Projet communautaire OpenOpcUa- Les VPIs

Virtual Protocol Interface

Add-In - pour la communication

Michel Condemine

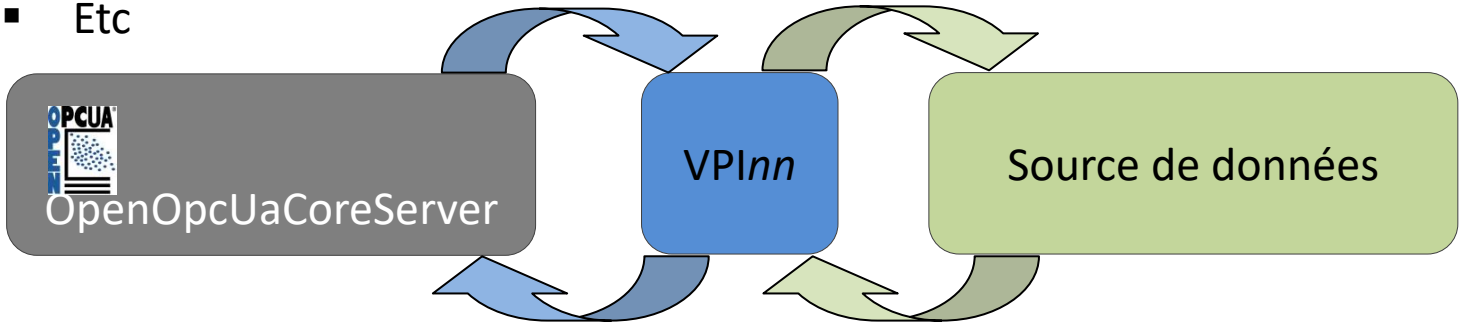
- OPC Foundation France (Technique)
- Directeur de **4CE** Industry
- Leader du projet OpenOpcUa
- MichelC@4CE-Industry.com

Le serveur comme passerelle de communication

A quoi sert un Vpi ?

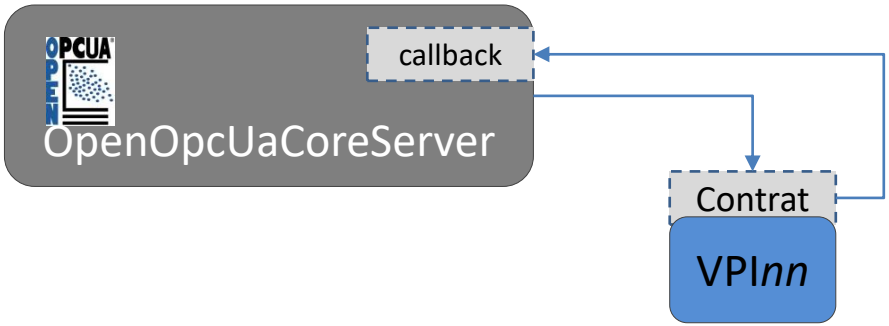
Vous utiliserez un VPI pour échanger de l'information entre l'OpenOpcUaCoreServer et une source donnée externe. Cette source de donnée peut être :

- Un équipement communiquant, PLC, RTU exposant des données au travers d'un
  - socket TCP
  - port série
  - Etc.
- Une application externe qui expose
  - Une interface de programmation (API)
  - Une zone de mémoire partagé
  - Etc

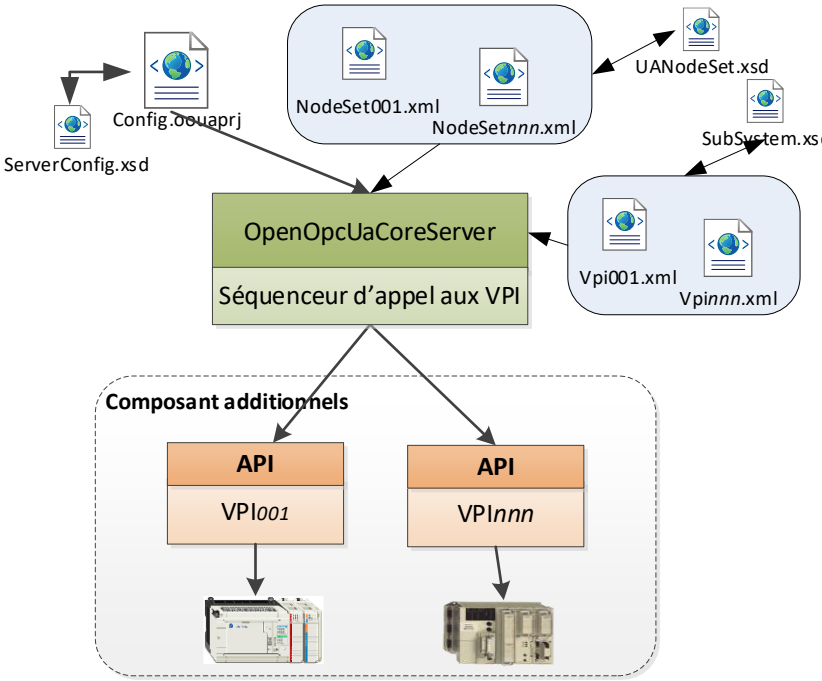


Le serveur comme passerelle de communication

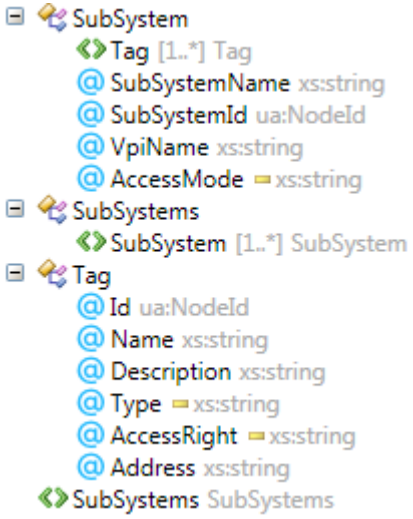
- Les VPI sont des extensions (Add-In) pour le serveur OpenOpcUa. Ils permettent d'implémenter des protocoles de communications.
- Les VPI peuvent être implémentés sous Windows, Linux, MacOs.
- Les VPI permettent d'implémenter une communication de type :
  - Client/Serveur
  - Peer to Peer (Non-sollicité)
  - Hybride
- Un contrat d'interface, qu'implémente tous les VPIs, est défini entre le serveur OpenOpcUa et les VPIs.



