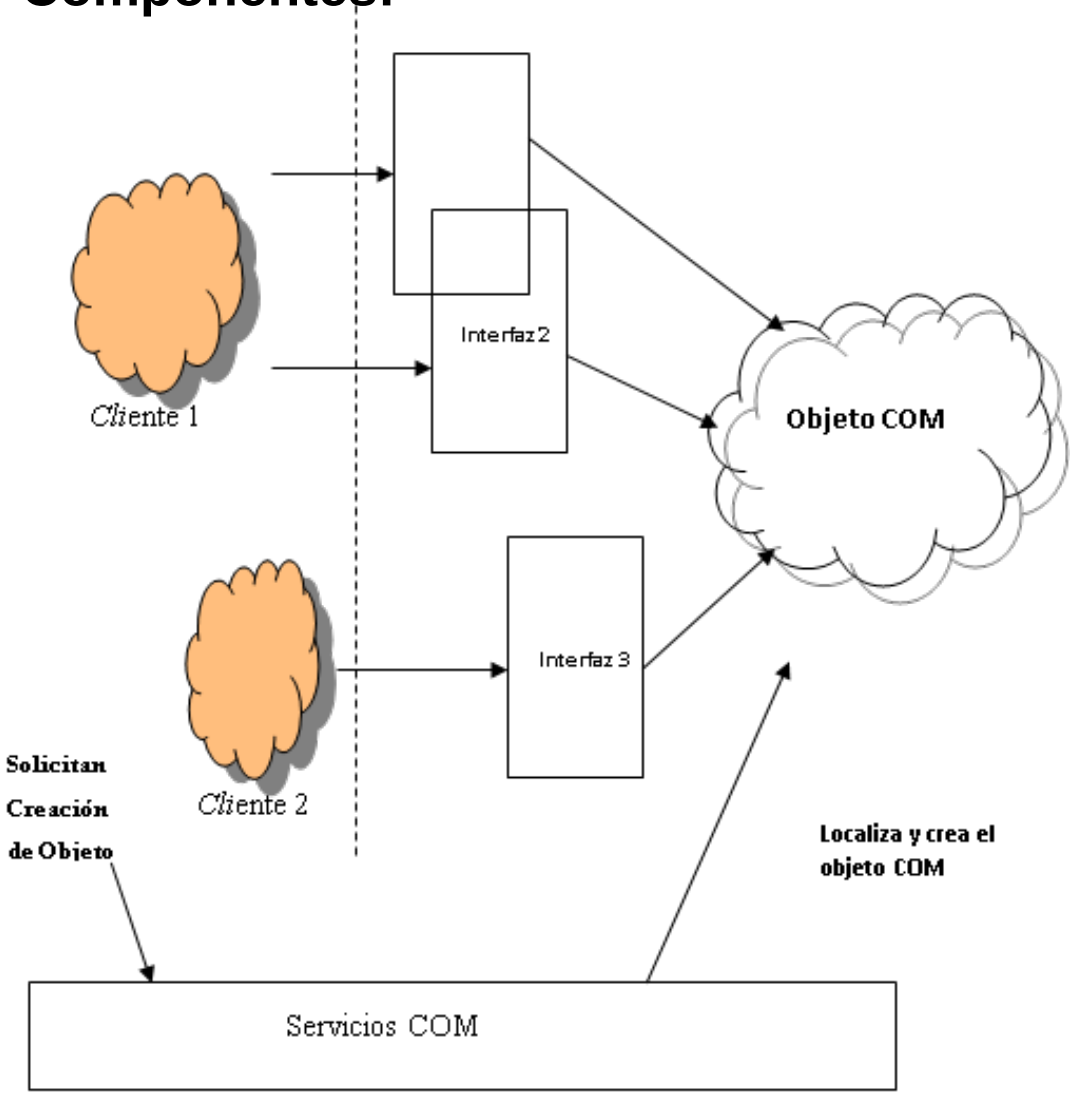
OPC clásico. Fundamentos

Componentes:

Pero la POO es útil en el momento de la creación del programa. ¿Cómo extender las ideas de la POO a nivel de todos el Sistema Operativo (SO) o incluso de diferentes ordenadores conectados en red (aplicaciones distribuidas)?
Se necesitaría en este caso de un **estándar binario** (las aplicaciones, probablemente hechas en lenguaje diferentes, ya está en código de máquina) y de un mecanismo a nivel del SO que establezca las normas de intercambio de información y que permita a los diferentes componentes binarios interactuar a pesar de estar, quizá en procesos (espacios de direcciones diferentes) e incluso diferentes ordenadores. El **(D)COM**: ***(Distributed) Component Object Mode***l es ese tipo de intermediario (broker de objetos) de Mircrosoft. Otros intermediarios conocidos sería **CORBA** (para sistemas UNIX).
Actualmente Microsoft no se seguirá actualizando **DCOM** (se ha apostado por **.NET** en su lugar) aunque las aplicaciones existentes pueden confiar en su existencia para sucesivas versiones del SO.

OPC clásico. Fundamentos

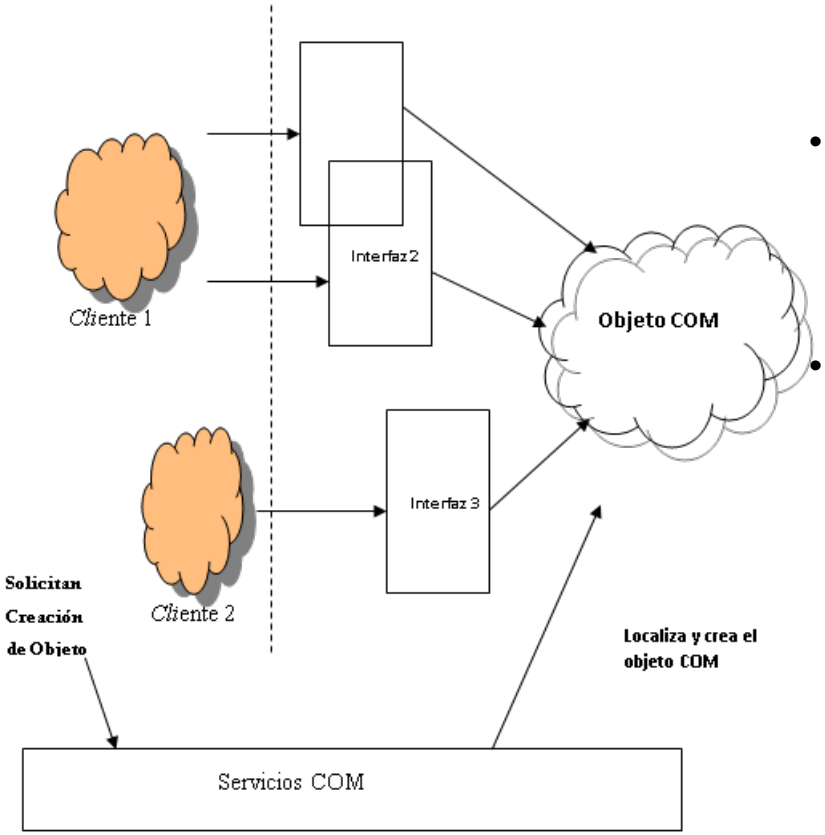
Componentes:



- Para poder participar en estas aplicaciones distribuidas multi-componente, el objeto (D)COM o componente debe “exportar” diferentes **interfaces**.
- Cada interfaz contiene la descripción de determinado **conjunto de funcionalidades** (relacionadas semáticamente entre sí) que el objeto ofrece.
- EL *broker* **(D)COM** es el encargado de **localizar** el **objeto y la interfaz** (que puede estar en el mismo ordenador u otro accesible mediante res) solicitada por el cliente, **ejecutar el servidor COM** y ocuparse de **garantizar la “interacción”** efectiva entre ambos.

OPC clásico. Fundamentos

Componentes:

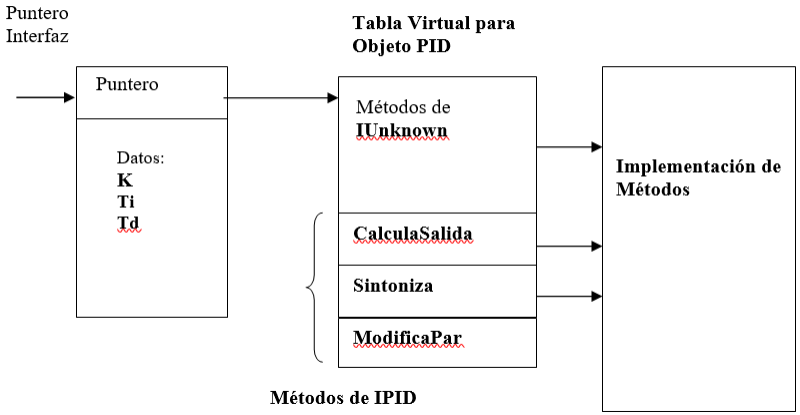


¿Cómo un objeto cliente accede a la funcionalidad que le interesa del objeto servidor (D)COM?

- Una vez que los servicios (D)COM han establecido la conexión con el objeto COM a través de la interfaz de interés, la comunicación del cliente y el servidor se establece, en la mayoría de las casos, como una simple **llamada a función**.
- Típicamente la llamada es **síncrona**: la función del cliente simplemente ejecuta la función en el servidor y su ejecución es detenida hasta que el servidor devuelve la respuesta esperada: similar a como se ejecuta una función en un lenguaje concreto como el C o C++.
- Responsabilidad del DCOM:
  - **Servidor es una dll en el mismo ordenador**: función DCOM mínima, es realmente una llamada síncrona convencional.
  - **Servidor en mismo ordenador diferente proces**o (otro espacio de direcciones). DCOM debe utilizar mecanismo de comunicación entre procesos, en este caso **RPC** (*Remote Procedure Call*). Elementos como el paso de parámetros de la función y el resultado de la misma debe ser manejado apropiadamente por DCOM.
  - **Servidor en ordenador diferente**: A todas las dificultades anteriores, se añade el enviar los datos a través de la red.

OPC clásico. Fundamentos

Componentes:



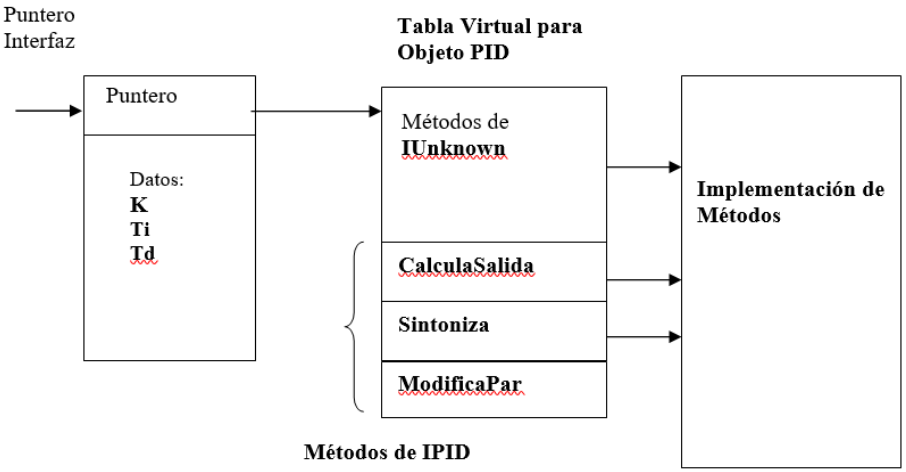
Ejemplos de Interfaces estándar (D)COM:

- **IUnknown:** Todo objeto COM tiene que soportar esta interfaz. A través de sus métodos se acceden al resto de las interfaces tanto (estándar y *custom*)
- **IClassFactory:** Soportada por objetos COM directamente instanciables por una aplicación cliente.
- **Istorage, IPersistStorage, IStreamStorage:** Implementadas por clases capaces de almacenar su estado de forma permanente en almacenamiento permanente (discos, etc).

Además de las interfaces estándar, existe la interfaces a la medida (custom). En el ejemplo, se muestra una supuesta Interfaz *custom* que brinda las funcionalidades de un controlador PID. Nótese que todo objeto COM debe soportar Iunknown. En este caso además soporta una supuesta interfaz Ipid con tres métodos.

OPC clásico. Fundamentos

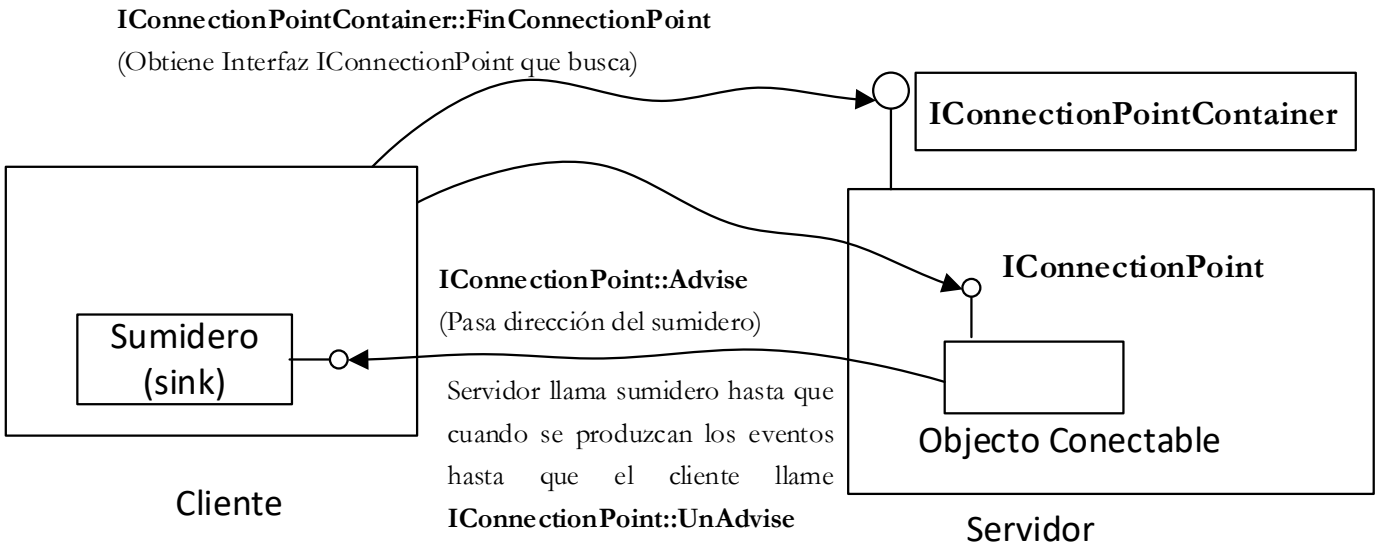
Componentes:



Un cliente que acceda (mediante servicios COM) a el puntero de la interfaz podría invocar los **métodos** que exporta la interfaz ***custom***: CalculaSalida, Sintoniza y ModificaPar. Nótese que como corresponde al **modelo cliente-servidor**, el cliente es el que toma la iniciativa y hace las llamadas a los métodos correspondientes como si fueran simples llamadas a funciones. (D)COM se encarga de “esconder” la complejidad subyacente.
El modo de actuar anterior es muy cómodo para aquella aplicaciones clientes realizadas en C/C++, que son lenguajes dotados de la capacidad de trabajar fácilmente con **punteros**.

OPC clásico. Fundamentos

Componentes:



Existen otros tipos de interfaces (D)COM de interés:

- La interfaz **IConnectionPoint** es atípica en el sentido de que permite configurar en el Servidor el mecanismo que permite que este informe al cliente de la ocurrencia de determinado evento (objeto conectable). De alguna manera se modifica la actuación pasiva del servidor, permitiéndole que tome la iniciativa e inicie la comunicación con el cliente de la dirección de una función del mismo previamente configurada.
- Existen interfaces que representan contenedores (*container*) de otras interfaces y que tiene métodos para recorrer los elementos contenidos (En el ejemplo: **IConnectionPointContainer** es una interfaz que devuelve la colección de todos los objetos conectables de determinado servidor COM.

OPC *Automation*

La ***OPC Foundation*** ha especificado un conjunto de interfaces **COM A la medida o *Custom*** para facilitar un mecanismo estándar de intercambio de datos entre aplicaciones clientes y fuentes de datos para las aplicaciones de Automatización Industrial y control de Procesos.

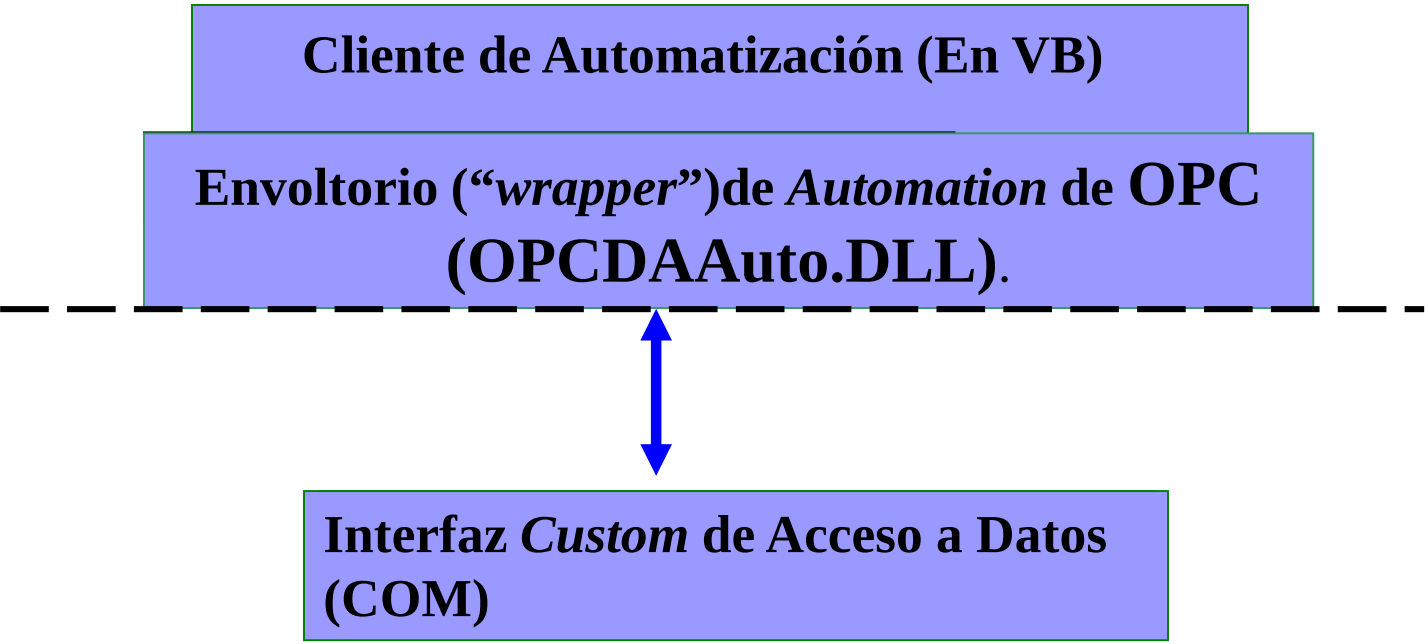
Creación de clientes utilizando Interfaces COM requiere un esfuerzo considerable y conocimientos de lenguajes de bajo nivel como C/C++.

