Patni - Presentation

"SMI-S Provider – Architecture Explored"

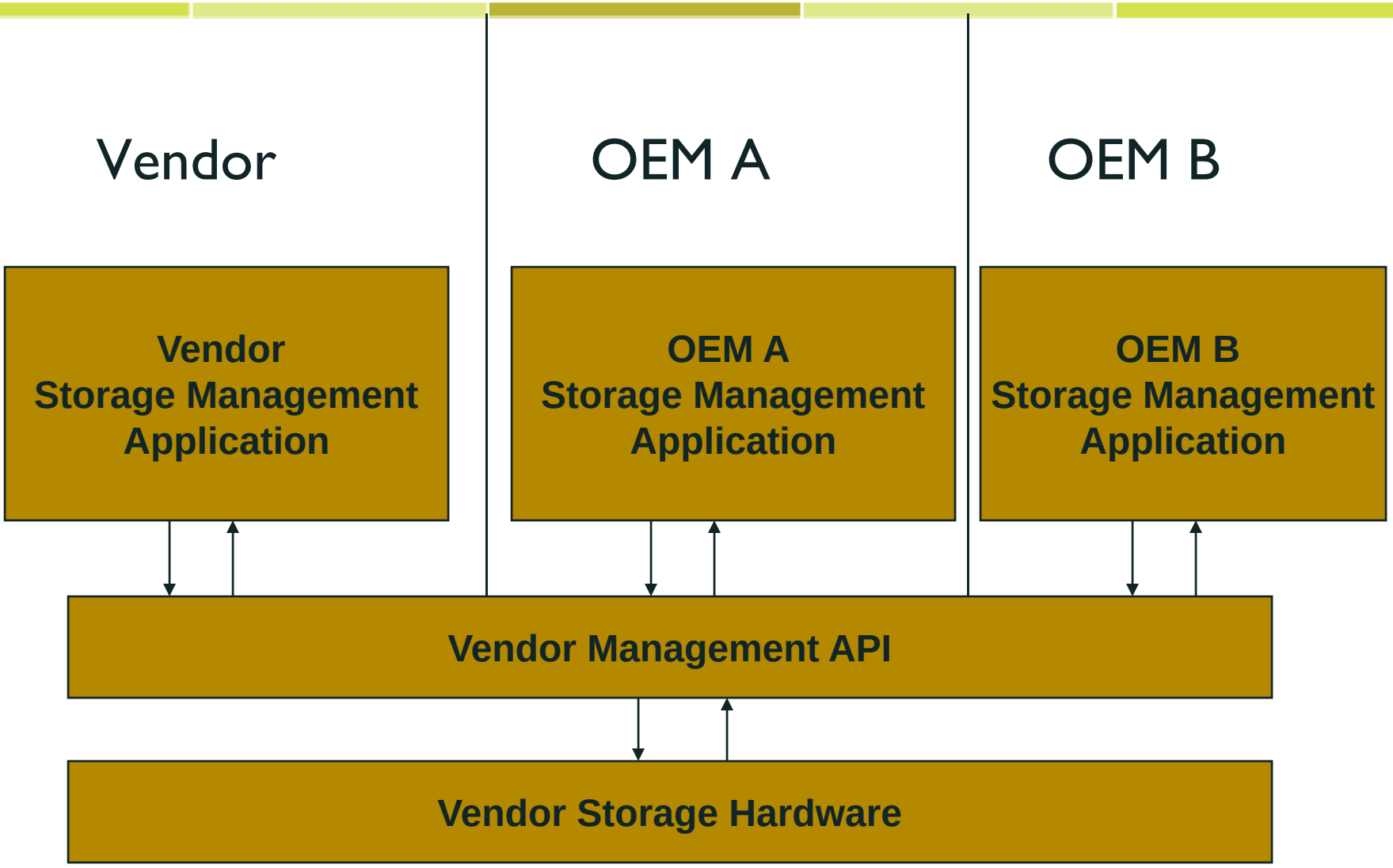
Prepared By: Ashish Jangam, Neerav

Parikh

Ability to scope projects is an essential element of the engineering discipline

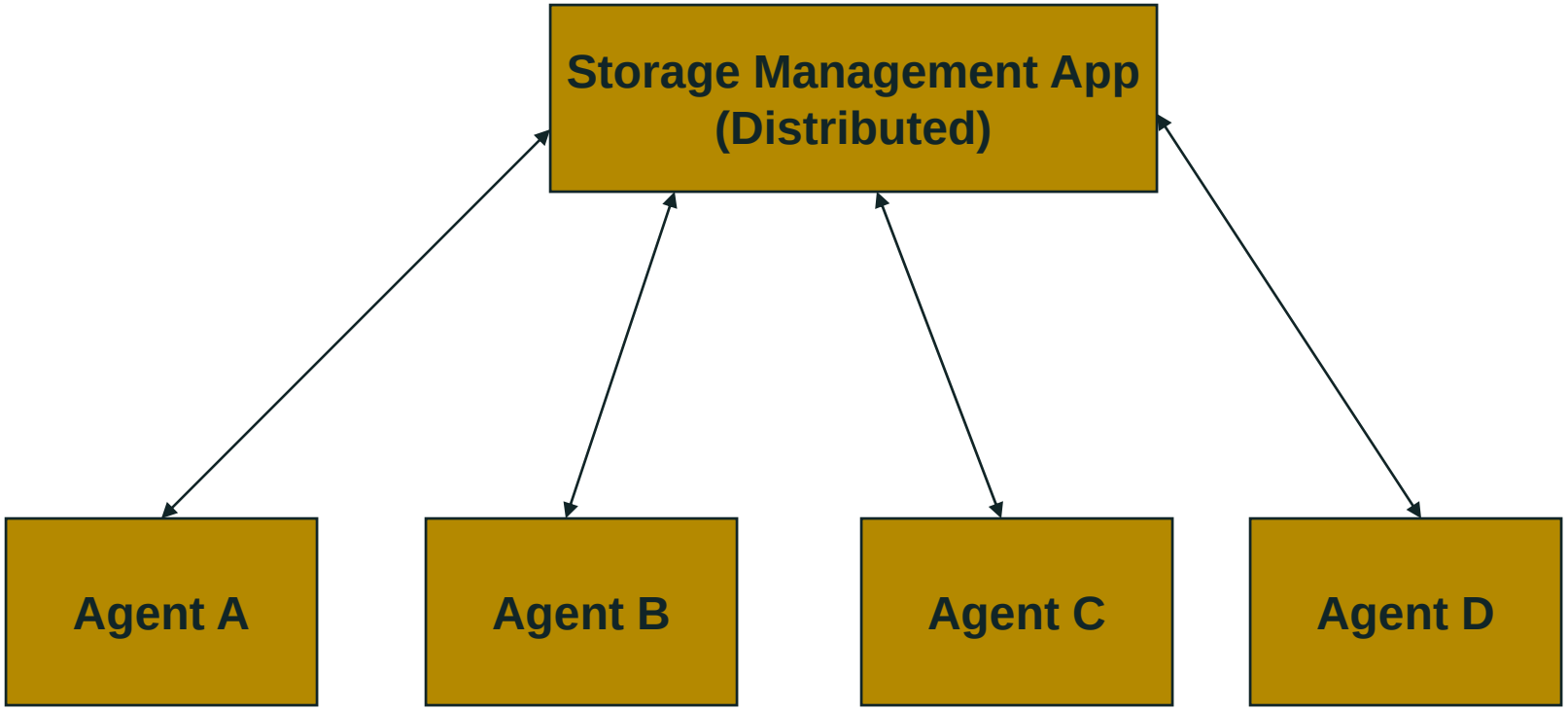
Contents

Evolution of SMI Providers



Evolution of SMI Providers

PROPRIETARY



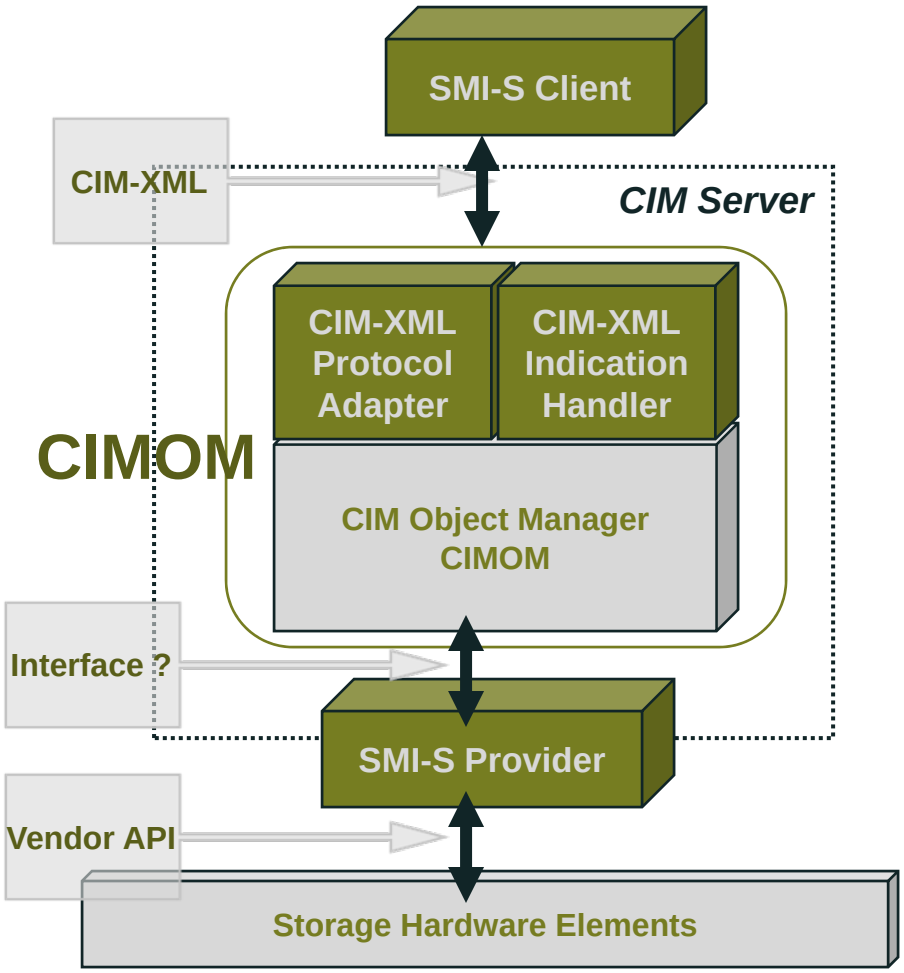
Evolution of SMI Providers

- Storage Management Gurus got together under groups like SNIA, DMTF, etc. to formalize industry level standards to manage storage.
- SMI-S (Storage Management Initiative – Spec) under SNIA
- Define Storage Related Information Models or Profiles based on CIM/WBEM standards from DMTF

SOW Update

- SMI-S Providers required to manage your hardware device(s) with our SMI-S Management Application (OEM)

WBEM Providers



SMI-S Client
- *Interacts with a CIM Server by issuing CIM Operation Message Requests and receiving/processing CIM Operation Message Responses*

CIM Server
- *Server that receives and processes CIM Operation Message Requests and issues CIM Operation Message Responses*

CIM Object Manager (CIMOM)
- *Central component of the CIM Server responsible for the communication between other components*

SMI-S Provider
- *Instruments one or more aspects of the CIM Schema, reflecting the "real world"*