Le serveur comme passerelle de communication

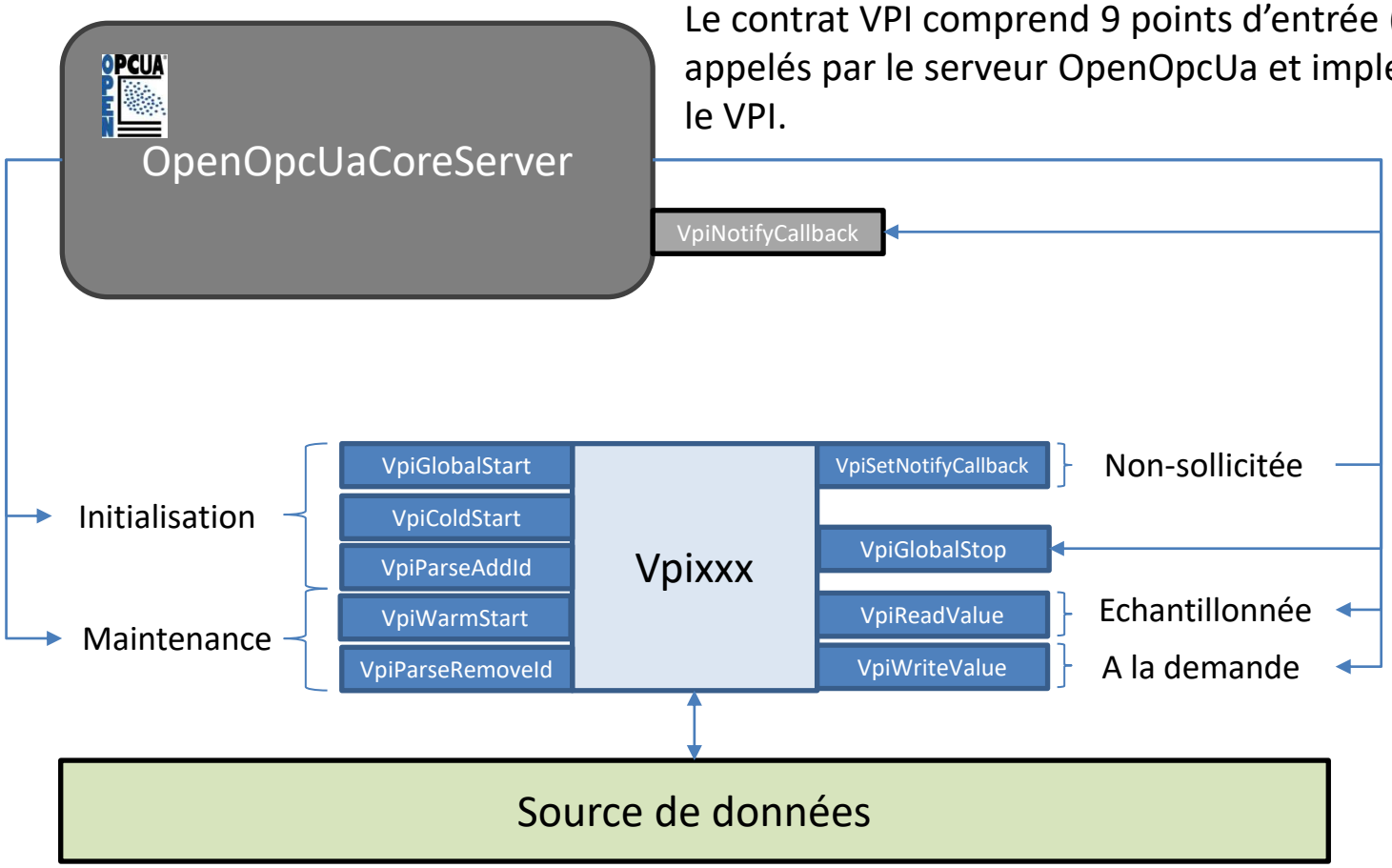


- Le serveur peut prendre en charge une infinité de VPI en même temps.
- Les VPI sont conçus pour être écrits en en C ou en C++ mais cela n'est pas obligatoire.
- Les VPI sont configurés dans un fichier XML conforme au schéma XML *Subsystem.xsd*.