Para aliviar esta situación *OPC Foundation* brinda **Interfaz de *Automation* de Acceso a Datos que permite** programar clientes OPC desde **Visual Basic.** Técnicamente los objetos *Automation* son aquellos que implementan la interfaz COM ***IDispatch.***
En lo que sigue veremos la jerarquía de objetos que propone OPC DA clásico tal y como se perciben desde clientes como VB.

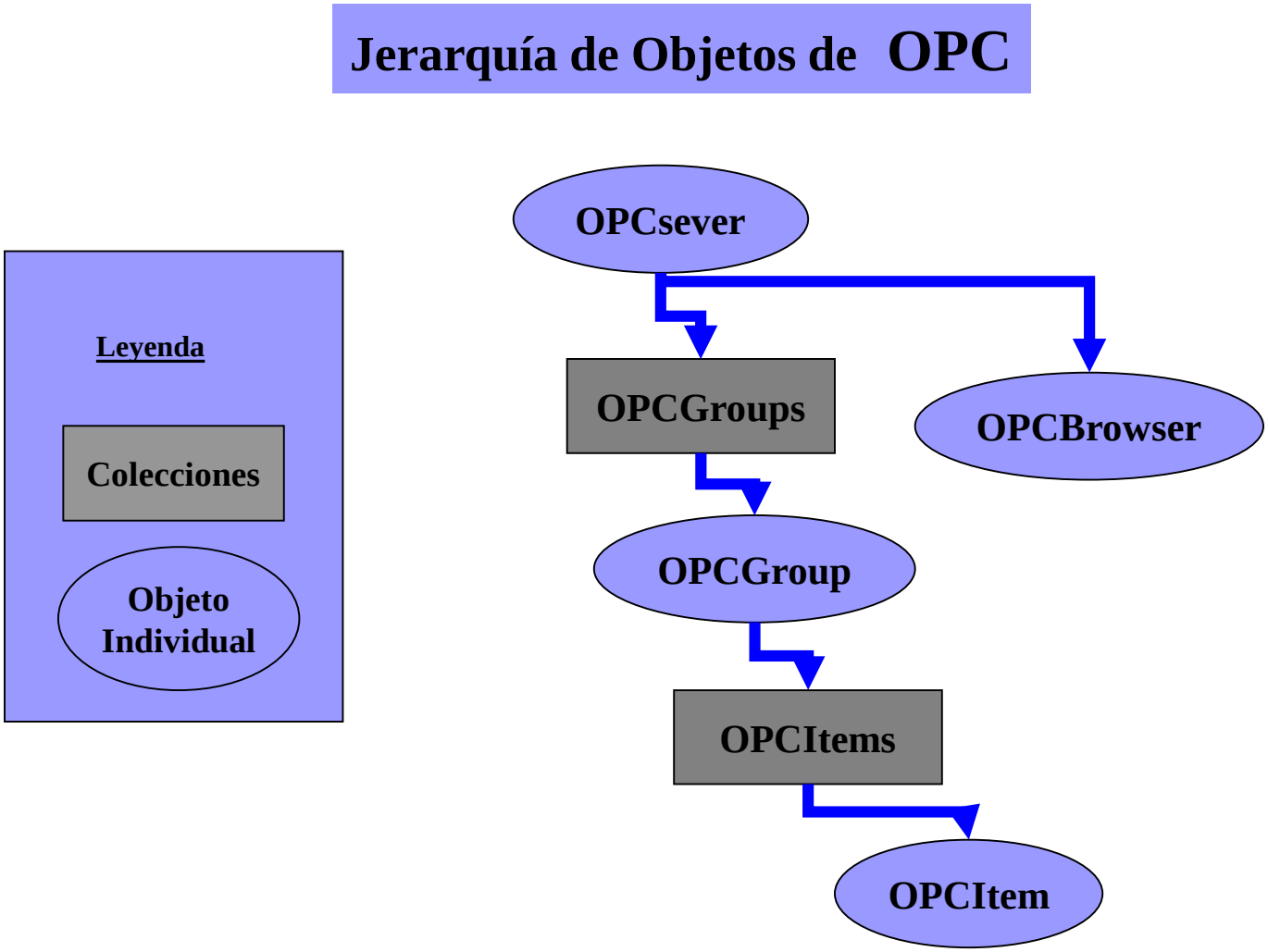
OPC *Automation*

La especificación de *Automation* de OPC se brinda como una DLL. Contiene:

- El código que implementa la Jerarquía OPC.
- La Biblioteca de Tipos (*Type Library*) que permite realizar enlace temprano (en tiempo de compilación) desde VB a los objetos de Jerarquía OPC.

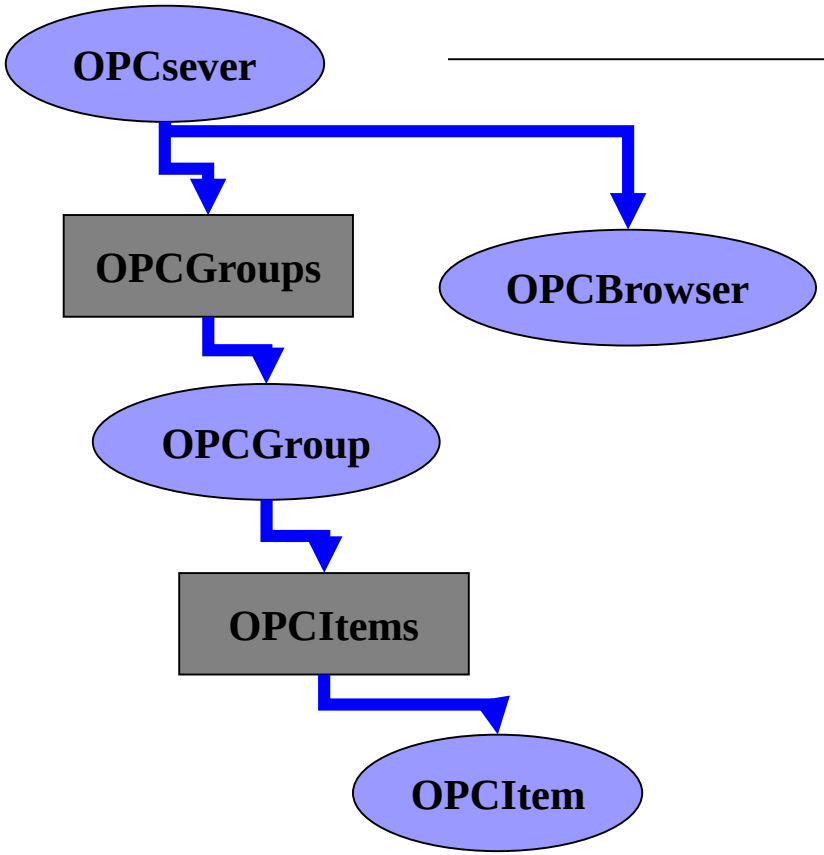


OPC *Automation.* Jerarquía



OPC *Automation.* Jerarquía

Jerarquía de Objetos de OPC

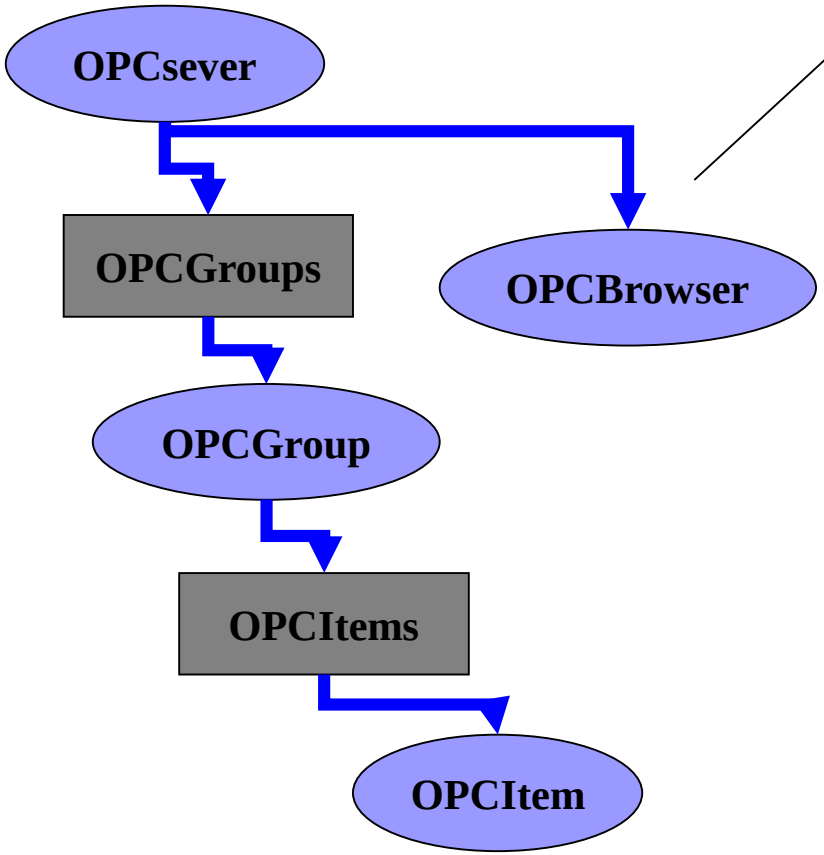


OPCServer:

Objeto Raiz de la Jerarquía OPC.
Representa a cualquier servidor OPC.
Antes de usarlo, se debe **conectar** a un servidor específico.
Es el único que puede ser creado directamente desde el cliente (bajo cuerda tiene la interfaz **IClassFactory**).

OPC *Automation.* Jerarquía

Jerarquía de Objetos de OPC

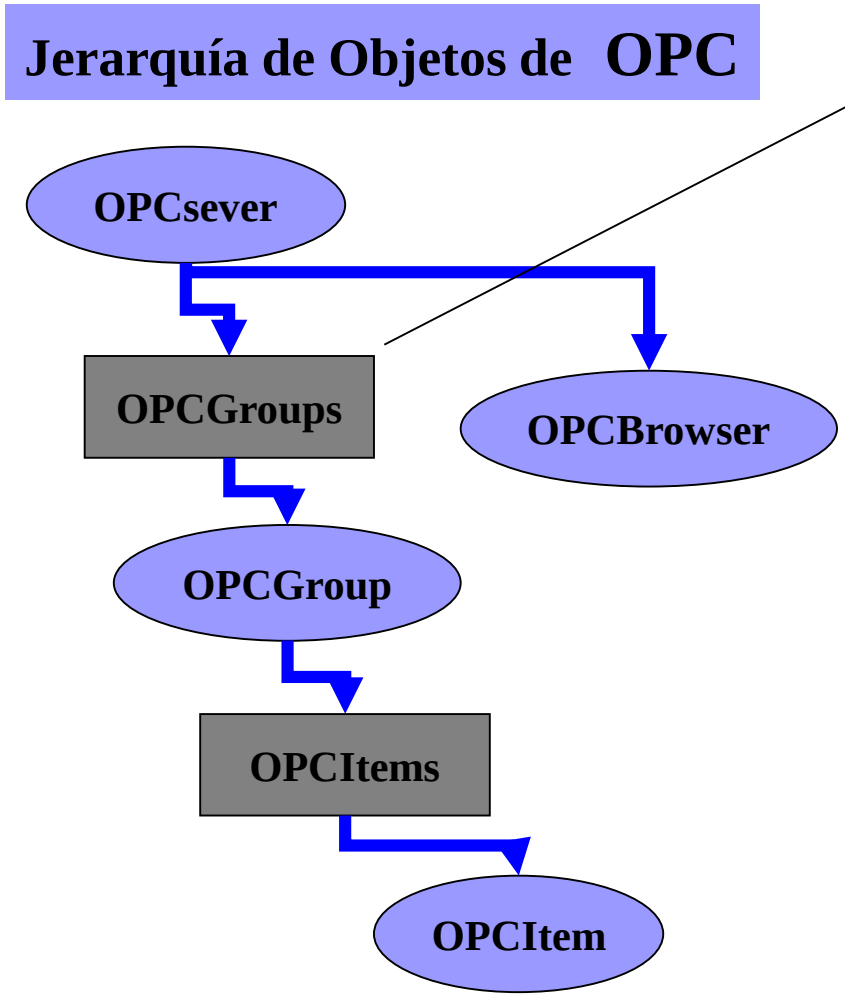


OPCBrowser:

Objeto opcional. Permite recorrer el espacio de nombres que ofrece un servidor OPC específico.

El espacio de nombres es el "inventario" de los nombres (o *tags*) de las posibles fuentes de datos que brinda determinado servidor OPC

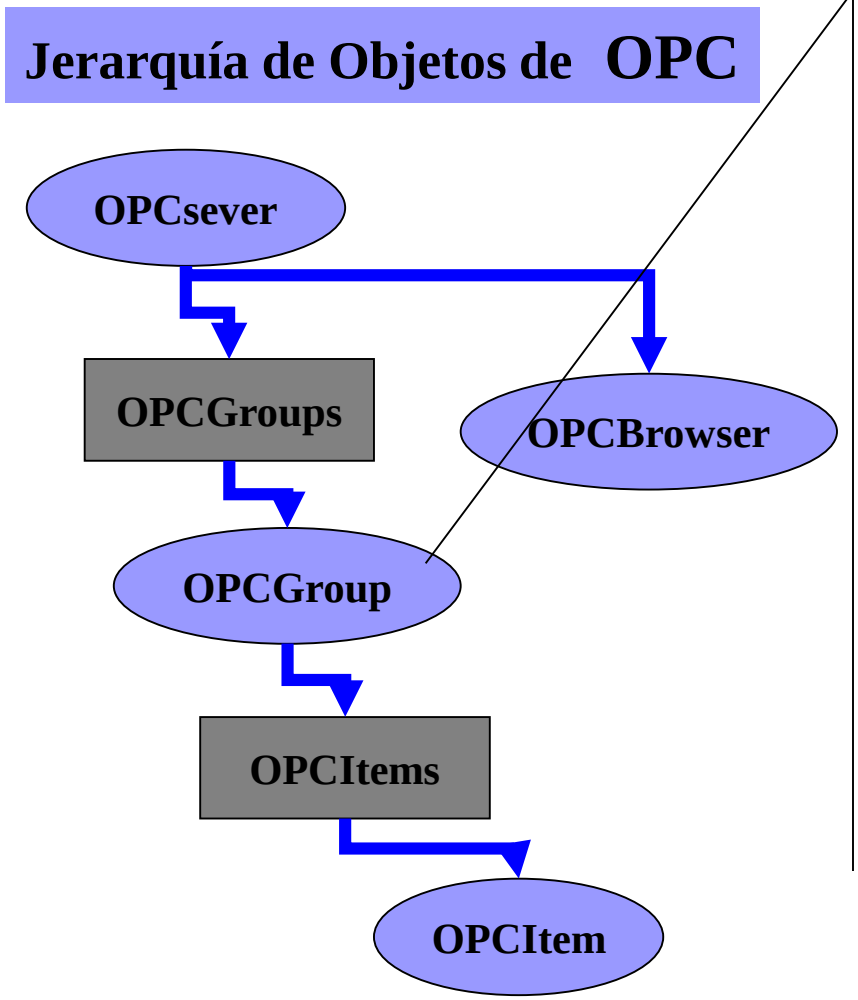
OPC *Automation.* Jerarquía



OPCGroups:

Objeto Colección que agrupa a todos los objetos **OPCGroup** que determinado cliente ha registrado con un servidor OPC específico. Implementa interfaces de tipo colección o de enumeración.

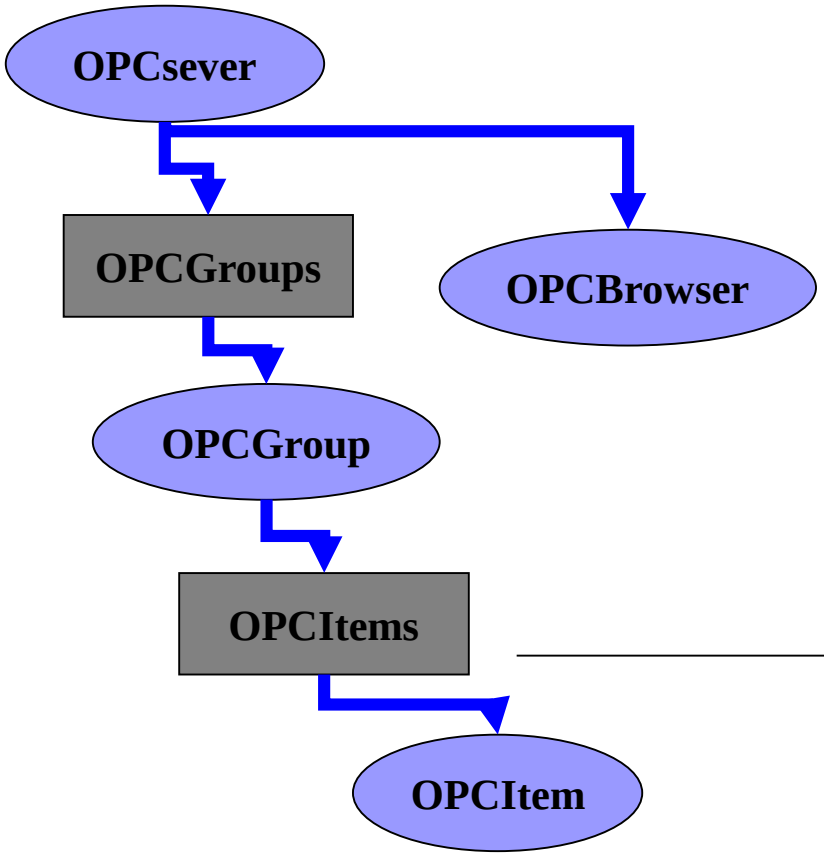
OPC *Automation.* Jerarquía



OPCGroup:
Representa un Grupo de elementos de datos o *items* (uno o más) que la aplicación cliente ha considerado conveniente registrar en un mismo grupo.
Elemento fundamental de la Jerarquía OPC.
Los criterios para crear los grupos son decididos por las necesidades de la aplicación cliente.
Un *item* puede formar parte simultáneamente de varios grupos.