WBEM Providers

- WBEM Implementations : OpenPegasus, OpenWBEM, WBEM Services, etc.
- SMI-S Provider Interface with CIMOM
  - Native (CIMOM/Implementation specific)
    - C/C++/JAVA Interface API
    - Native Types
  - CMPI (Common Management Programming Interface)
    - C based API (standard interface for all supporting this standard)
    - CMPI Types

CMPI Providers

Final Result
Number of Devices

SMI-S Provider

CIM Client

CIM/XML CIM/XML

CIM Server

Checks for Registered Provider.

Repository

CMPI Interface

SMI-S Profile Implementation

Instrumentation Adapter

Devices

STORAGE

CMPI Providers

- CMPI Data Types
- CMPI Interfaces
  - Instance (enumerate, create, query)
  - Association (relation)
  - Indication (alerts, hardware status)
  - Method (configure)

CMPI Provider – EnumInstance Names Sample Code


```
extern "C" CMPIStatus StorageVolumeProviderEnumInstanceNames(
                                                        CMPIInstanceMI *mi,
                                                        const CMPIContext *ctx,
                                                        const CMPIResult *rslt,
                                                        const CMPIObjectPath * ref)
{
   //Initializations

    CMPIStatus rc = { CMPI_RC_OK, NULL };
    CMPIObjectPath *cmpi_op = NULL;
    char *nameSpace = (char *)CMGetCharsPtr(CMGetNameSpace(ref, &rc), &rc);