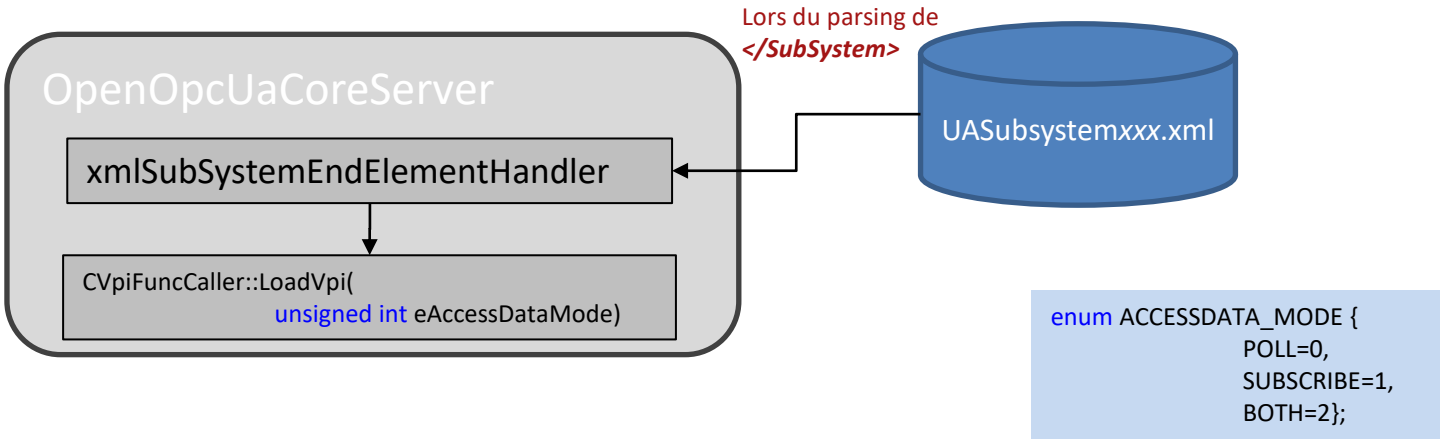


Le serveur comme passerelle de communication

Le contrat VPI comprend 9 points d'entrée (API). Ils sont appelés par le serveur OpenOpcUa et implémentés par le VPI.



Le Vpi vue du serveur
Initialisation du VPI



OpcUa_StatusCode CVpiFuncCaller::LoadVpi(unsigned int eAccessDataMode)

First we load the VPI based on the name read in the configuration file

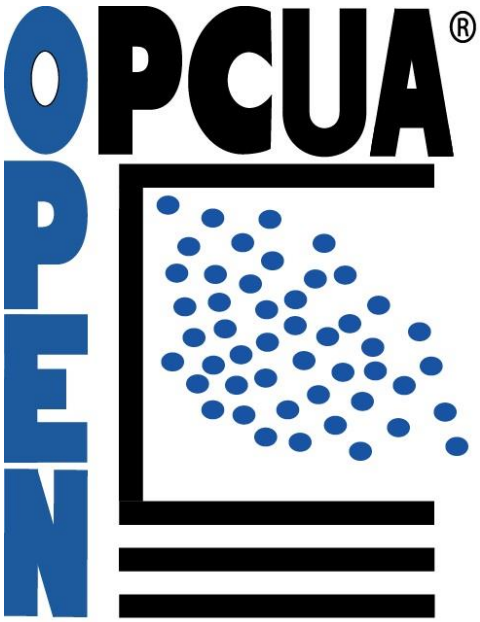
```
uStatus=OpcUa_LoadLibrary(m_pLibraryName,(void**)&m_phInst);
```

If it succeed with load the entry point

```
GlobalStop = (PFUNCGLOBALSTOP)OpcUa_GetProcAddress(m_phInst,"VpiGlobalStop");
GlobalStart = (PFUNCGLOBALSTART)OpcUa_GetProcAddress(m_phInst,"VpiGlobalStart");
ColdStart = (PFUNCCOLDSTART)OpcUa_GetProcAddress(m_phInst,"VpiColdStart");
WarmStart = (PFUNCWARMSTART)OpcUa_GetProcAddress(m_phInst,"VpiWarmStart");
ReadValue = (PFUNCREADVALUE)OpcUa_GetProcAddress(m_phInst,"VpiReadValue");
WriteValue = (PFUNCWRITEVALUE)OpcUa_GetProcAddress(m_phInst,"VpiWriteValue");
ParseAddId = (PFUNCPARSEADDID)OpcUa_GetProcAddress(m_phInst,"VpiParseAddId");
ParseRemoveId = (PFUNCPARSEREMOVEID)OpcUa_GetProcAddress(m_phInst,"VpiParseRemoveId");
SetNotifyCallback= (PFUNCSETNOTFIFYCALLBACK)OpcUa_GetProcAddress(m_phInst,"VpiSetNotifyCallback");
```

Les Vpis

La signature des fonctions exposées par un Vpi



Les fonctions d'initialisation

Vpi_StatusCode **VpiGlobalStart**(Vpi_String szSubSystemName,
Vpi_NodeId SubsystemId,
Vpi_String szProjectFolder,
Vpi_String szLogFolder,
Vpi_Handle* hVpi)

Vpi_StatusCode **VpiGlobalStop**(Vpi_Handle hVpi)

Vpi_StatusCode **VpiColdStart**(Vpi_Handle hVpi)

Vpi_StatusCode **VpiWarmStart**(Vpi_Handle hVpi)

Vpi_StatusCode **VpiSetNotifyCallback**(Vpi_Handle hVpi,
PFUNCNOTIFYCALLBACK lpCallbackNotify)

```
typedef Vpi_StatusCode(__stdcall *PFUNCNOTIFYCALLBACK)
        (Vpi_UInt32 uiNoOfNotifiedObject, Vpi_NodeId* Id, Vpi_DataValue* pValue);
```

Les fonctions de lecture/écriture

OpcUa_Vpi_StatusCode **VpiReadValue**(Vpi_Handle hVpi,
Vpi_UInt32 uiNbOfValueRead,
Vpi_NodeId* Ids,
Vpi_DataValue** ppValue)

OpcUa_Vpi_StatusCode **VpiWriteValue**(Vpi_Handle hVpi,
Vpi_UInt32 UiNbOfValueWrite,
Vpi_NodeId* Ids,
Vpi_DataValue** ppValue)

```
typedef struct _Vpi_NodeId
{
	Vpi_UInt16 IdentifierType;
	Vpi_UInt16 NamespaceIndex;
	union
	{
		Vpi_UInt32	Numeric;
		Vpi_String	String;
		Vpi_Guid*	Guid;
		Vpi_ByteString ByteString;
	}  Identifier;
} Vpi_NodeId;
```

```
typedef struct _Vpi_DataValue
{
	Vpi_Variant	Value;
	Vpi_StatusCode StatusCode;
	Vpi_DateTime	SourceTimestamp;
	Vpi_DateTime	ServerTimestamp;
	Vpi_UInt16	SourcePicoseconds;
	Vpi_UInt16	ServerPicoseconds;
} Vpi_DataValue;
```