OPC *Automation.* Jerarquía

Jerarquía de Objetos de OPC

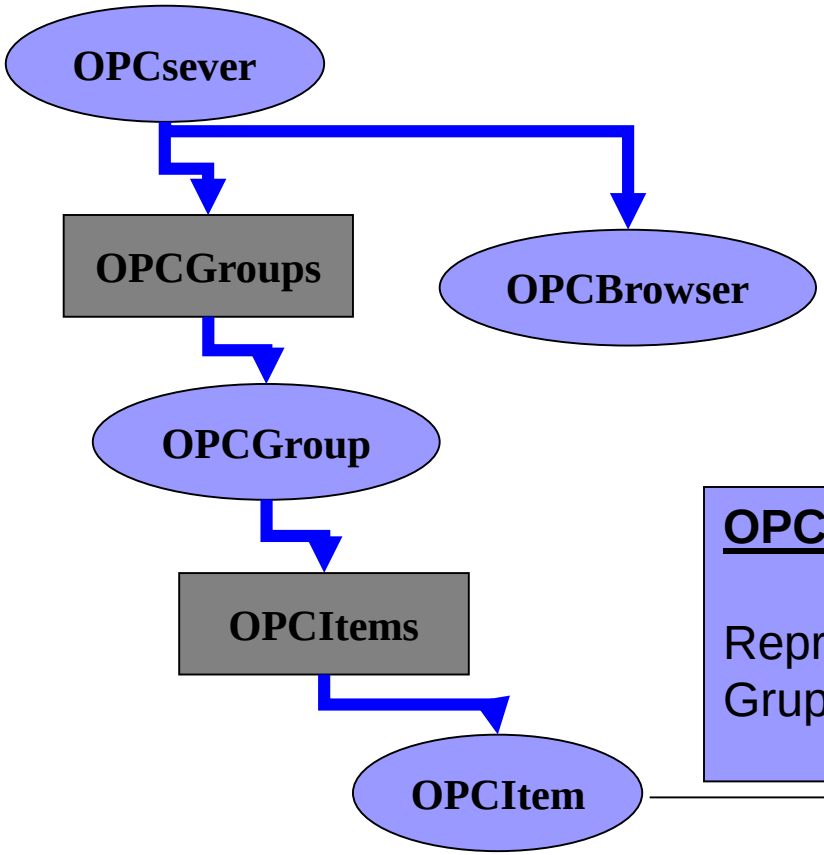


OPCItems:

Un objeto colección que contiene los *items* de determinado Grupo.

OPC *Automation.* Jerarquía

Jerarquía de Objetos de OPC

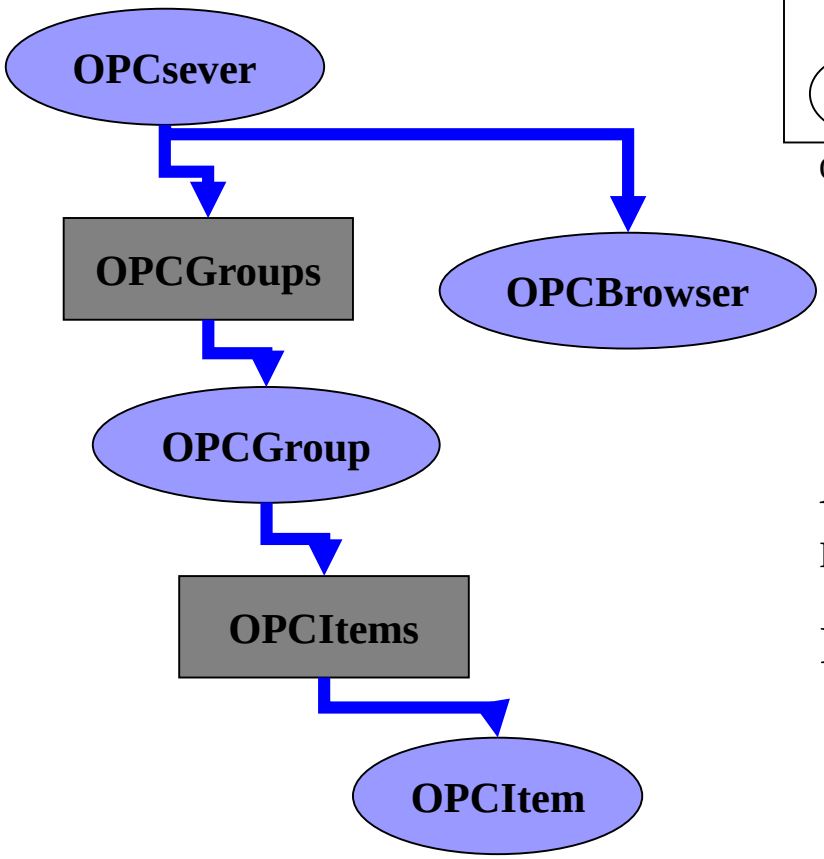


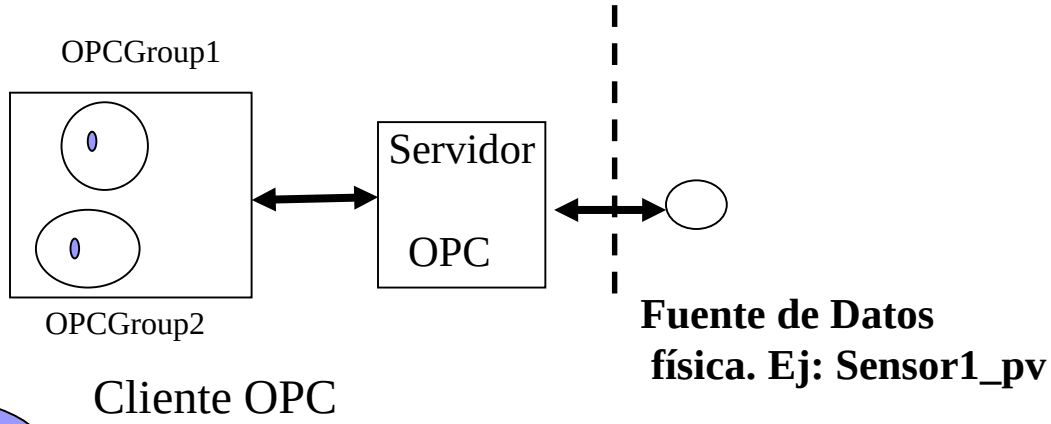
OPCItem:

Representa a un *item* individual **dentro** del Grupo en que haya sido configurado.

OPC *Automation.* Jerarquía

Jerarquía de Objetos de OPC



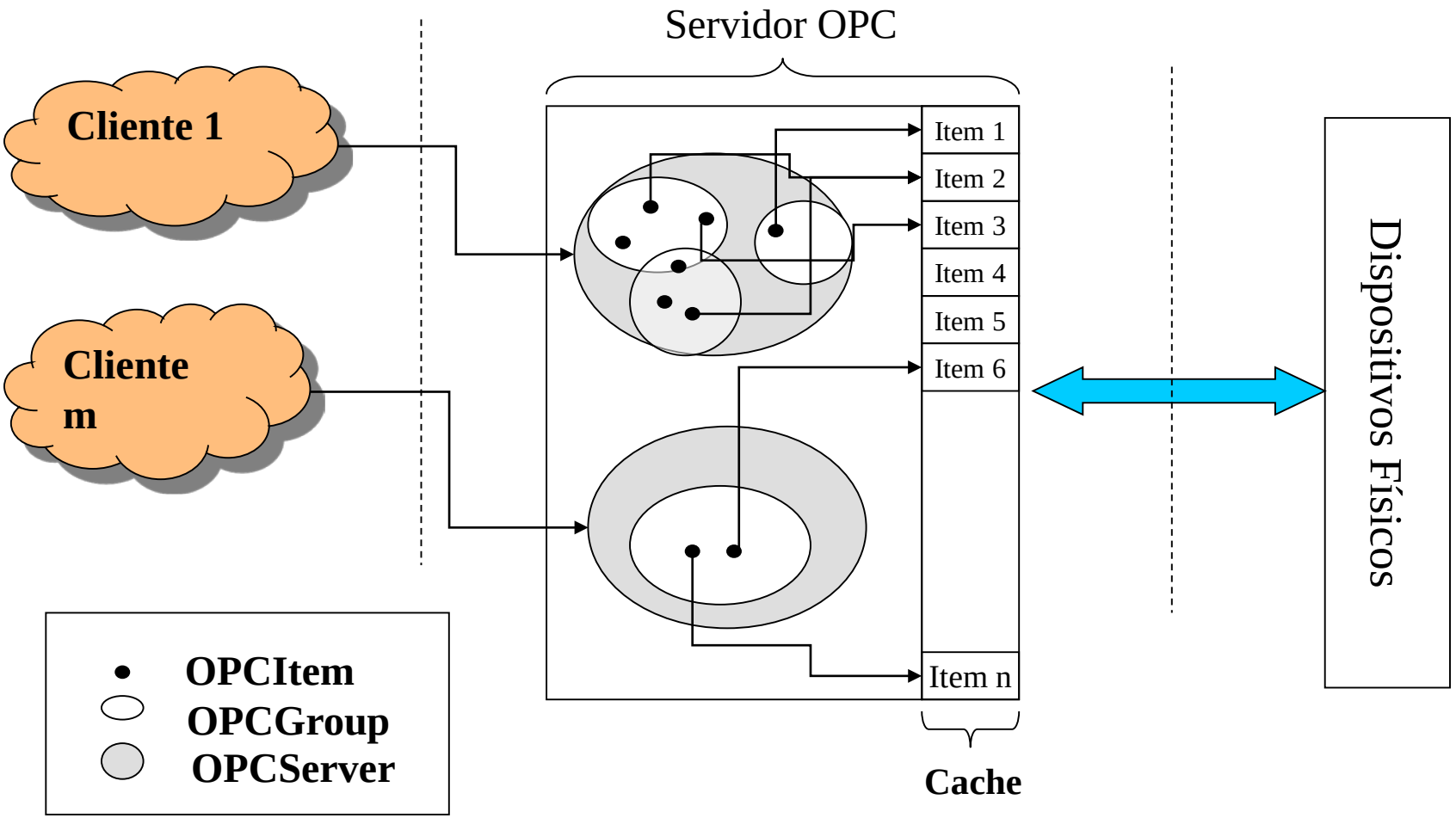


Ambos Grupos contienen una referencia al mismo sensor físico.

Pero objetos OPCItem son distintos

OPC *Automation.* **Esquema General**

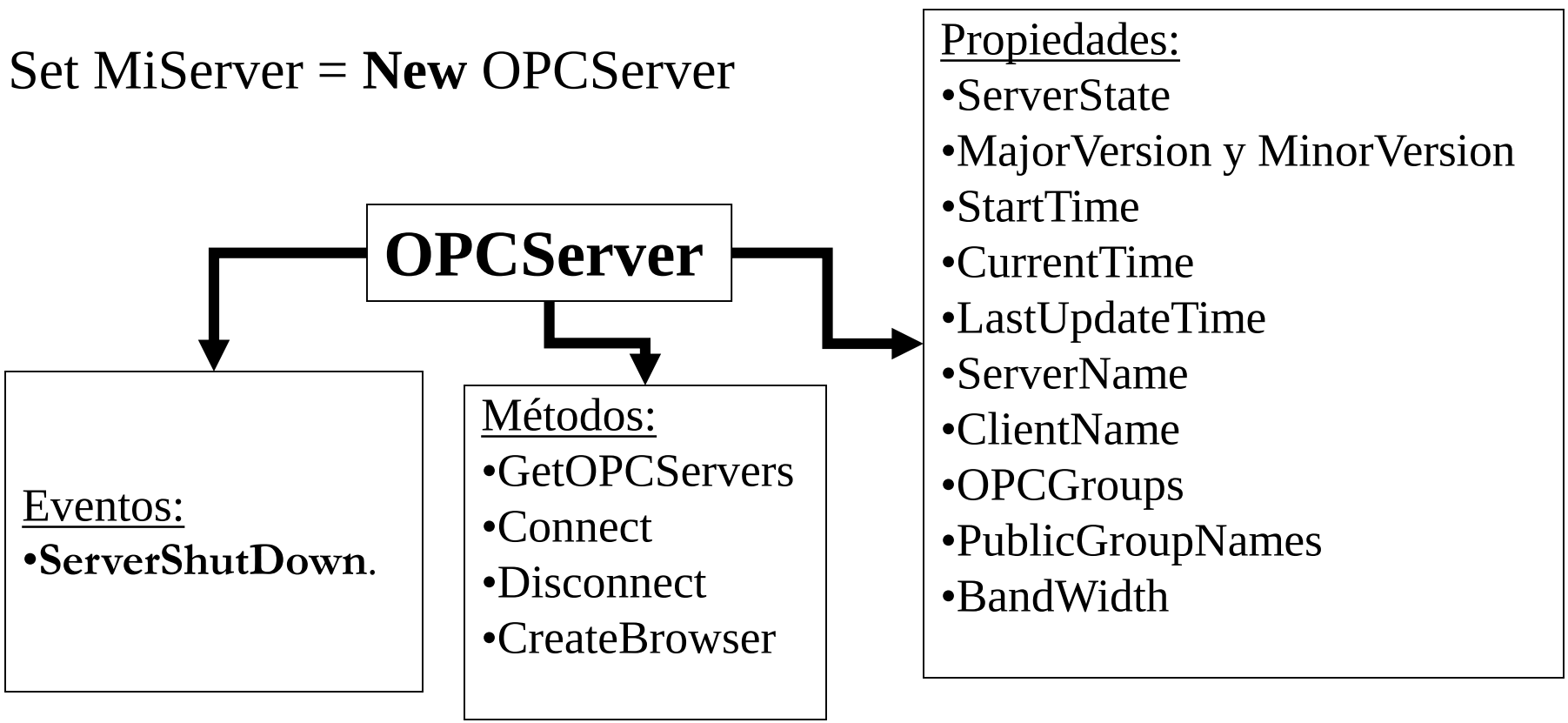
Esquema de Servidor OPC



OPC *Automation.* *OPCSever* **Descripción Detallada**

Objeto **OPCServer**: Único directamente creable desde el cliente.
Ejemplo en VB
Dim Withevents MiServer **as** OPCServer

Set MiServer = **New** OPCServer



OPC *Automation.* *OPCSever* **Descripción Detallada**

OPCServer

Propiedades:
- ServerState
- BandWidth
- StartTime
- CurrentTime
- LastUpdateTime
- ServerName
- ClientName
- PublicGroupNames
- OPCGroups

OPCserver.ServerState Propiedad de sólo lectura. Indica el estado del servidor. Puede tomar una de las siguientes valores constantes (definidas en la *Type Library*):

OPCStatusRunning: Funcionamiento normal
OPCStatusFailed: Error fatal en el servidor
OPCStatusNoConfig: Servidor No configurado.
OPCStatusSuspended: Servidor suspendido momentáneamente.
OPCStatusTest: El servidor desconectado del *hardware*.

OPC *Automation.* *OPCSever* **Descripción Detallada**

OPCServer

Propiedades:

- ServerState
- BandWidth
- StartTime
- CurrentTime
- LastUpdateTime
- ServerName
- ClientName
- PublicGroupNames
- OPCGroups

OPCserver.BandWidth (long): Propiedad de sólo lectura. *OPC* sugiere que el servidor debe utilizar esta propiedad para informar al cliente del ancho de banda actual de la comunicación con el servido. Se da por ciento en relación al ancho de banda total disponible.

OPCserver.StartTime (Date): Tiempo en que fue creado el OPCServer.

OPCserver.CurrentTime (Date): Tiempo actual.

OPCserver.LastUpdateTime (Date): El tiempo de la última actualización de los datos en el servidor.

OPCserver.ServerName (String) lectura: Brinda al cliente el nombre del servidor al cual se ha conectado.

OPCserver.ClientName (Date) Lectura-Escritura: El cliente puede registrar en el servidor el nombre que desee.

OPC *Automation.* *OPCSever* **Descripción Detallada**

OPCServer

Propiedades:
- ServerState
- BandWidth
- StartTime
- CurrentTime
- LastUpdateTime
- ServerName
- ClientName
- PublicGroupNames
- OPCGroups

OPCserver.PublicGroupNames (String(): Matriz de string) lectura: Devuelve los nombres de los grupos públicos que el servidor tenga definidos. Si el servidor no soporta o no tiene definidos Grupos Públicos, la matriz devuelta estará vacía

OPCserver.OPCGroups (OPCGroups) Lectura: El servidor devuelve el objeto colección que contiene todos los grupos (**OPCGroup**) que el cliente haya registrado en el servidor.

OPC *Automation.* *OPCSever* **Descripción Detallada**

OPCServer

Métodos:
- GetOPCServers
- Connect
- Disconnect
- CreateBrowser

OPCserver.GetOPCServer. Devuelve matriz de cadenas de caracteres con el nombre de todos los servidores *OPC* registrados en el sistema. Para ejecutar este método no es necesario que la variable **OPCServer** esté conectado a un servidor concreto.

GetOPCServers(Optional Nodo As Variant) As Variant

Nodo: Es opcional. De darse, representa el identificador de un ordenador en red, donde se desea obtener la lista de servidores registrados. Nombres válidos:

UNC (“Server”)
Nombres DNS (“Server.com”)
Direcciones TCP\IP: “180.23.32.01”

OPC *Automation.* *OPCSever* **Descripción Detallada**

OPCServe
r

Métodos:
- GetOPCServers
- Connect
- Disconnect
- CreateBrowser

OPCserver.Connect. Conecta el objeto **OPCServer** a determinado servidor concreto registrado en el sistema

Connect(ProgID as string, Optional Nodo As Variant) As Variant

Ej:

```
Dim miOPCServer as OPCServer

Set miOPCServer=OPCServer.Connect("ServidorConcreto")
```

ProgId: El nombre del servidor concreto según está registrado en el Registro del sistema.
Nodo: Identificador de ordenador remoto

OPC *Automation.* *OPCSever* **Descripción Detallada**

OPCServer

Métodos:
- GetOPCServers
- Connect
- Disconnect
- CreateBrowser

OPCserver.Disconnect. Desconecta el objeto **OPCServer** del servidor servidor concreto al que esté conectado.

Disconnect

Ej:

```
Dim miOPCServer as OPCServer

miOPCServer.Disconnect
```

Una vez desconectados, el Objeto **OPCServer** puede ser conectado a cualquier otro servidor que se decida.

OPC *Automation.* *OPCSever* **Descripción Detallada**

OPCServer

Eventos:

- **ServerShutDown**

ServerShutDown. Este evento es señalizado por el servidor **OPC**, cuando por algún motivo, el servidor está planeando cesar su funcionamiento y quiere hacerle saber este hecho a los clientes que tenga conectados.