Vpi_NodeId_Initialize
Vpi_xxxxx_Initialize avec xxxx = String, DataValue, Semaphore, Mutex, Thread, etc.
Vpi_xxxxx_fffffffff avec ffffffff = Initialize, Clear, CopyTo, AttachCopy, Compare,

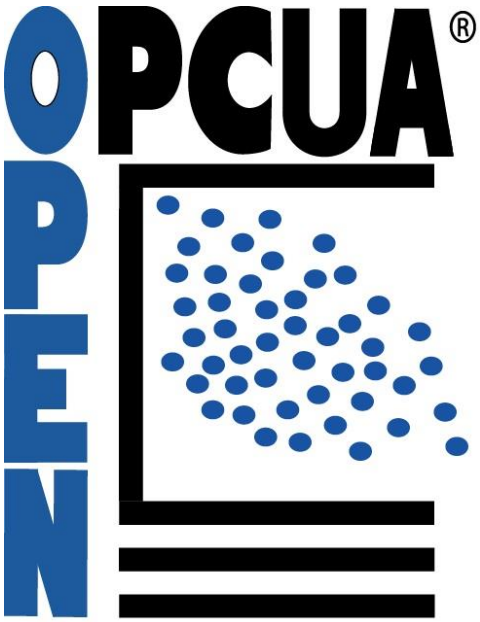
Les fonctions d'analyse syntaxique

Vpi_StatusCode **VpiParseAddId**(Vpi_Handle hVpi,
Vpi_NodeId Id, OpcUa_Byte Datatype,
Vpi _UInt32 iNbElt,
Vpi _Byte AccessRight,
Vpi _String ParsedAddress);

Vpi_StatusCode **VpiParseRemoveId**(Vpi_Handle hVpi,
Vpi _NodeId Id);

Les Vpis

Description de l'API



Description de l'API

Vpi_StatusCode VpiGlobalStart(Vpi_String szSubSystemName,
Vpi_NodeId SubsystemId,
Vpi_String szProjectFolder,
Vpi_String szLogFolder,
Vpi_Handle* hVpi)

Cette fonction est appelée une seule fois au chargement du VPI par le serveur OpenOpcUa.
C'est dans cette fonction que le VPI réalise sa séquence d'initialisation interne.

```
Vpi_StatusCode VpiGlobalStart(Vpi_String szSubSystemName, Vpi_NodeId SubsystemId, Vpi_String szProjectFolder, Vpi_String szLogFolder, Vpi_Handle* hVpi)
{
    (void)szSubSystemName;
    (void)SubsystemId;
    Vpi_StatusCode uStatus;
    CVpiInternalData* phVpiInternalData=NULL;
    phVpiInternalData=new CVpiInternalData();
    if (phVpiInternalData)
    {
        // Save SubsystemName andSubSystemId
        phVpiInternalData->SetSubSystemName(szSubSystemName);
        phVpiInternalData->SetSubsystemId(SubsystemId);
        // Load extra configuration parameter if needed
        phVpiInternalData->LoadConfigurationFile();
        // SetVpiData to the internalData instance
        phVpiInternalData->SetVpiHandle((Vpi_Handle)phVpiInternalData);
        gVpiInternalData.push_back(phVpiInternalData);
        *hVpi = (Vpi_Handle)phVpiInternalData;
        uStatus = Vpi_Good;
    }
    else
        uStatus=Vpi_BadInternalError;
    return uStatus;
}
```

Description de l'API

OpcUa_Vpi_StatusCode **VpiGlobalStop**(OpcUa_Vpi_Handle hVpi)
Cette fonction est appelée une seule fois au moment de l'arrêt du serveur OpenOpcUa. C'est dans cette fonction que le VPI la libération de ses ressources.

Exemple

```
Vpi_StatusCode VpiGlobalStop(OpcUa_Handle* hVpi)
{
        OpcUa_Vpi_StatusCode uStatus=OpcUa_Vpi_Good;
        if (!hVpi)
                uStatus=OpcUa_Vpi_Bad;
        else
        {
                CVpiInternalData* pVpiInternal=(CVpiInternalData*)hVpi;
                // Do something
        }
        return uStatus;}
```

Description de l'API

OpcUa_Vpi_StatusCode **VpiColdStart**(Vpi_Handle hVpi)

Cette fonction est utilisée pour réaliser les traitements de configuration qui ne seront réalisés qu'une seule fois. Le serveur OpenOpcUa appelle cette fonction lors de l'activation de la relation entre le serveur et l'équipement sous-jacent.

Exemple

```
Vpi_StatusCode VpiColdStart(Vpi_Handle hVpi)
{
        OpcUa_Vpi_StatusCode uStatus=OpcUa_Vpi_Good;
        if (!hVpi)
                uStatus=OpcUa_Vpi_Bad;
        else
        {
                CVpiInternalData* pVpi=(CVpiInternalData*)hVpi;
                … something to do
        }
        return uStatus;
}
```

Description de l'API

OpcUa_Vpi_StatusCode **VpiWarmStart**(Vpi_Handle hVpi)
Cette fonction est utilisée pour réaliser les traitements de configuration qui seront réalisés chaque fois que la relation entre le Vpi et l'équipement sous-jacent sera interrompue. Le Vpi informe le serveur que la fonction **VpiWarmStart** doit être appelée en lui transmettant le code d'erreur **Vpi_WarmStartNeed**.

Exemple

```
Vpi_StatusCode VpiWarmStart(Vpi_Handle hVpi)
{
        OpcUa_Vpi_StatusCode uStatus=OpcUa_Vpi_Good;
        if (!hVpi)
                uStatus=OpcUa_Vpi_Bad;
        else
        {
                CVpiInternalData* pVpi=(CVpiInternalData*)hVpi;
                … something to do
        }
        return uStatus;
}
```

Description de l'API

```
Vpi_StatusCode VpiParseAddId(
                Vpi_Handle hVpi,
                Vpi _NodeId Id,
                Vpi _Byte Datatype,
                Vpi _UInt32 iNbElt,
                Vpi _Byte AccessRight,
                Vpi _String ParsedAddress )
```

Cette fonction est appelée par le serveur pour chaque balise **<Tag>** du fichier de configuration du sous-système. Tous les paramètres de la fonction sont des paramètres d'entrée. Le Vpi devra vérifier que la **ParsedAddress** est syntaxiquement cohérente pour ce Vpi et stocker ces informations afin de répondre aux demandes d'écriture et/ou de lecture qui seront faites ultérieurement par le serveur.

Codes renvoyés :

- Vpi_Good si l'analyse syntaxique est correcte et que le Tag est accepté.
- Vpi_ParseError en cas d'erreur de d'analyse syntaxique

Description de l'API

Vpi_StatusCode **VpiReadValue**(
Vpi_Handle hVpi,
Vpi_UInt32 uiNbOfValueRead,
Vpi_NodeId* Ids,
Vpi_DataValue** pValue)

Cette fonction réalise la lecture de *uiNbOfValueRead* NodeId dans la source de données. Le résultat est renvoyé dans *uiNbOfValueRead OpcUa_DataValue*. Il est recommandé de lire les données à partir de la cache du Vpi.

Codes renvoyés :

- **Vpi_Good** si la lecture s'est bien passée
- **Vpi_Bad** en cas de réception d'un *OpcUa_Vpi_Handle* incorrect.
- **Vpi_BadNotFound** si le nodeId n'existe pas

Exemple

```
OpcUa_Vpi_StatusCode VpiReadValue(OpcUa_Vpi_Handle hVpi, OpcUa_UInt32 uiNbOfValueRead, OpcUa_NodeId* Ids, OpcUa_DataValue** pValue)
{
    OpcUa_Vpi_StatusCode uStatus = OpcUa_Vpi_Good;
    if (!hVpi)
        uStatus = OpcUa_Vpi_Bad;
    else
    {
        if (uiNbOfValueRead > 0)
        {
            for (OpcUa_Int32 i = 0; i < uiNbOfValueRead; i++)
            {
                CVpiInternalData* pVpi = (CVpiInternalData*)hVpi;
                CSourceObject* pObject = pVpi->GetSourceObject(Ids[i]);
                if (pObject)
                {
                    CVpiDataValue* apValue = pObject->GetValue();
                    apValue->SetStatusCode(OpcUa_Vpi_UncertainInitialValue);
                }
                else
                    uStatus = OpcUa_Vpi_BadNotFound;
            }
        }
    }
    return uStatus;
}
```

Description de l'API

Vpi_StatusCode **VpiWriteValue**(Vpi_Handle hVpi,
Vpi_UInt32 UiNbOfValueWrite,
Vpi_NodeId* Ids,
Vpi_DataValue** ppValue)

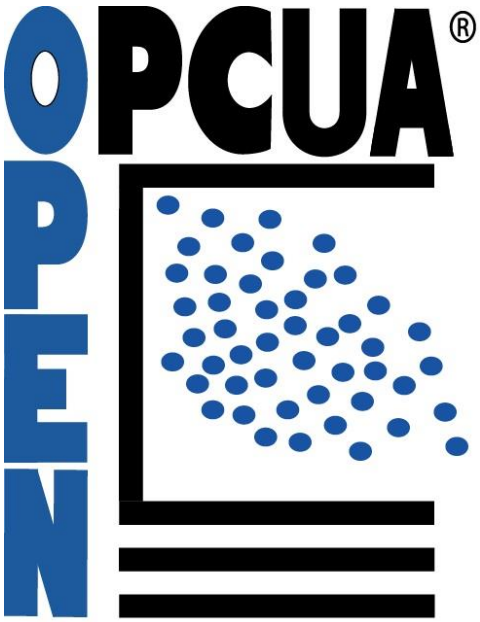
Cette fonction réalise l'écriture de *UiNbOfValueWrite OpcUa_DataValue* dans la source de données pour les *OpcUa_NodeIds passés en paramètre*. ***ppValue*** et ***Ids*** sont des pointeurs de ***UiNbOfValueWrite*** élément.

Codes renvoyés :

- Vpi_Good si l'écriture s'est bien passée
- Vpi_Bad en cas de réception d'un mauvais OpcUa_Vpi_Handle
- Vpi_BadNotFound si le nodeId n'existe pas