Para utilizarlo, el objeto Servidor (**OPCServer**) tiene que haber sido declarado, a nivel de módulo, con la palabra clave **WithEvents**:

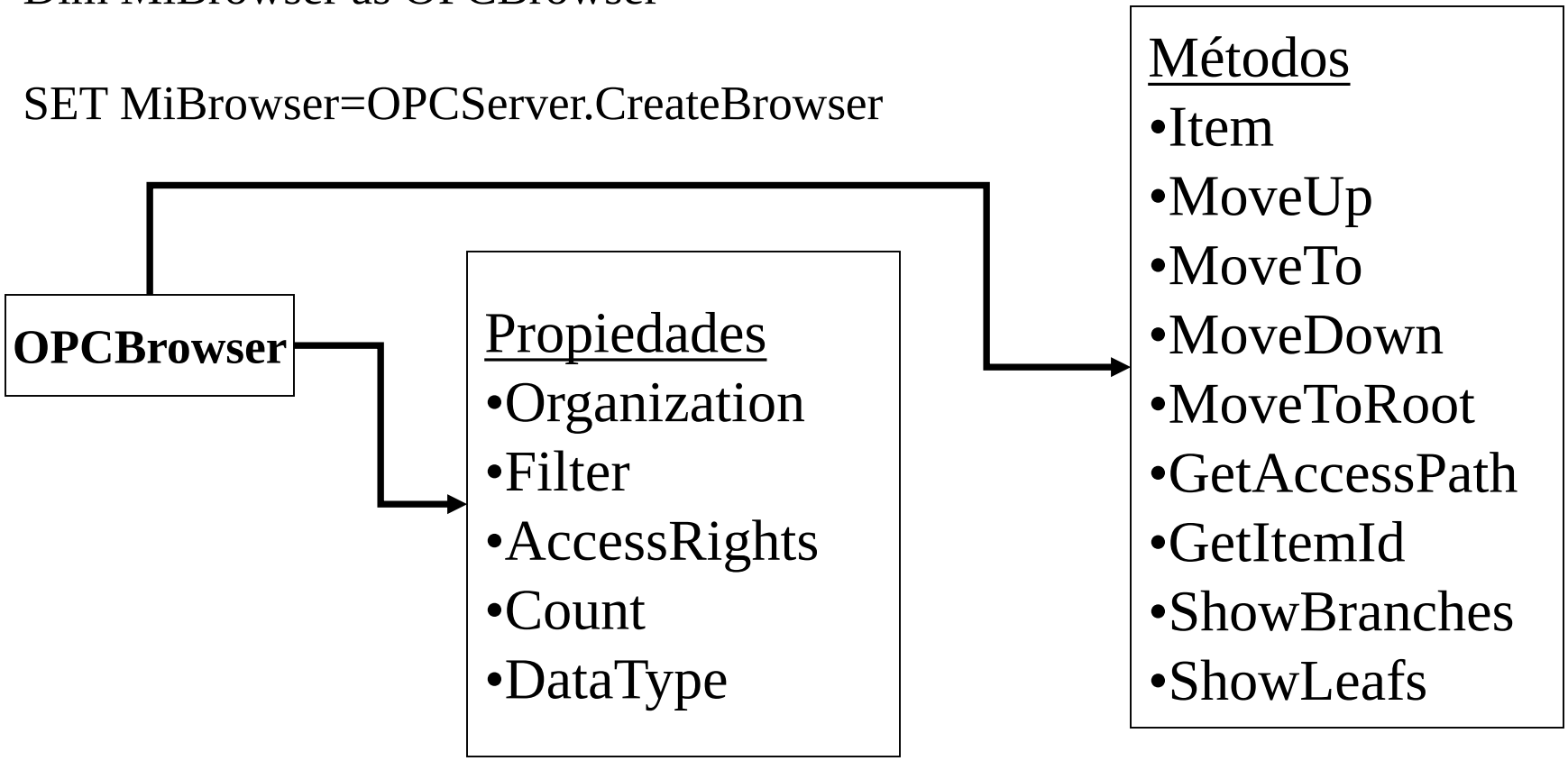
```
Dim withEvents miServer as OPCServer

Private Sub miServer_ServerShutDown(ByRef razon As String)

    'Se escribe código para liberar recursos y finalizar de
    'buena manera
End Sub
```

OPC *Automation.* *OPCBrowser*. **Descripción Detallada**

Objeto Opcional. De estar soportado se crea a partir del método **CreateBrowser** del objeto **OPCServer**. El objeto **OPCServer** debe estar conectado previamente a un servidor concreto.

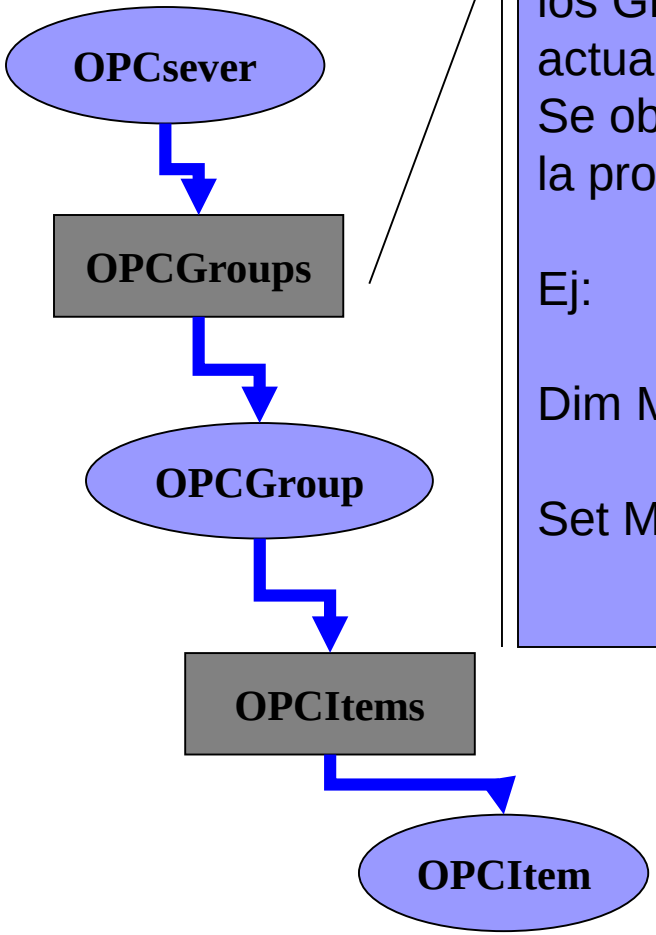
Dim MiBrowser as OPCBrowser

SET MiBrowser=OPCServer.CreateBrowser



OPC *Automation.* *OPCGroups*. **Descripción detallada**

Jerarquía de OPC



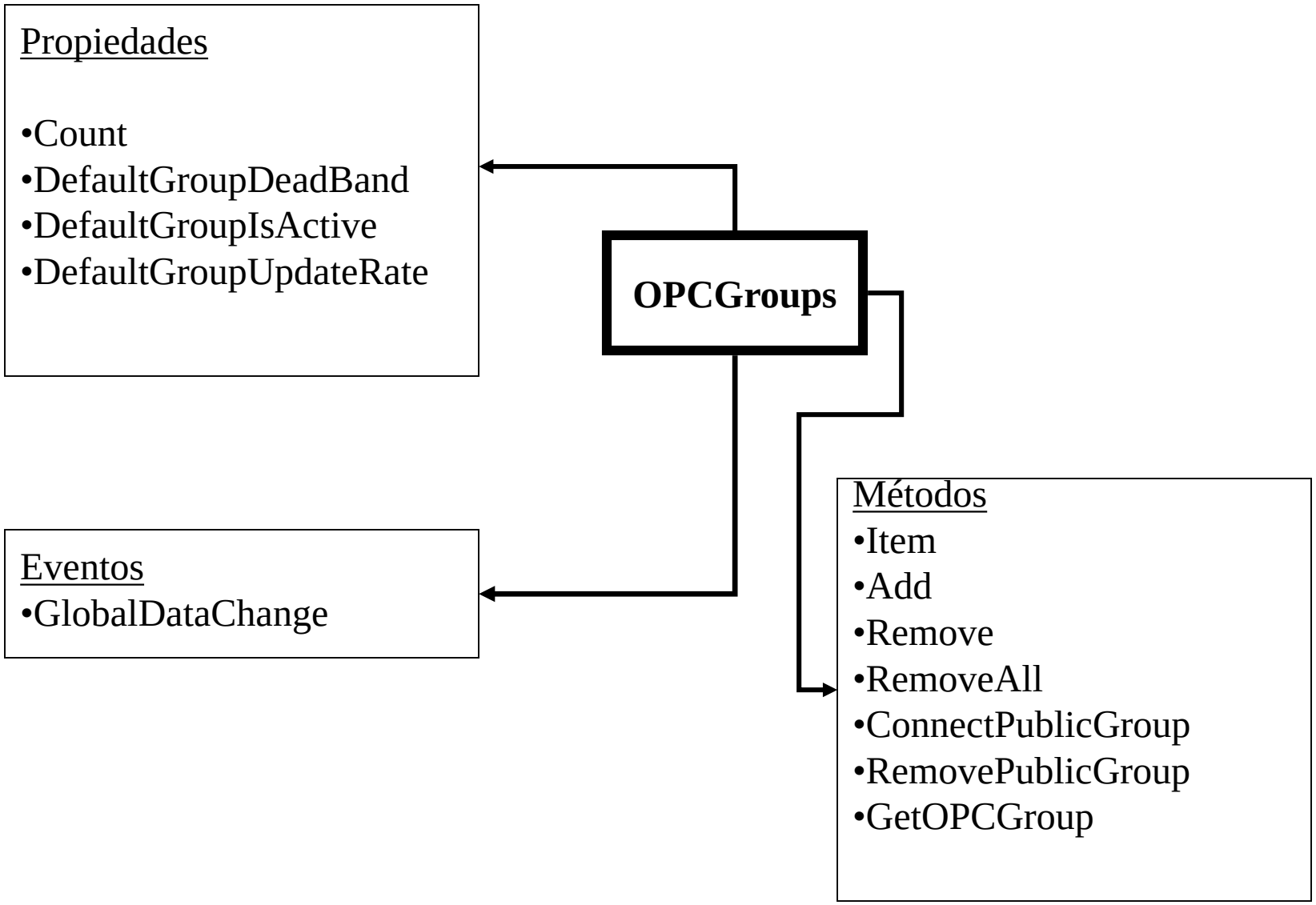
OPCGroups: Es un objeto colección, que contiene a todos los Grupos *OPC* que el cliente ha creado en la sesión actual con el servidor *OPC* (**OPCServer**).
Se obtiene una referencia al objeto **OPCGroups** mediante la propiedad **OPCServer.OPCGroups**.

Ej:

Dim MiServer as OPCServer, MisGrupos as OPCGroups

Set MisGrupos=MiServer.OPCGroups

OPC *Automation.* *OPCGroups*. **Descripción detallada**



OPC *Automation.* *OPCGroups*.

OPCGroups

Propiedades

- Count
- DefaultGroupDeadBand
- DefaultGroupIsActive
- DefaultGroupUpdateRate

Count: (Long) Propiedad típica de las colecciones. La cantidad de grupos *OPC* existentes.

Ej

```
For i=1 to MisGrupos.Count
        'Hacer algo con cada grupo
Next i
```

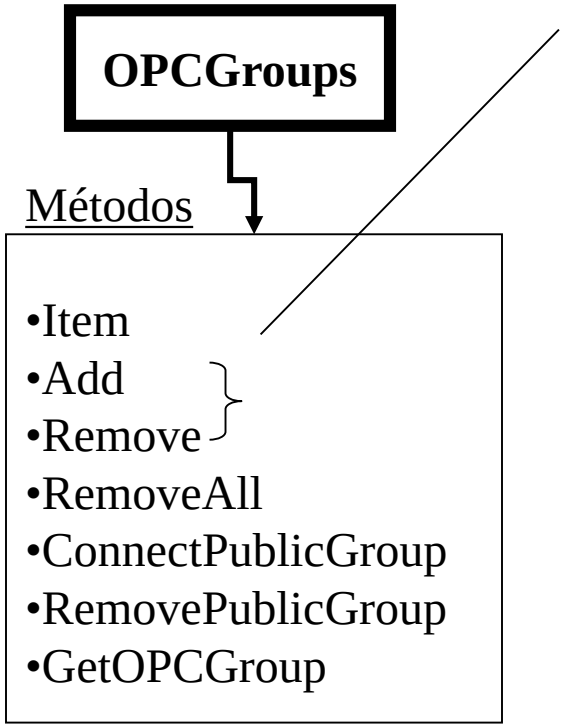
Default*XXX*: Hay un grupo de propiedades que especifican los valores que por defecto tomarán determinadas propiedades del los grupos que se creen.

DefaultGroupIsActive: Valor por defecto de **IsActive**.

DefaultGroupDeadBand: valor por defecto de **DeadBand.**

DefaultGroupUpdateRate: Valor por defecto **UpdateRate**

OPC *Automation.* *OPCGroups*. **Métodos**



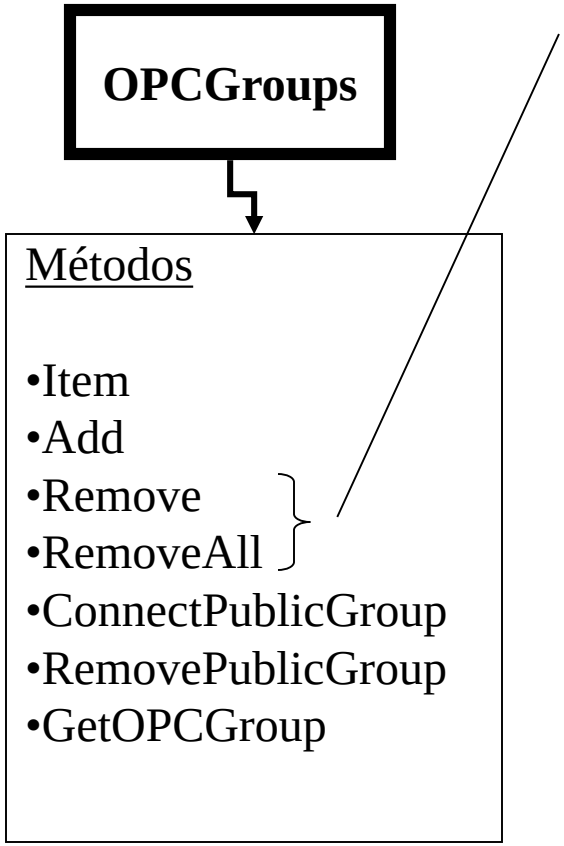
Item: Método típico de las colecciones que devuelve un elemento
(**OPCGroup**) de la colección.

```
Dim MiGrupo as OPCGroup, MisGrupos as OPCGroups

Set MiGrupo=MisGrupos.item(1)
```

Add: Método típico de las colecciones que añade un nueve elemento
(en este caso **OPCGroup**) a la colección.

```
Set MiGrupo= MisGrupos.Add("NombreGrupo")
```

OPC *Automation.* *OPCGroups*. **Métodos**



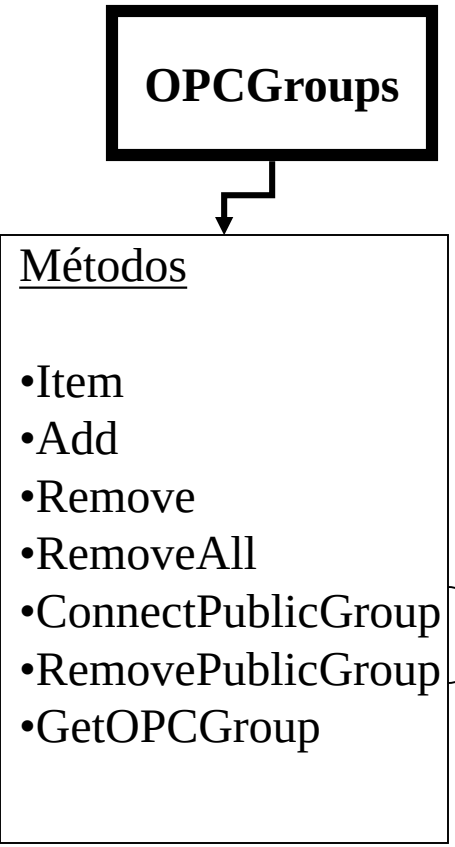
Remove: Método típico de las colecciones que elimina un elemento
de la colección.

```
Dim MiGrupo as OPCGroup, MisGrupos as OPCGroups

MisGrupos.Remove(1)
```

RemoveAll: Método típico de las colecciones elimina todos los
elementos de la colección.

```
MisGrupos.RemoveAll
```

OPC *Automation.* *OPCGroups*. **Métodos**



ConnectPublicGroup: Los grupos públicos preexisten en el servidor.
Un cliente se conecta al grupo público mediante este método.

```
Dim MiGrupo as OPCGroup, MisGrupos as OPCGroups

Set MiGrupo=
        MisGrupos.ConnectPublicGroup("NombreGPub")
```

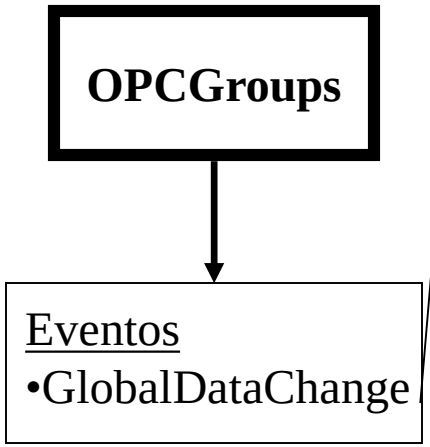
RemovePublicGroup: Determina que dejemos de estar conectados al Grupo Público. Ej:

```
        MisGrupos.RemovePublicGroup("NombreGPub")
```

GetOPCGroup: Permite obtener un grupo, identificándolo por su nombre:

```
Dim unGrupo as OPCGroup, MisGrupos as OPCGroups

Set unGrupo = MisGrupos.GetOPCGroup("Nombre_grupo")
```

OPC *Automation.* *OPCGroups*. **Descripción detallada**



GlobalDataChange: Mediante este evento, se comunica al cliente
La ocurrencia de un cambio de datos en alguno de los grupos que
se hayan definidos.

Se recomienda que se utilice los eventos de los Grupos específicos
(**OPCGroup**).