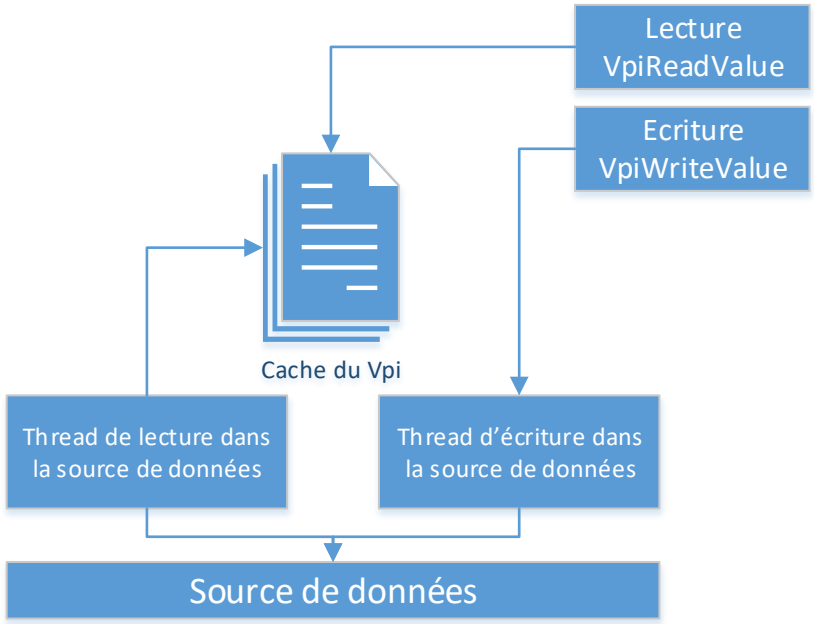
```
OpcUa_Vpi_StatusCode VpiWriteValue(OpcUa_Vpi_Handle hVpi,
                                   OpcUa_UInt32 UiNbOfValueWrite,
                                   OpcUa_NodeId* Ids,
                                   OpcUa_DataValue** ppValue)
{
    OpcUa_Vpi_StatusCode uStatus=OpcUa_Vpi_Good;
    if (!hVpi)
        uStatus=OpcUa_Vpi_Bad;
    else
    {
        CVpiInternalData* pVpi=(CVpiInternalData*)hVpi;
    }
    return uStatus;
}
```

Architecture et recommandations



Architecture

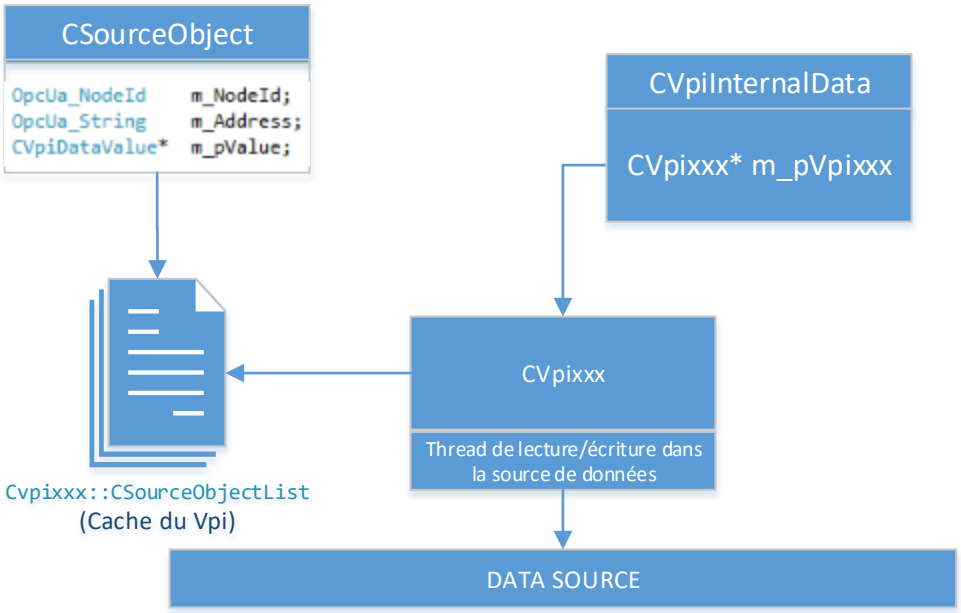
Le schéma ci-dessous présente l'architecture recommandé pour un Vpi. Un Vpi doit dans la mesure du possible et sauf exception s'articuler autour de deux threads. Ces thread seront en charge des interactions avec la source de données.



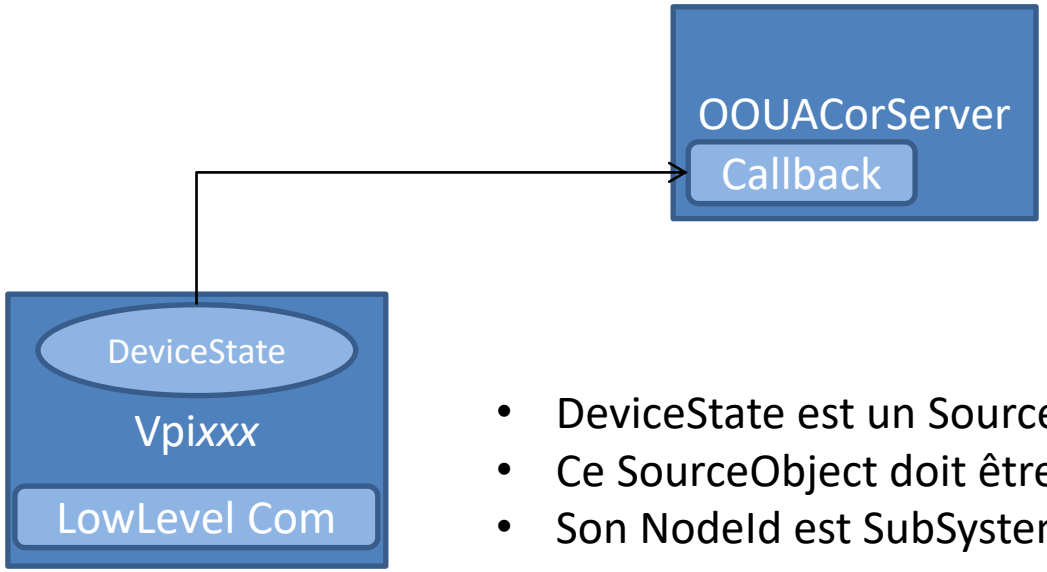
Architecture

Les classes et les namespaces qui sont utilisées dans les modèles de Vpis sont les suivantes :

Namespace	Description
UASubSystem	Ce namespace contient les classes chargées de gérer la cache du Vpi
UABuiltInType	Ce namespace contient les types natifs utilisé par le Vpi, OpcUa_Boolean, OpcUa_Double, OpcUa_Variant,etc.
Vpixxx.	Ce namespace contient la partie en charge du protocole est des interactions avec les équipements



Notification des erreurs

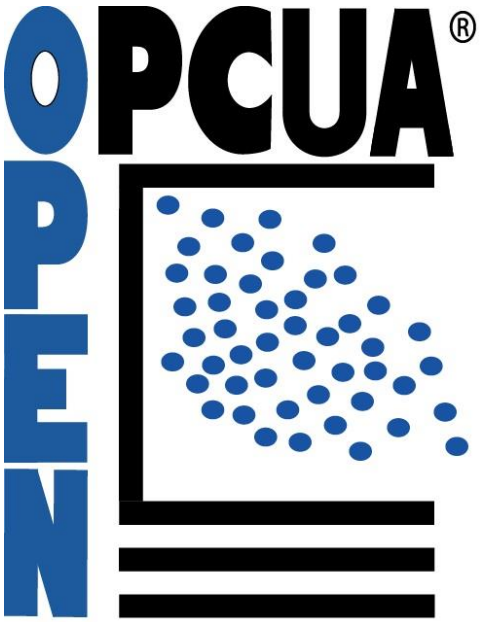


- DeviceState est un SourceObject de type StatusCode.
- Ce SourceObject doit être créé par le Vpi*xxx*
- Son NodeId est SubSystemId+4

Recommandations pour l'écriture d'un Vpi

1. Le Vpi doit gérer une cache interne
2. La fonction de lecture doit simplement accéder à cette cache
3. Un thread indépendant doit rafraichir cette cache interne à partir de la source de donnée.
4. La durée d'exécution de la fonction de lecture ne doit pas être supérieure à la vitesse d'abonnement ou d'échantillonnage des clients OPC UA (Sampling).
5. Utilisez les modèles existant pour écrire un nouveau Vpi
6. VpiRead, Vpiwrite et VpiCall ne doivent pas appeler la Callback.

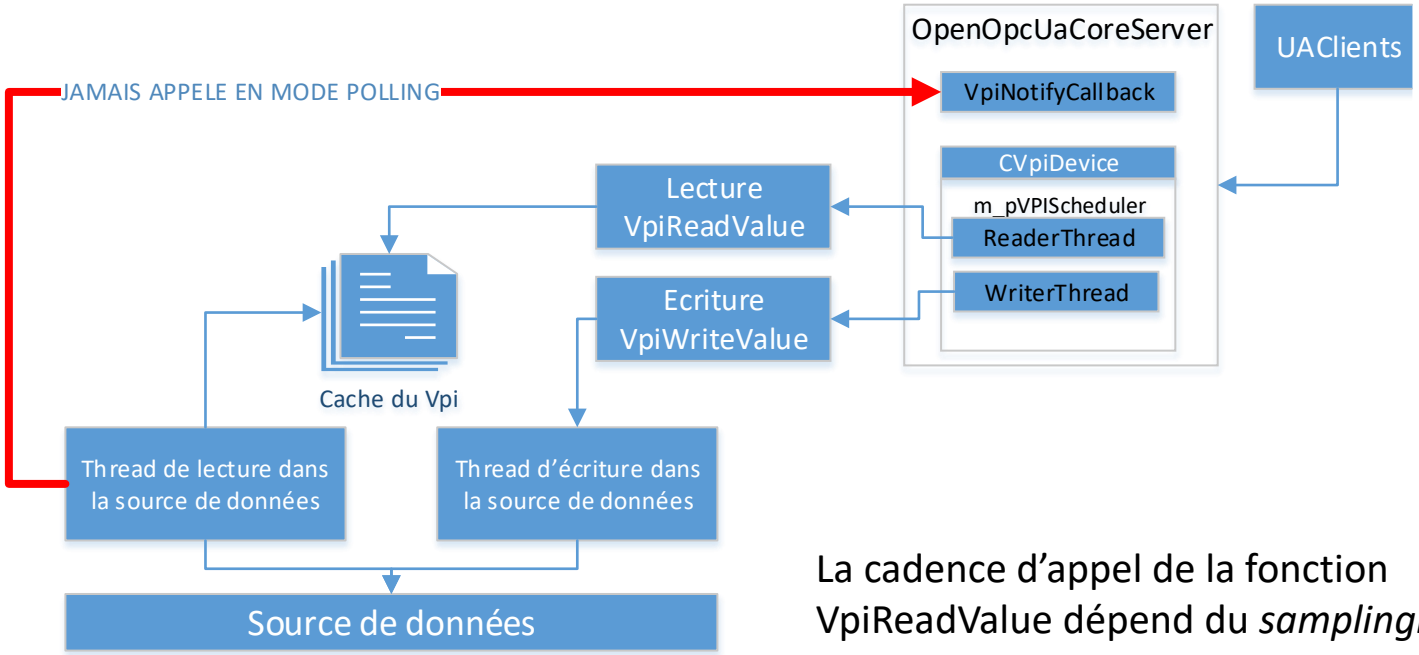
Bien choisir le mode de fonctionnement de son VPI