OPC *Automation.* *OPCGroup*. **Descripción detallada**

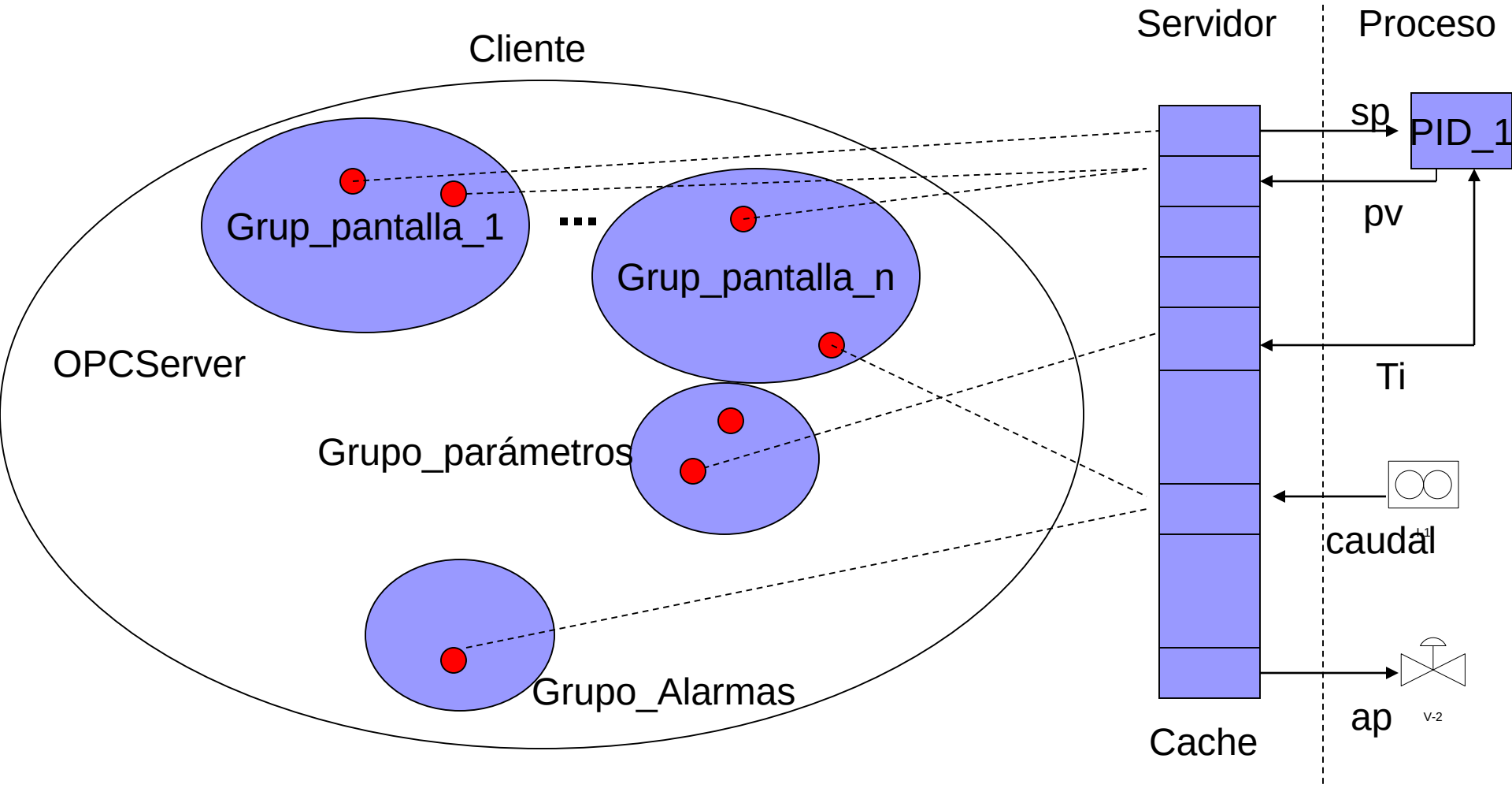
El Objeto **OPCGroup** es el objeto clave de la Jerarquía *OPC*.

Agrupa los *items* que el cliente (por algún criterio) haya decido agrupar para tratarlos como una unidad.

```
Dim MisGrupos as OPCGroups, MiGrupo as OPCGroup

Set MiGrupo= MisGrupos.Add("MiNombre")
```

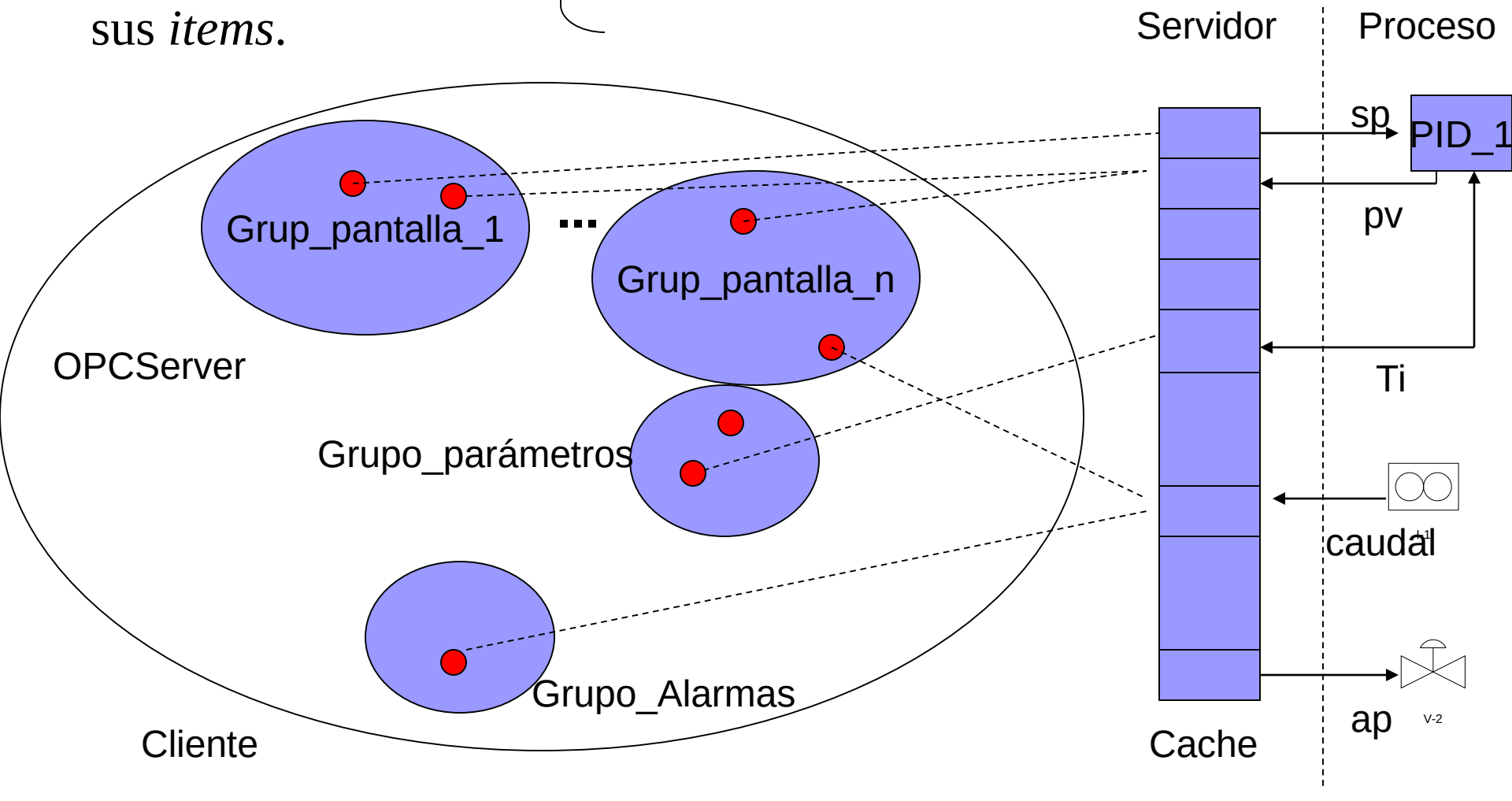
El grupo **MiNombre** se habrá creado. Algunas propiedades (**UpdateRate**, **IsActive**, **DeadBand**) tendrán por defecto los valores especificados en las Propiedades MisGrupos.Default*XXX*.

- El cliente crea tantos grupos como convenga a su aplicación.
- Cada grupo contiene lógicamente los *items*.



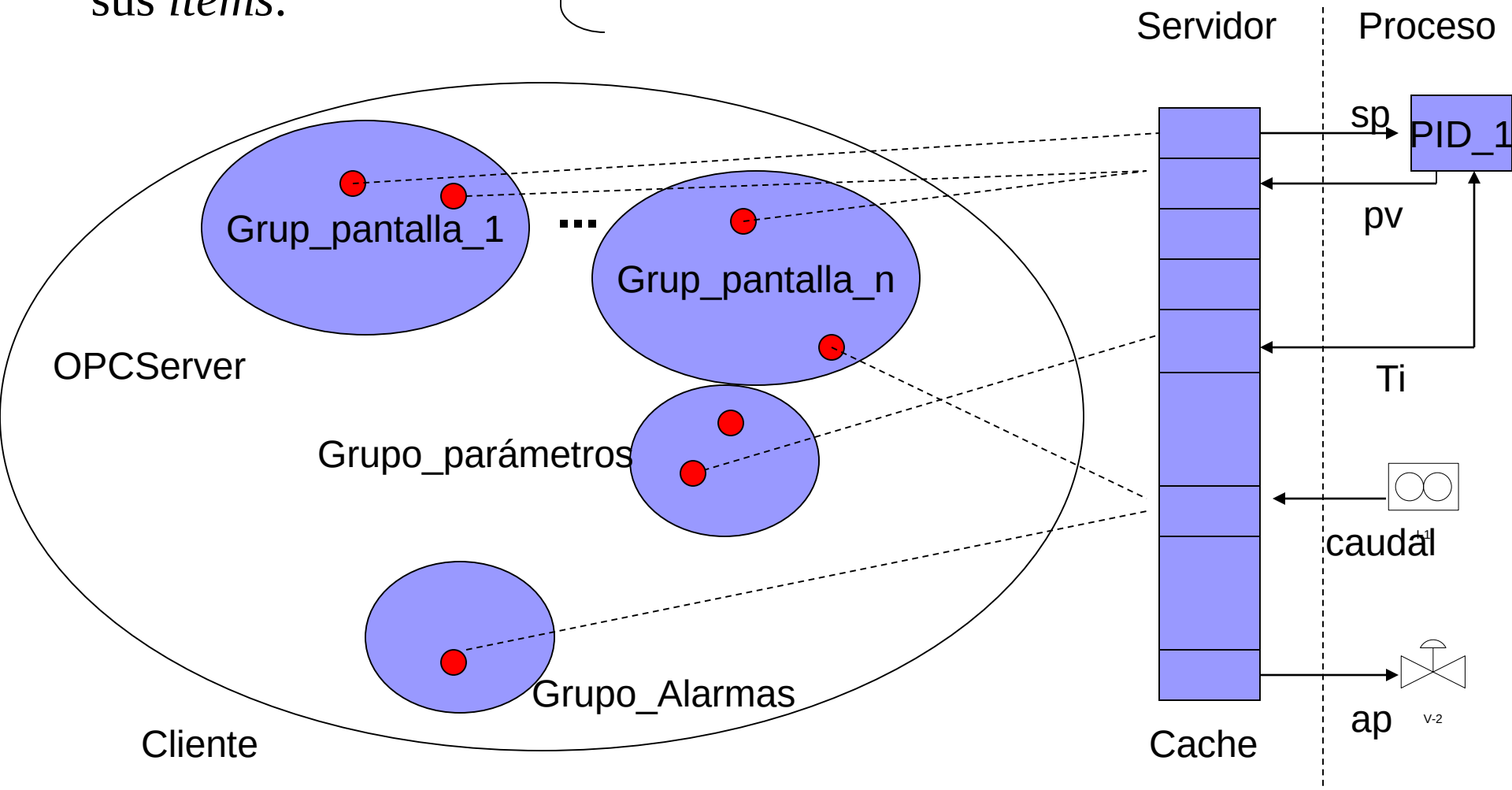
Cada grupo tendrá necesidades diferentes de lectura/escritura de sus *items*.

- Activo (si hay que actualizar o no)
- Periodicidad.
- Frecuencia (si es periódica).



Cada grupo tendrá necesidades diferentes de lectura/escritura de sus *items*.

•Hay que intentar "ahorrar" *ancho de banda*: no exigir frecuencias de actualización innecesariamente altas.



Se trata de **leer/escribir** datos. OPC describe varias alternativas. ¿Quién inicia el intercambio?

Cliente

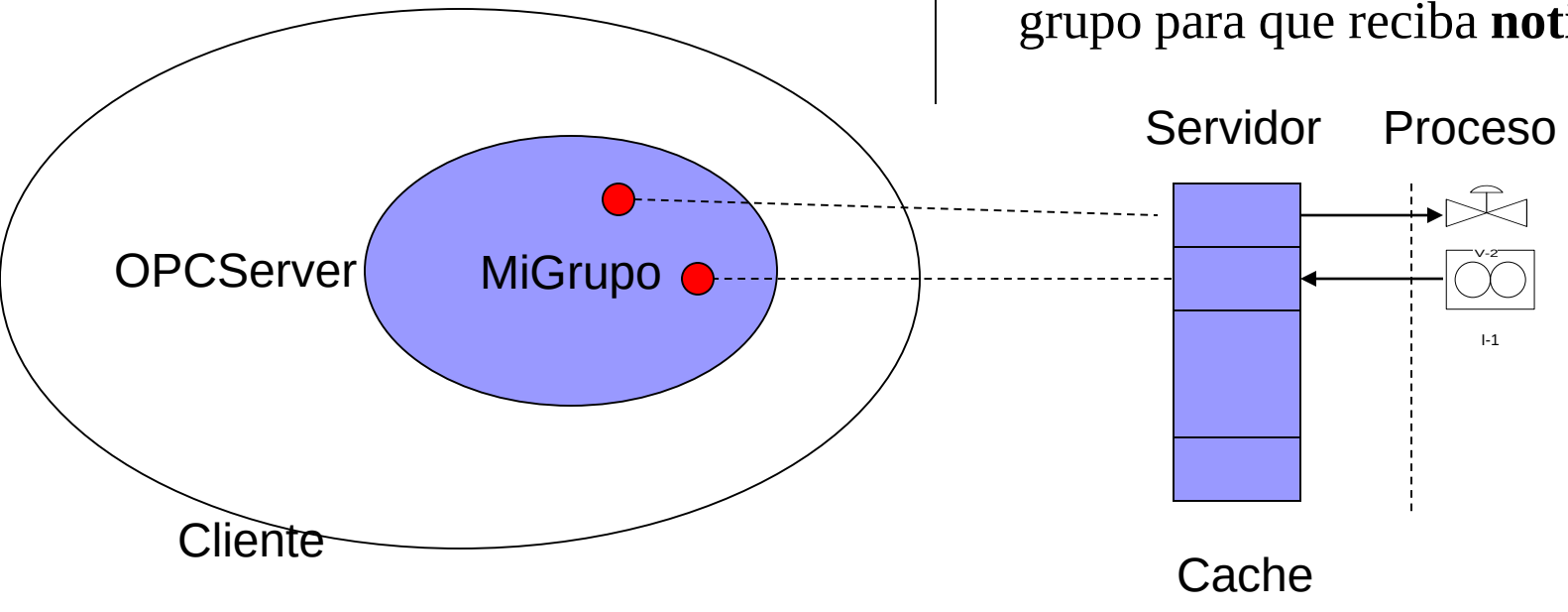
- Lectura/Escritura síncrona.
- Lectura/Escritura asíncrona.

El **cliente** decide cuando quiere leer/escribir: ejecuta los **métodos** apropiados del grupo en cuestión.

Servidor

El servidor **notifica** o "decide" enviar datos al cliente cuando cambian los datos. Lo hace mediante los eventos del **OPCGroup**.

Pero antes el cliente ha tenido que declarar (a través de **Propiedades** del grupo) que desea ser informado por el servidor: ha tenido que suscribir el grupo para que reciba **notificaciones.**



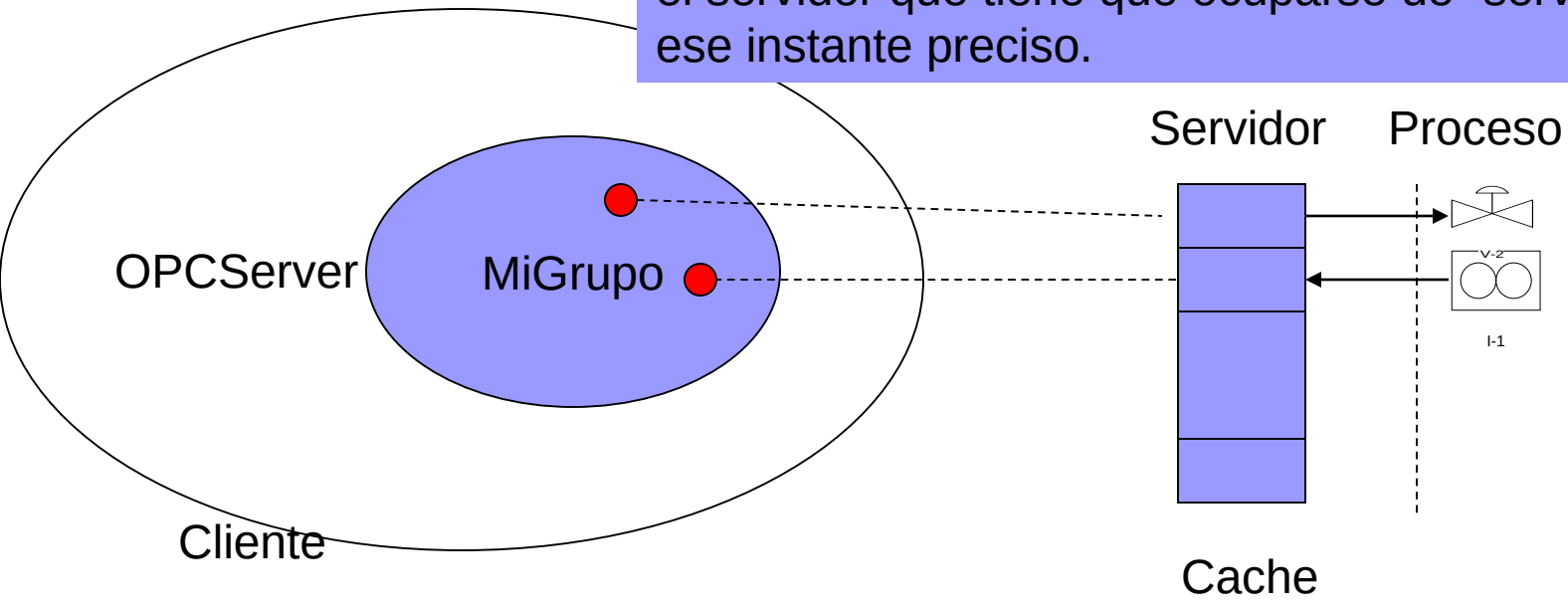
Se trata de **leer/escribir** datos. OPC describe varias alternativas. Diferencia entre síncrono y asíncrono.

Cliente

•Lectura/Escritura síncrona: cliente ejecuta método de **OPCGroup** (llamada a función) y en la línea siguiente de código ya tiene el resultado de la lectura/escritura: Se espera hasta que la operación se complete

Servidor

Mucho más fácil. Ineficiente para el cliente que tiene que esperar a que la operación termine y para el servidor que tiene que ocuparse de “servir” datos en ese instante preciso.



Se trata de **leer/escribir** datos. OPC describe varias alternativas. Diferencia entre síncrono y asíncrono.

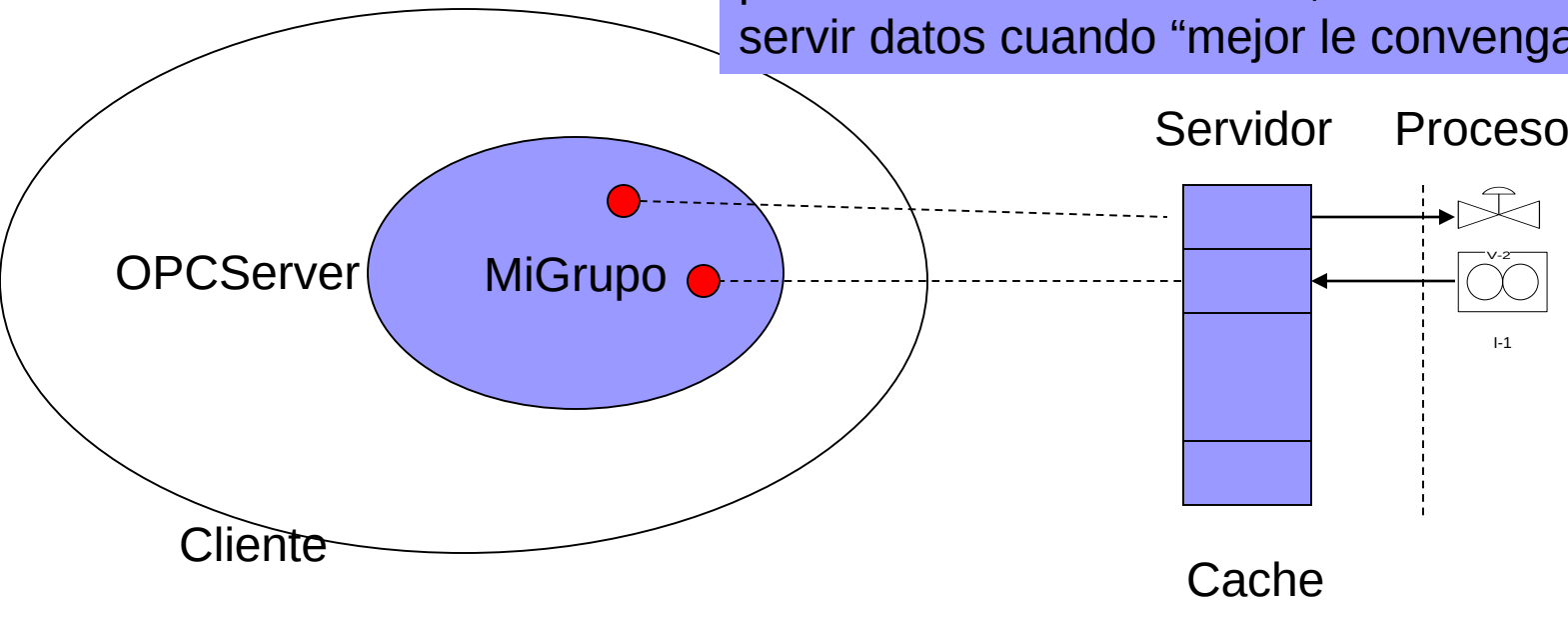
Cliente

•Lectura/Escritura asíncrona: cliente ejecuta método de **OPCGroup** (llamada a función) dando la orden de la operación. En la línea siguiente de código **no** se tiene el resultado de la lectura/escritura.

Servidor

El servidor recibe la orden de la operación y cuando la haya procesado, envía un evento del **OPCGroup** para entregar los resultados.

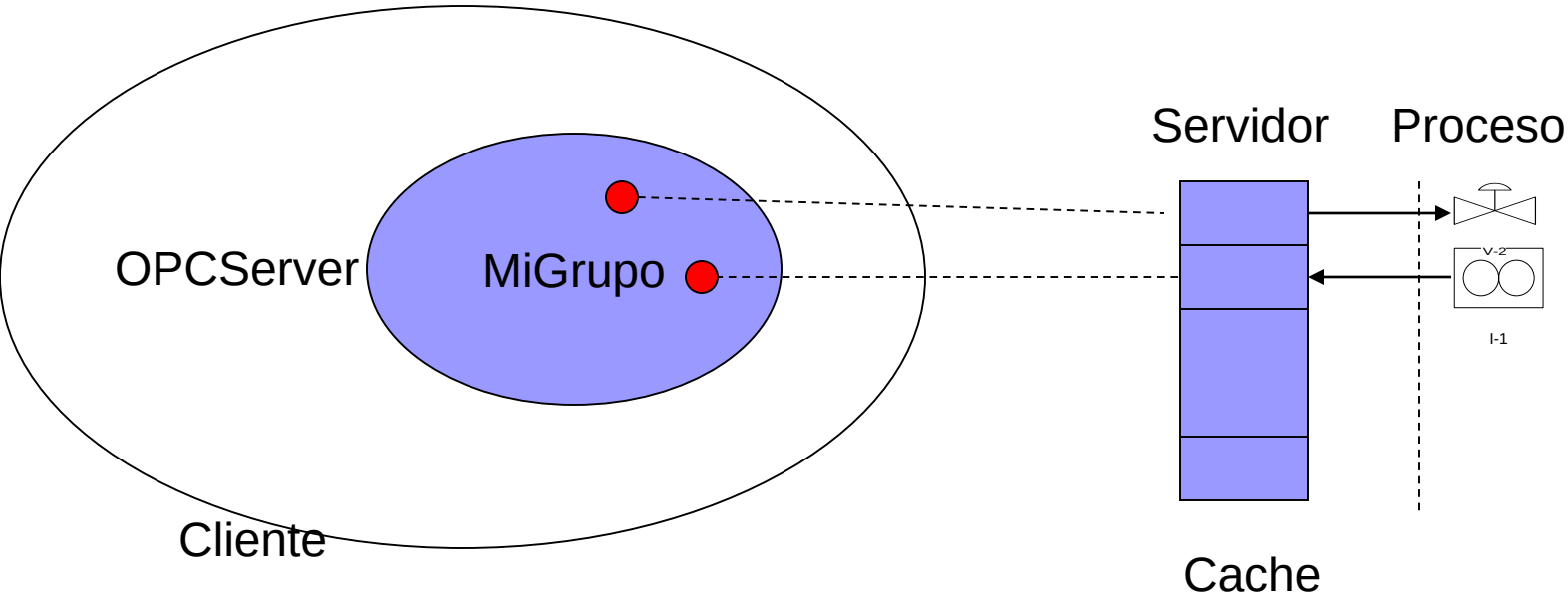
Compleja de programar. Eficiente: el cliente puede realizar otras tareas, el servidor puede servir datos cuando “mejor le convenga”



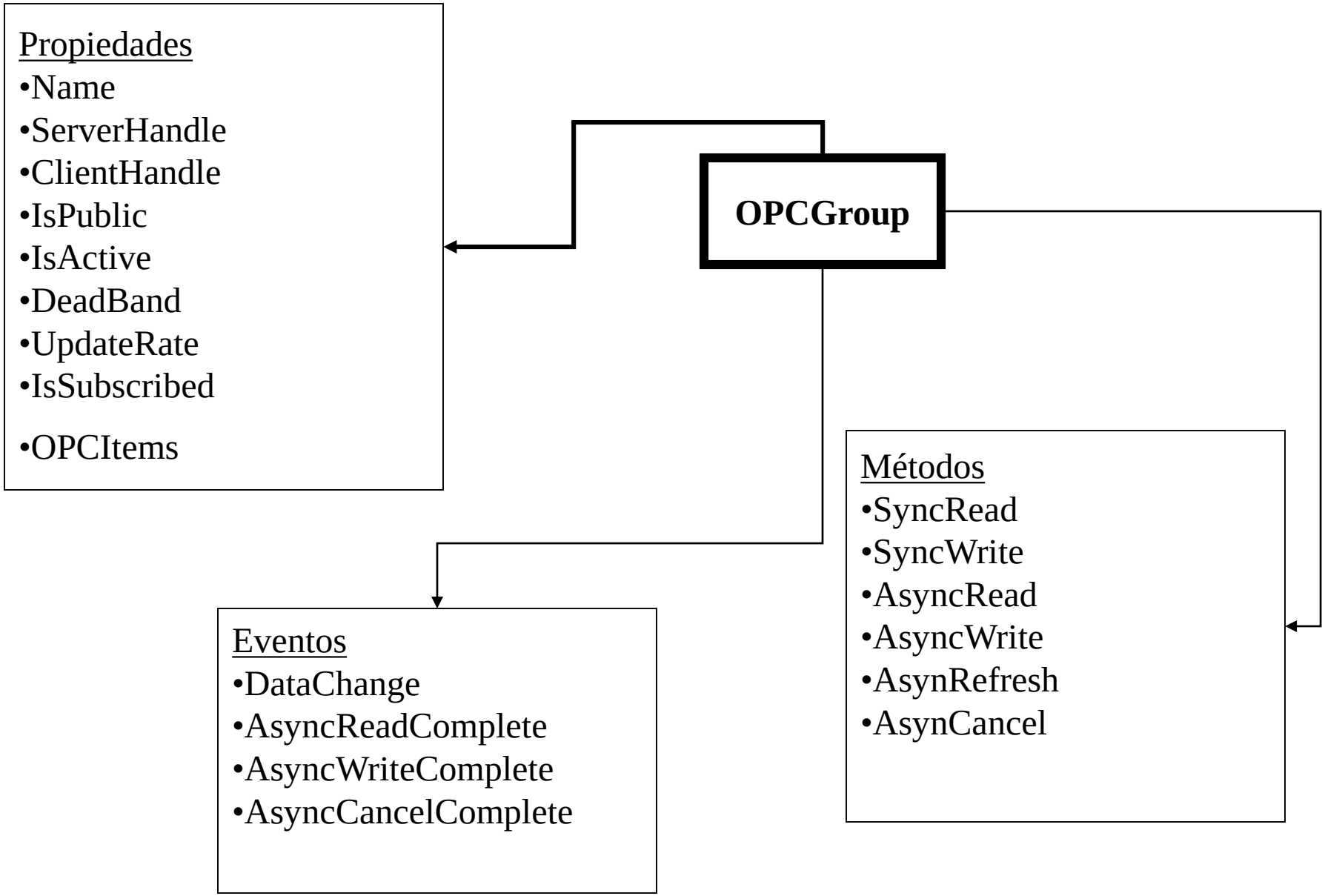
Se trata de **leer/escribir** datos. OPC describe varias alternativas. ¿A qué valor se accede?

- **A la variable física**: más lento pero se tiene la garantía de leer el valor actual o de que el valor escrito vaya directamente al dispositivo.

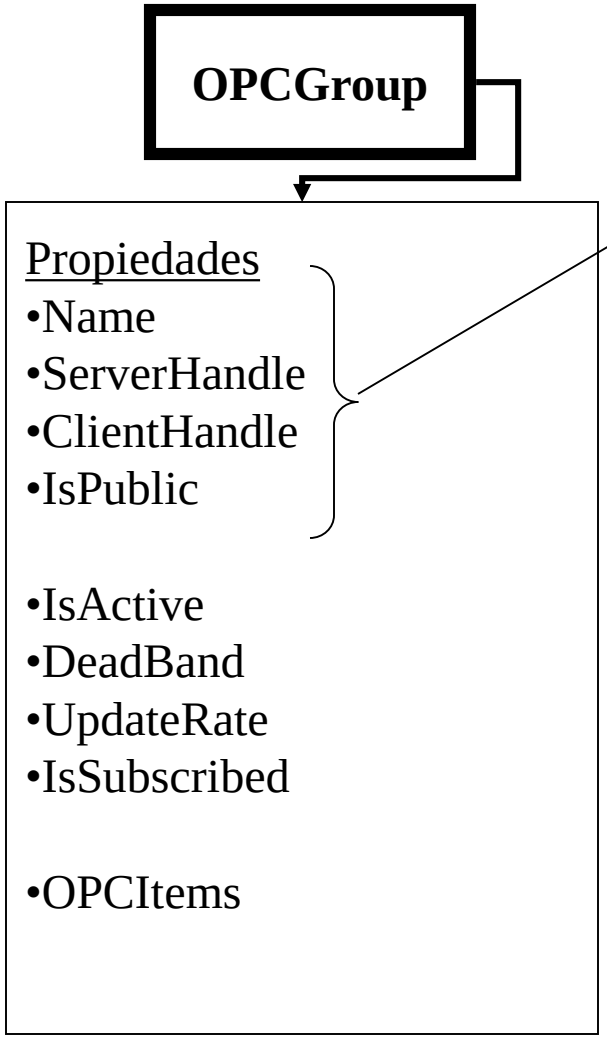
- **Al valor almacenado en la cache**: mucho más rápido (sobrecarga menos el servidor) pero sólo es exacto en los límites definidos por la configuración (las propiedades) del **OPCGroup** de que se trate.



OPC *Automation.* *OPCGroup*.

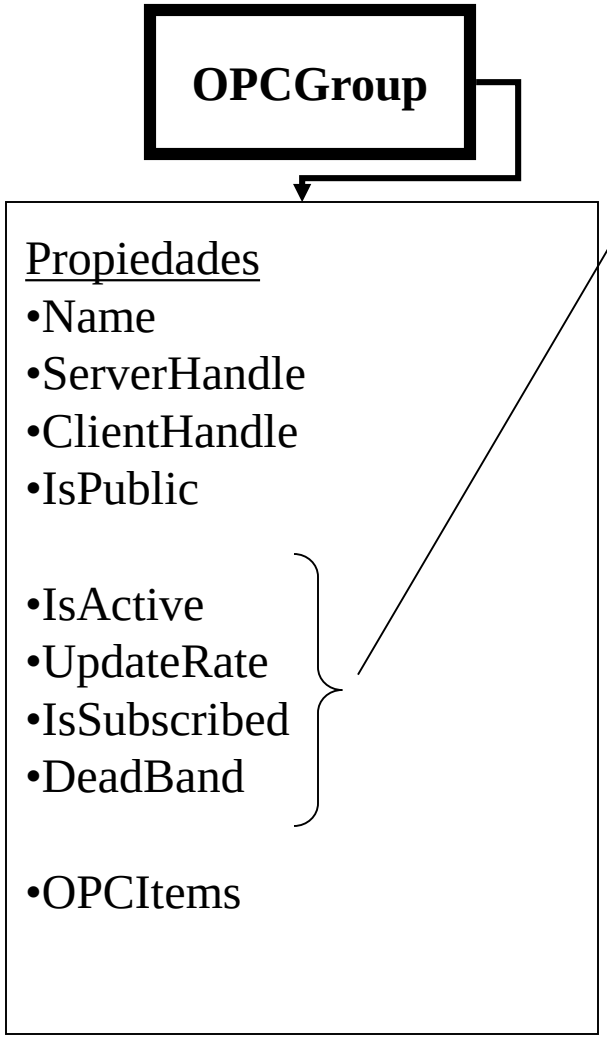


OPC *Automation.* *OPCGroup*.



Name (String) Escritura-Lectura: Un nombre (arbitrario) que el cliente puede dar al Grupo.
ClientHandle (Long) Escritura-Lectura: Un número de identificación del grupo que el cliente escoge.
ServerHandle (long) Lectura: Número de identificación de Grupo escogido por el servidor.
IsPublic (Boolean) Lectura. Retorna **True** si el grupo es Público.

OPC *Automation.* *OPCGroup*.



IsActive (Boolean) Escritura-Lectura. Controla si el grupo está activo. A un grupo inactivo, el servidor no envía notificaciones de Cambio de datos (Evento **DataChange**). De todas formas si admite Lecturas/escrituras que el cliente solicite.

IsSubscribed (Boolean) Escritura-Lectura: Determina si el cliente requiere que el servidor le notifique de los posibles cambios en los valores de los *items* que contiene el Grupos.

UpdateRate (Long) Escritura-Lectura: Un número en milisegundos mediante el cual el cliente solicita la frecuencia a la que el cliente desea que el servidor le informe de posibles cambios de datos. Esta propiedad es una petición. El servidor puede cumplir o no con esa petición. En cualquier caso no habrán **notificaciones** a una frecuencia mayor que la especificada en esta propiedad.

OPC *Automation.* *OPCGroup*.

OPCGroup

Propiedades
- Name
- ServerHandle
- ClientHandle
- IsPublic

- IsActive
- UpdateRate
- IsSubscribed
- DeadBand

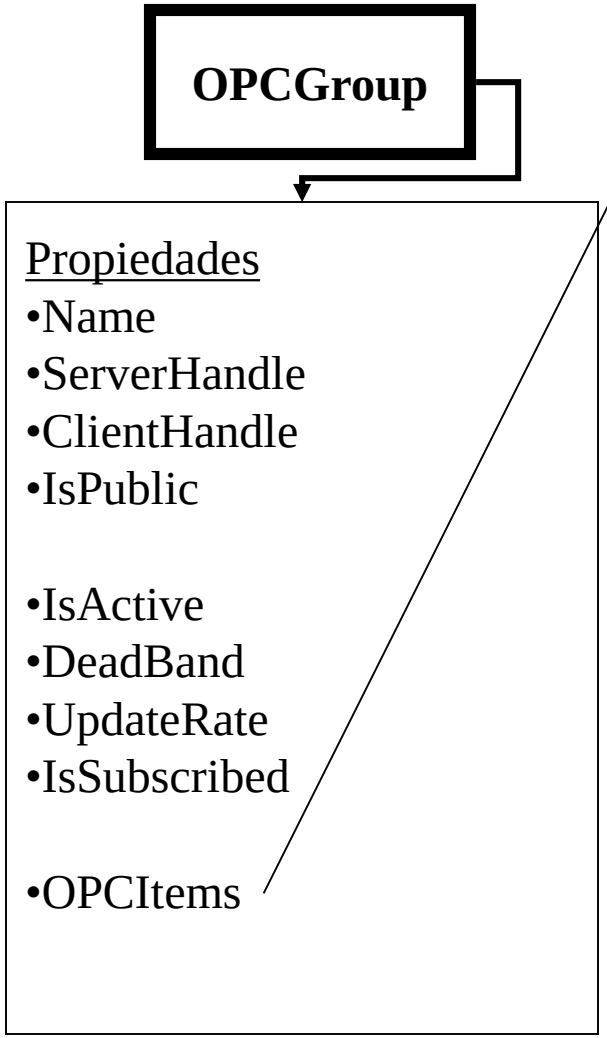
- OPCItems

DeadBand (Single) Escritura-Lectura. Especifica que cambio de dato se considera suficiente como para generar un evento **DataChange**. Para que esta propiedad sea útil, al menos algunos de los *items* del Grupo debe tener definida las propiedades (opcionales) **EUType** y **EUInfo.** Además **EUType** debe ser “1” o sea analógica. Cómo se verá más adelante **EUInfo** define valores máximos y mínimos de la variable.