VPI en mode polling

Actif/autorisé

Inactif/Interdit

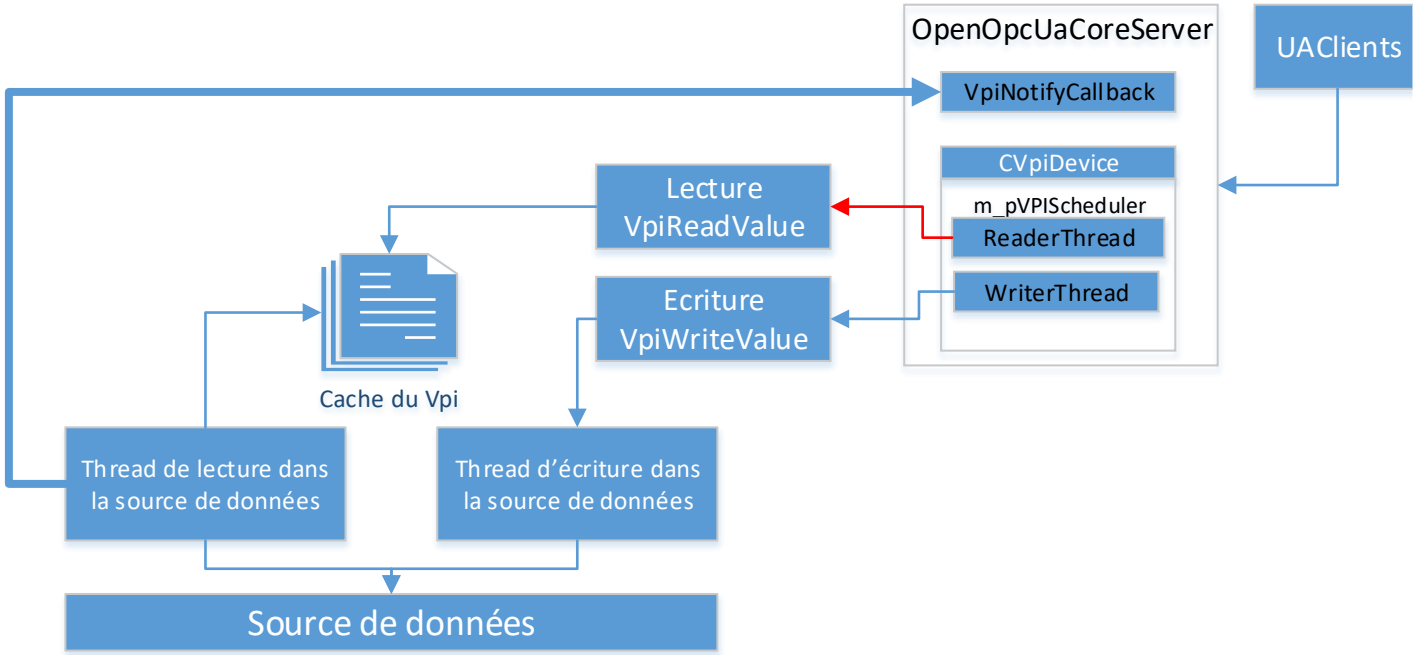


La cadence d'appel de la fonction VpiReadValue dépend du *samplinginterval* mis en place par le client UA.
Par défaut un valeur de 1 seconde est utilisée.

VPI en mode Unsollicited

Actif/autorisé

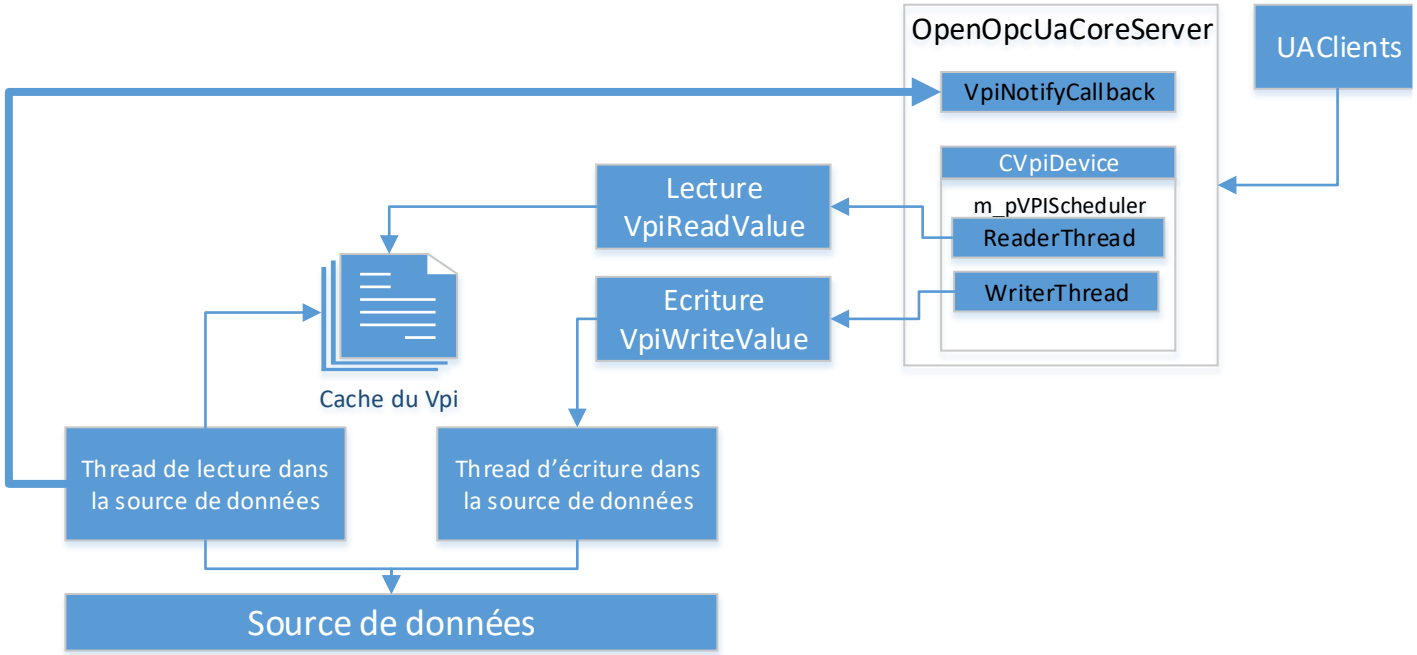
Inactif/Interdit



VPI en mode hybride

Actif/autorisé

Inactif/Interdit



Questions ?

Michel Condemine

- OPC Foundation France (Technique)
- Directeur de **4CE** Industry
- Leader du projet OpenOpcUa
- MichelC@4CE-Industry.com