Ocurriría una notificación si se cumple:

(|ValorCache-ValorDispositivoFísico| > RangoItem*__DeadBand__

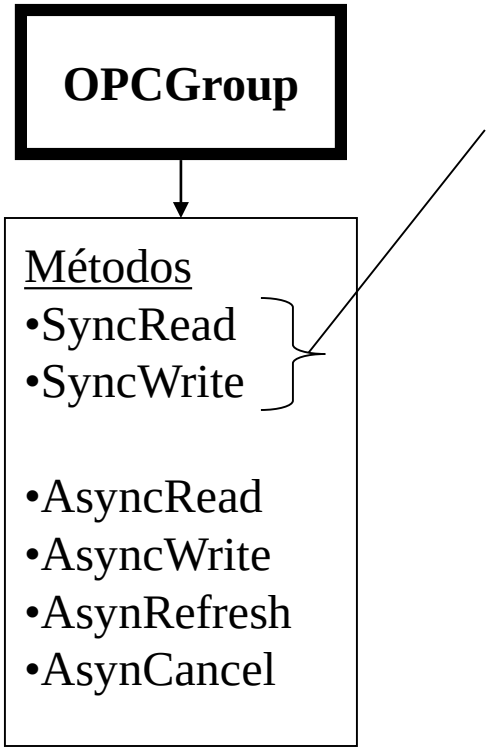
OPC *Automation.* *OPCGroup*.



OPCItems Lectura: Accede al objeto colección **OPCitems** que contiene los *items* definidos en el grupo.

```
Dim MisItems as OPCItems

Set MisItems = MiGrupo.OPCItems
```

OPC *Automation.* *OPCGroup*.

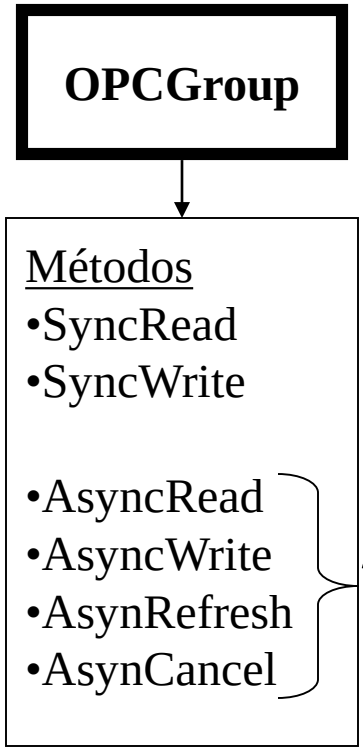


Lectura y Escritura **síncronas** de los datos representados por los *items* del grupo.

Lectura: El cliente ejecuta el método **SyncRead** y cuando la sentencia se ejecuta, ya tiene los resultados de los valores leídos.

Escritura: Cuando el cliente ejecuta **SyncWrite** tiene la seguridad que los datos han sido escritos en el dispositivo físico.

OPC *Automation.* *OPCGroup*.



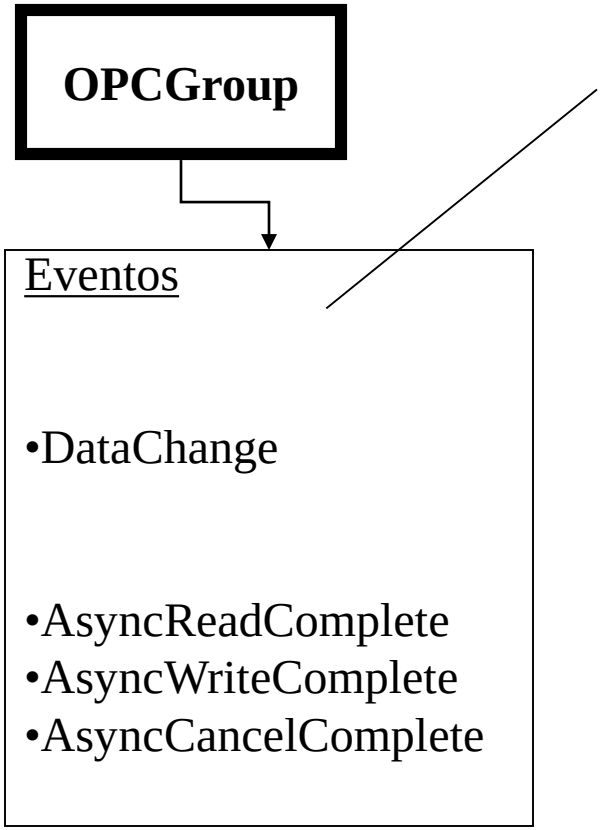
Lectura, Escritura y Refrescamiento **asíncronos** de los datos representados por los *items* del grupo.

El cliente solicita la operación y puede inmediatamente pasar a ejecutar otras sentencias sin obtener los resultados de la operación.

Cuando los resultados estén listos el servidor lo comunicará mediante el evento correspondiente.

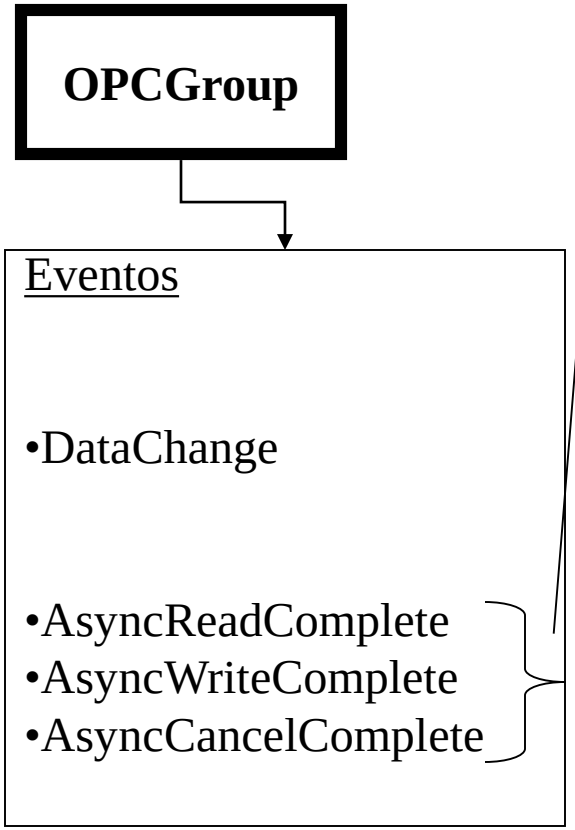
AsynCancel: Mediante la ejecución de este método el cliente puede cancelar una operación pendiente: que ya se haya solicitado (mediante **AsynRead**, **AsynWrite** o **AsynsRefresh**) y que todavía no haya sido ejecutada por el servidor.

OPC *Automation.* *OPCGroup*.



Evento que es invocado por el servidor para notificar al cliente de los valores de los *items* del grupo. Para que el evento se dispare el grupo deberá estar activo y suscrito al servicio (Propiedades **IsActive** y **IsSubscribed** activadas**),** alguno de los datos habrán cambiado (el cambio será mayor que lo especificado en **DeadBand**) y habrá transcurrido, al menos, la cantidad de milisegundos especificados en **UpdateRate**.

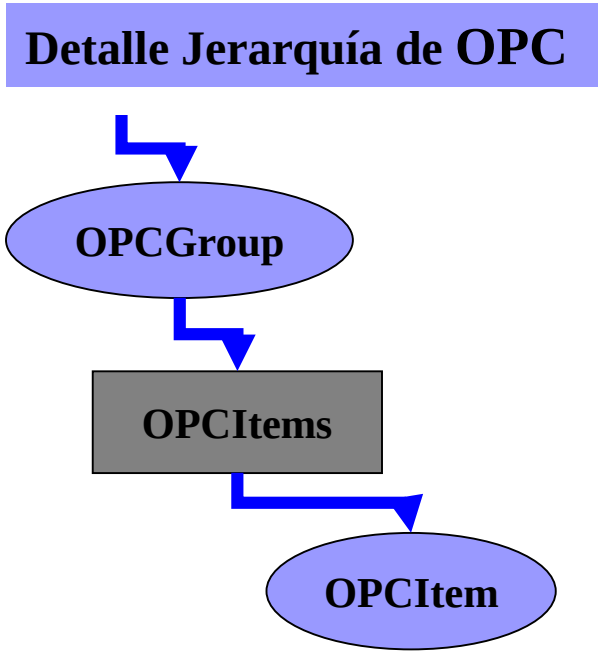
OPC *Automation.* *OPCGroup*.



Eventos invocados por el servidor para indicar al cliente que las operaciones correspondientes (lectura, escritura o cancelación) que ha solicitado previamente (mediante las llamadas a los métodos **AsyncRead**, **AsyncWrite** y **AsyncCancel**) han sido ejecutadas por el servidor.

OPC *Automation.* *OPCItems*.

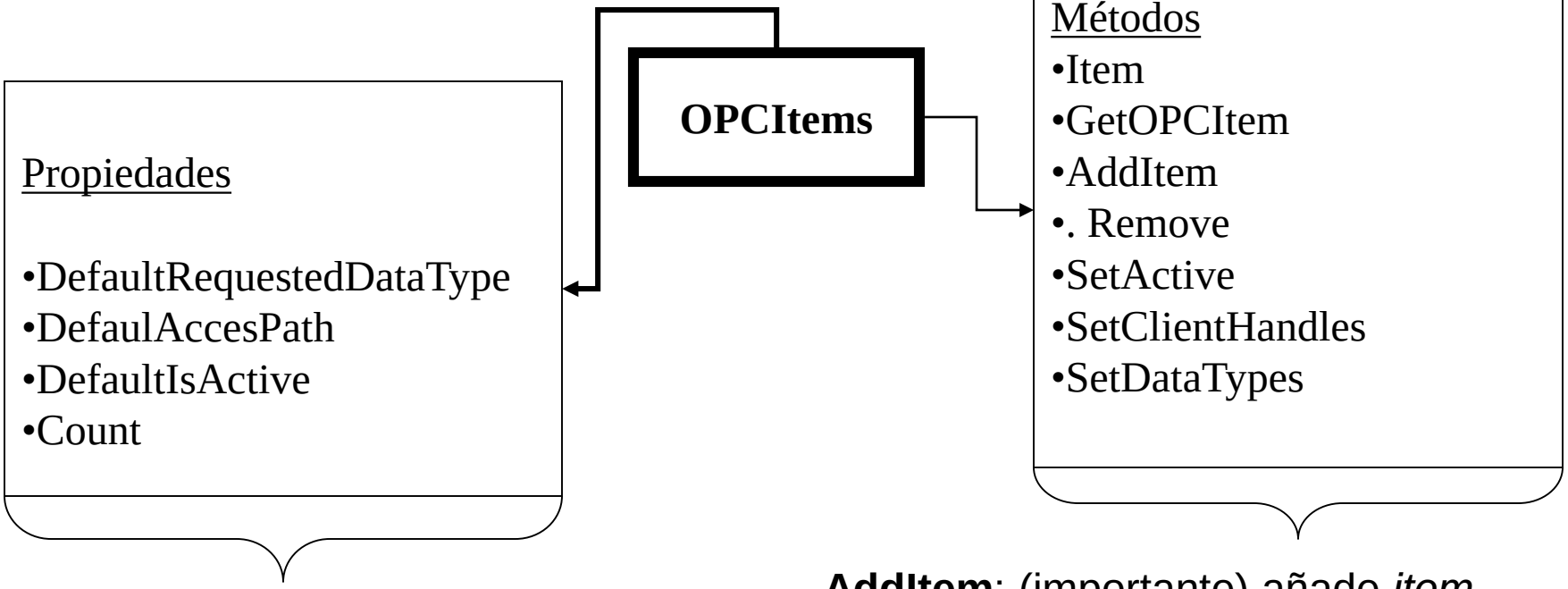
¿Cómo se agrega un *item* a un Grupo?



Agregándolo a la colección **OPCItems** de ese grupo. A dicha colección se accede a través del método **OPCItems** del objeto **OPCGroup** correspondiente.

```
Dim MiGrupo as OPCGroup
Dim MisItems as OPCItems

....
Set MisItems = MiGrupo.OPCItems
```

OPC *Automation.* *OPCItems.*

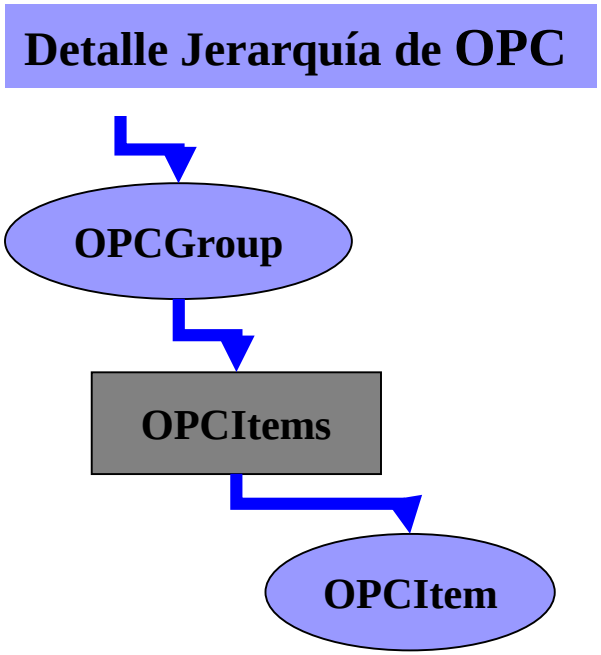


Count: cantidad de *items* en la colección
Default*XXX*: Valor por defecto que tendrán
las correspondientes propiedades de los nuevos items que se agreguen a la colección.

AddItem: (importante) añade *item* a la colección.
Item: accede a un *item* a través de su índice.
Set*XXX*: Permite cambiar las correspondientes propiedades de un conjunto de *items*.
GetOPCItem: Devuelve el objeto **OPCItem** que corresponde a determinado *ServerHandle.*

OPC *Automation.* *OPCItems*.

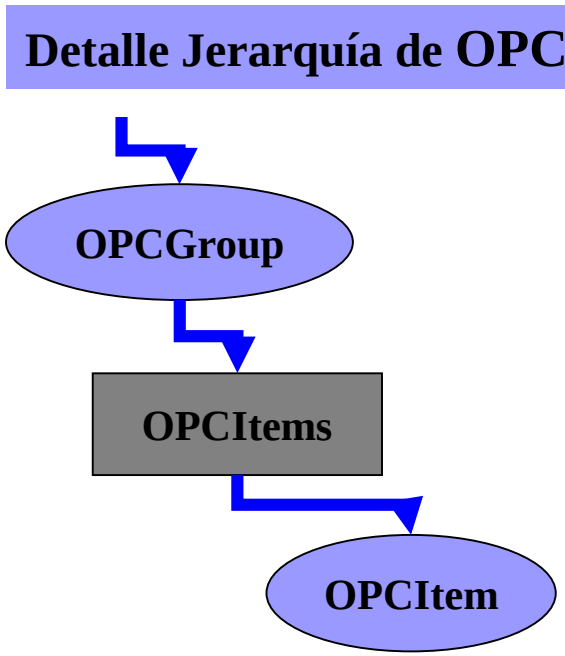
¿Cómo se agrega un *item* a un Grupo?



Agregándolo a la colección **OPCItems** de ese grupo, mediante su método **AddItem** (agrega un solo *item*) o **AddItems** (agrega varios *items* de una vez).

OPC *Automation.* *OPCItems*.

Cuando se añade un *item* al grupo el **cliente** debe:



•Especificar a que *tag* se hace referencia (el nombre de la fuente de datos (“calderas.temp_col.pv”).
•Un ***ClientHandle***: es un número arbitrario (inventado por el cliente pero único dentro del grupo) que sirva para identificar el *item* a los efectos del cliente.

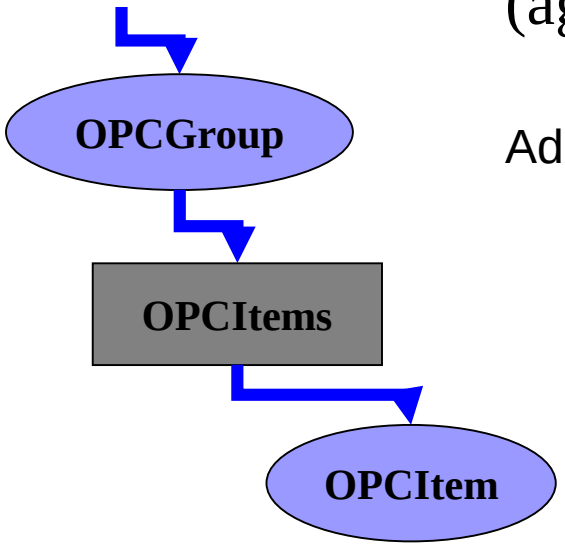
Cuando el cliente haya agregado con éxito el item a la colección:

•Le asignará al *item* un número de identificación (***ServerHandle***) que deberá a partir de ese instante ser presentado por el cliente al servidor para identificar el dicho *item*.

OPC *Automation.* *OPCItems*.

¿Pero, al final, cómo se agrega un *item* a un Grupo?

Detalle Jerarquía de OPC

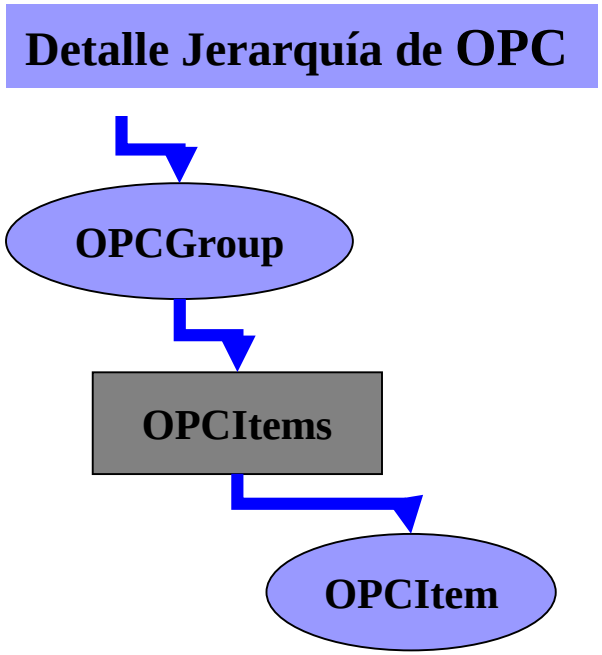


Agregándolo a la colección **OPCItems** de ese grupo, mediante su método **AddItem** (agrega un solo *item*) cuya sintaxis es.

AddItem(itemId as String, ClientHandle as Long) as OPCItem

OPC *Automation.* *OPCItems*.

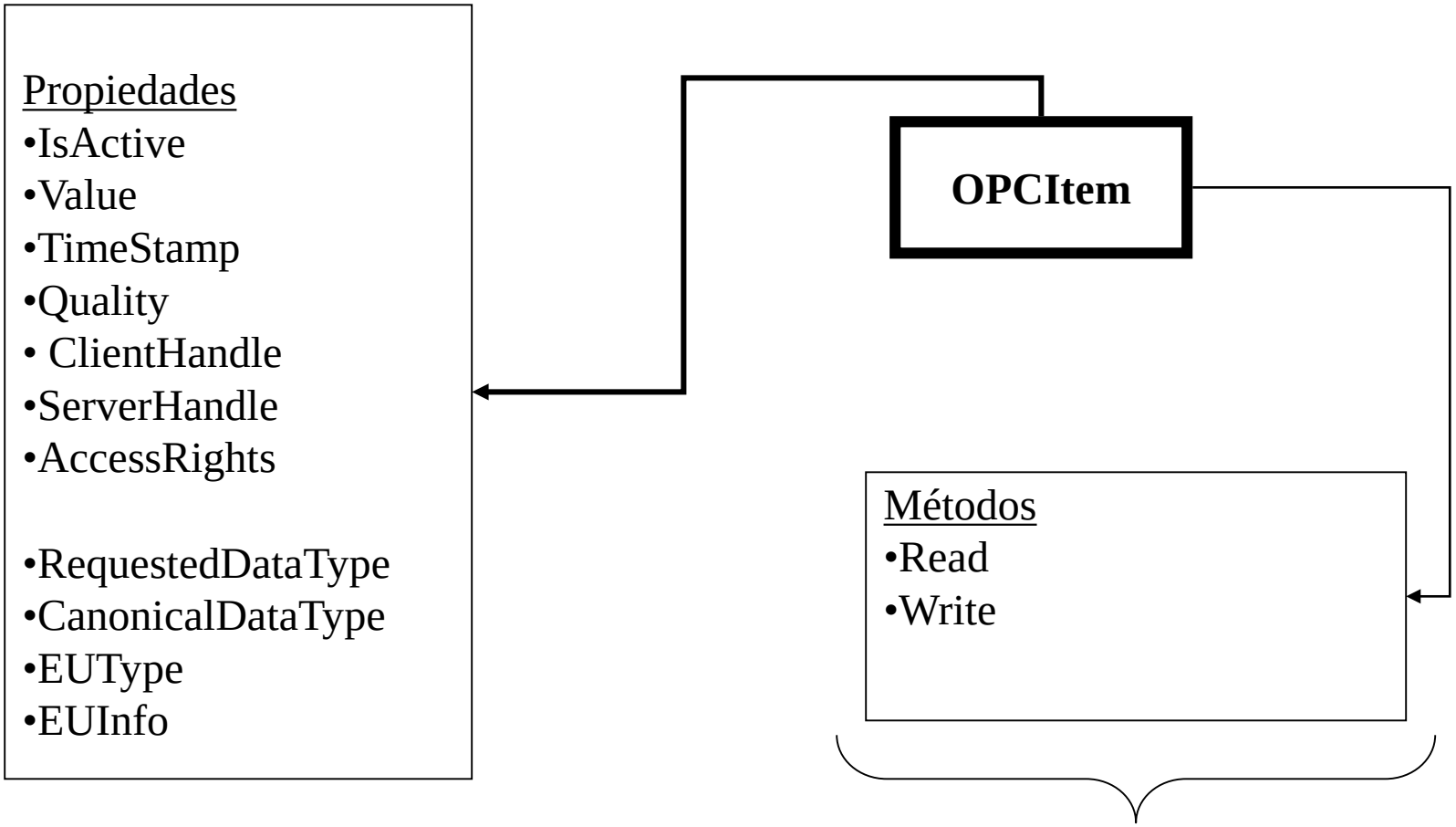
¿Cómo se agrega un *item* a un Grupo?



Agregándolo a la colección **OPCItems** de ese grupo, (mediante su método **AddItems**):

```
Dim MiGrupo as OPCGroup
Dim MisItems as OPCItems
Dim UnItem as OPCItem
Dim UnServerHandle as Long

Set MisItems=MiGrupo.OPCItems
Set UnItem =MisItems.AddItem("Calderas.Pcol_pv",1)
UnServerHandle=UnItem.ServerHandle
```

OPC *Automation.* *OPCItem*. **Descripción detallada**



Permiten Leer/escribir cada *item* de forman individual. **Son poco importantes.**

OPCItem

Propiedades
- IsActive
- Value
- TimeStamp
- Quality
- ClientHandle
- ServerHandle
- AccessRights

- RequestedDataType
- CanonicalDataType
- EUType
- EUInfo

IsActive (Boolean) Escritura-Lectura: Determina si el *item* está activo (un *item* activo envía notificaciones)

Value (Variant): El último valor leído.

TimeStamp (Date**)**: La marca de tiempo del último dato leído.

Quality (Long): La calidad del último dato leído.
OPCQualityBad
OPCQualityGood
OPCQualityUncertain

ClientHandle (Long): El identificador del cliente para el item.

ServerHandle(Long): El identificador del servidor para el item (muy importante hay que presentar en operaciones de lectura/escritura).
AccessRigths (Long): Si el *item* es de Lectura, o Escritura o Escritura/Lectura: **OPCReadable**, **OPCWritable** ó **OPCReadable** OR **OPCWritable** .

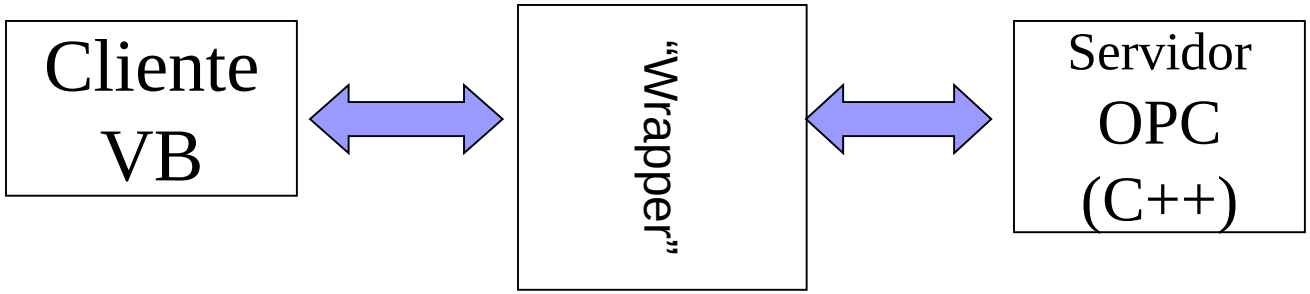
OPCItem

Propiedades
- IsActive
- Value
- TimeStamp
- Quality
- ClientHandle
- ServerHandle
- AccessRights

- CanonicalDataType
- RequestedDataType

- EUType
- EUInfo

CanonicalDataType (Integer**)**: El tipo de datos definido por el servidor.
RequestedDataType (Integer**)**: El tipo de datos que el cliente solicita para el *item.* Por defecto es **VT_Empty** lo cual significa que se solicita que los datos sean enviados con el Tipo de Dato en que está definidos en el Servidor o tipo de datos canónico.



Si el cliente "se empeña" en solicitar un tipo de datos para el que no hay conversión el **OPCItem** no es creado.

Traslada hacia el tipo de datos más aproximado sin perder exactitud

El servidor puede definir un *item* con un tipo de datos que no sea soportado por VB\Automation

OPC *Automation.* *OPCItem*. **Descripción detallada**

OPCItem

Propiedades
- IsActive
- Value
- TimeStamp
- Quality
- ClientHandle
- ServerHandle
- AccessRights

- CanonicalDataType
- RequestedDataType

- EUType
- EUInfo

EUType (Integer) Lectura : Propiedad opcional. Indica el tipo de información de unidades de Ingeniería (si existe):

0: No existe
1:Analógica
2: Enumerada (discreta)

EUInfo (Variant) : El tipo de datos definido por el servidor.

- Si **EUType** es 1: Matriz de 2 elementos (double) que contiene los valores Mínimo y Máximo del rango del *item*.

- Si **EUType** es 2: Matriz de cadena de caracteres que contiene una lista de etiquetas que corresponden a cada uno de los valores discretos que puede tener el *item*

OPC *Automation.*

Lectura/Escritura Síncronas

Lectura Síncrona.

Método de OPCGroup.

SyncRead: Realiza una lectura síncrona de los *items* indicados (no necesariamente todos los del grupo). La aplicación cliente queda bloqueada hasta que el método retorna.

```
SyncRead(Fuente as Integer, NumeroItems as Long, ServerHandles() as Long, ByRef Valores() as Variant, ByRef Errores() as Long, Optional ByRef as Calidades() as Variant, Optional ByRef TimeStamps() as Date)
```

Fuente:

- **OPC_DS_Cache**: Lee desde el almacenamiento secundario
- **OPC_DS_Device**: Lee desde el dispositvo físico.

ServerHandles() Matriz de los identificadores de *Item* que hemos obtenido durante la creación de cada uno.

Lectura Síncrona.

Método de OPCGroup.

```
SyncRead(Fuente as Integer, NumeroItems as Long,
ServerHandles() as Long, ByRef Valores() as Variant, ByRef
Errores() as Long, Optional ByRef as Calidades() as
Variant, Optional ByRef TimeStamps() as Date)
```

errores(): vector que indica si se produjo algún error a la hora de leer cada item individual. OPC define una serie de errores descritos en la librería de tipos por las constantes:

- OPCBadRights
- OPCBadType
- OPCInvalidHandle
- OPCInvalidItemID, etc

Escritura Síncrona.

Método de OPCGroup:

SyncWrite: Realiza una escritura síncrona de los *items* indicados (no necesariamente todos los del grupo). La aplicación cliente queda bloqueada hasta que el método retorna**.**

SyncWrite(NumeroItems as Long, ServerHandles() as Long, Valores() as Variant, ByRef Errores() as Long)

La escritura síncrona siempre se realiza sobre dispositivo físico.

OPC *Automation.* *OPCItem*.

- **Read**: Hace lectura síncrona del *Item*

  Read (Source As Integer, Optional ByRef Value As Variant, Optional ByRef Quality As Variant, Optional ByRef TimeStamp As Variant)

- **Write**: Hace escritura síncrona del item.

  Write (Value As Variant)

No se recomienda utilizar estos métodos sino los del Objeto OPCGroup.

Ejemplo lectura síncrona.

Se quiere leer los valores de dos *tags* de nombre *var1*, *var2*. Se quiere además que estas variables conformen un grupo llamado *control_valv*.

```
Dim Serv as OPCServer, ContGroup as OPCGroup

Dim cont_handles(2) as Long, valores(2) as Variant

Dim TimeStamps(2) as Dates, Calidad(2) as Variant, Errores(2) as Long

....

Set ContGroup =Serv.OPCGroups.Add(“control_valv”)

cont_handles(1)=ContGroup.OPCItems.Add(“var1”,1).ServerHandle

cont_handles(2)=ContGroup.OPCItems.Add(“var2”,2).ServerHandle

ContGroup.SyncRead OPC_DS_Cache, 2, cont_handles, valores, Errores,
Calidad, TimeStamps

MsgBox "Valor 1 ->" & valor(1), vbOKOnly
```

OPC *Automation.*

Lectura mediante Notificación

OPC *Automation.* *Notificación*

El evento **OPCGroup.DataChange** es señalizado cuando el servidor detecta cambios en uno o más *items* del grupo. El evento ocurre si se cumplen las condiciones indicadas en las propiedades **UpdateRate** y **DeadBand** de grupo. Ocurriría si el objeto **OPCGroup** está activo (**OPCGroup.IsActive**) y el *item* que ha cambiado también está activo **OPCItem.IsActive**.

DataChange (TransactionID As Long, NumItems As Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)

Observese que en este caso se identifican los *items* cuyos valores han cambiado mediante el ClientHandle. El identificador que dió el cliente durante la creación de cada *item*.

Este evento también es señalado para reportar los datos leídos con **AsyncRefresh**. En este caso se devolverán los valores de **todos** los *items* del grupo.

OPC *Automation.* *Notificación*

DataChange (TransactionID As Long, NumItems As Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)

Si **TransactionID** es *cero* entonces **DataChange** ha ocurrido como *notificación* porque:

- la propiedad **IsSubscribed** del grupo es ***True***.

- han transcurrido al menos **UpdateRate** milisegundos desde la última notificación y ha ocurrido algún cambio en los valores de los *items* **activos** del grupo (un cambio que cumpla con los requerimiento del **DeadBand**, si fuera aplicable) o algún cambio en la *calidad* de los valores de los items.

OPC *Automation.* *Notificación*

¿Cómo implementar la Notificación?

Ej: Se quiere leer los valores de dos *tags* de nombre *var1*, *var2*. Se quiere además que estas variables conformen un grupo.

Primero se configura el proceso de notificación

```
Dim WithEvents MiGrupo as OPCGroup

Dim ClientHandles(2) as Long

...
MiGrupo.OPCItems.Add(“var1”,1)
MiGrupo.OPCItems.Add(“var2”,2)

MiGrupo.IsSubscribed=True

MiGrupo.UpdateRate = 1000

....
```

OPC *Automation.* *Notificación*

¿Cómo implementar la Notificación?

Ej: Se quiere leer los valores de dos *tags* de nombre *var1*, *var2*. Se quiere además que estas variables conformen un grupo.

Segundo se programa el evento DataChange correspondiente

```
Private Sub MiGrupo_DataChange(TransactionID As Long, NumItems As Long,
ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long,
TimeStamps() As Date)
{
        MsgBox  “Valor de”  & ClientHandle(1) & ItemValues(1), vbOKOnly
        MsgBox  “Valor de”  & ClientHandle(2) & ItemValues(2), vbOKOnly
}
```

OPC *Automation.*

Lectura/Escritura Asíncronas.

OPC *Automation.* *OPCGroup*. Métodos

AsyncRead: Realiza una lectura asíncrona de los *items* indicados. La lectura siempre es desde el dispositivo físico. El método retorna inmediatamente. Cuando el servidor termina de realizar la lectura, los resultados son comunicados al cliente a través del evento **AsyncReadComplete.**

AsyncRead(NumItems As Long, ServerHandles() As Long, ByRef Errors() As Long, TransactionID As Long, ByRef CancelID As Long)

TransactionID: Identificador de la transacción escogido por el cliente. Nos permite, cuando arribe el evento **AsyncReadComplete**, identificar a que operación de lectura se refiere.

CancelID: Un identificador útil (devuelto por el servidor) en el caso que el cliente decida cancelar una operación de lectura en curso.

OPC *Automation.* *OPCGroup*. **Eventos**

AsyncReadComplete: Este evento informa de la conclusión de la lectura asíncrona identificada por *TransactionID*. Ofrece los valores leídos

AsyncReadComplete (TransactionID As Long, NumItems As Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date, Errors() As Long)

OPC *Automation.* *OPCGroup*. Métodos

AsyncWrite: Realiza una escritura asíncrona de los *items* indicados. La escritura siempre es desde el dispositivo físico. El método retorna inmediatamente. Cuando el servidor termina de realizar la escritura, informa al cliente a través del evento **AsyncWriteComplete.**

AsyncWrite(NumItems As Long, ServerHandles() As Long, Values() As Variant, ByRef Errors() As Long, TransactionID As Long, ByRef CancelID As Long)

OPC *Automation.* *OPCGroup*. **Eventos**

AsyncWriteComplete: Este evento informa de la conclusión de la escritura asíncrona identificada por T**ransactionID**.

AsyncWriteComplete (TransactionID As Long, NumItems As Long, ClientHandles() As Long, Errors() As Long)

OPC *Automation.* *OPCGroup*. **Métodos**

AsyncRefresh: Realiza una refrescamiento (de manera asíncrona) de los *items* indicados. En realidad se genera un evento **Data_Change** que incluye a todos los *items* activos (hayan cambiado de valor o no). Los datos son devueltos a través del evento **DataChange.**

AsyncRefresh(Source As Integer, TransactionID As Long,ByRef CancelID As Long)

Observar que en **Refresh** se puede especificar si se desea un refrescamiento de los datos desde el dispositivo físico o desde la *Cache*.

OPC *Automation.* *Notificación*

DataChange (TransactionID As Long, NumItems As Long, ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)

Si **TransactionID** es diferente de cero entonces **DataChange** ha ocurrido como respuesta a una llamada previa a **AsyncRefresh**. Es evidente que el **TransactionID** escogido por el cliente ha de ser distinto de cero.

OPC *Automation.* *OPCGroup*. **Métodos**

AsyncCancel: Cancela una operación de escritura o lectura asíncrona en curso.

AsyncCancel(CancelID As Long)

CancelID: es el identificador que el servidor nos ha dado cuando ordenamos la operación asíncrona.

OPC *Automation.* *OPCGroup*. **Eventos**

AsyncCancelComplete: Este evento informa de la conclusión de la operación de cancelación de la operación asíncrona (lectura o escritura) identificada por *TransactionID*.

AsyncCancelComplete (TransactionID As Long)

Creación de servidores OPC

La creación de servidores OPC es una tarea mucha más compleja que la creación de clientes.
Suele programarse en C/C++, idóneos para la implementación mediante mecanismos de herencia, trabajo con punteros, creación de memoria dinámico y otros elementos de bajo nivel.
En todo caso, la creación desde cero de servidores en una tarea de muy difícil consecución: la alternativa realista es la de utilizar frameworks especializados que faciliten la misma (Ej: Softing).
Si el objetivo es tener un servidor OPC simulado para ser utilizado en la puesta a punto de clientes OPC, existes herramientas de libres distribución, brindadas por ejmplo por Softing, Matrikon, entre otras.
Una aplicación interesante es la de dotar simuladores dinámicos de procesos industriales y dotarles del envoltorio de software necesario para acceder a las variables mencionadas mediante el estándar OPC. EcosimPro 5.2 en un lenguaje de Modelado y Simulación Orientado a Objetos que permite hacer lo anterior de manera relativamente sencilla.

Creación de clientes OPC

La creación de cliente OPC es mucho más sencilla:

- Utilizando Automation, se puede hacer aplicaciones con relativa facilidad desde Visual Basic o desde VBA en aplicaciones como Excel.
- Existen clientes genéricos gratuitos brindados por Softing, Matrikon, entre otros.
- Matlab ofrece facilidades para acceder como cliente a servidores OPC. Este acceso puede ser realizado mediante programación o desde Simulink.

Bibliografía

- J. M. Zamarreño, R. Mazaeda, J. A. Caminero, A. J. Rivero, and J. C. Arroyo, “A new plug-in for the creation of OPC servers based on EcosimPro© simulation software,” *Simul. Model. Pract. Theory*, vol. 40, pp. 86–94, 2014.
- X. Hong and W. Jianhua, “Using standard components in automation industry: A study on OPC specification,” *Comput. Stand. Interfaces*, vol. 28, no. 4, pp. 386–395, 2006.
- R. Alves Santos, J. E. Normey-Rico, A. Merino Gómez, L. F. Acebes Arconada, and C. de Prada Moraga, “OPC based distributed real time simulation of complex continuous processes,” *Simul. Model. Pract. Theory*, vol. 13, no. 7, pp. 525–549, 2005.
- Iwanitz, F., Lange, J., 2002. OPC : fundamentals, implementation and application.
- Zamarre˜no, J., 2010. Acceso a datos mediante OPC. Editorial Andavira.