    VolumeList = GetVolumeList();
    for(VolumeList)
    {
                // Create new object Path and add key properties to it
                cmpi_op = CMNewObjectPath( _broker, nameSpace, "PATNI_StorageVolume“, &rc );
                SystemCreationClassName = getSysCreationalClassName();
                CMAddKey(op,"SystemCreationClassName", (CMPIValue*)SystemCreationClassName.c_str(),CMPI_chars);

                // Return object path asynchronously to the CIMOM
                CMReturnObjectPath( rslt, cmpi_op )
      } // End of for Loop
// Indicate to the CIMOM that we are done.
CMReturnDone( rslt );
// Send the status.
return rc;
}
```

SOW Update

- SMI-S Provider for Windows WMI

WMI Providers

- WMI Provider is a COM Object (Windows only)
- Windows and WMI specific Types
- WMI Native Interfaces
- No support for CMPI ☹

WMI Provider – CreateInstanceEnumAsync Sample Code


```
HRESULT CStdProvider::CreateInstanceEnumAsync(
        /* [in] */ BSTR strClass,
        /* [in] */ long lFlags,
        /* [in] */ IWbemContext __RPC_FAR *pCtx,
        /* [in] */ IWbemObjectSink __RPC_FAR *pResponseHandler
        )
{
   // Initializations
   IWbemClassObject *pClass = 0;
   IWbemClassObject *pNewInst = 0;
   HRESULT hRes = m_pSvc->GetObject(strClass, 0, NULL, &pClass, 0);
   VolumeList = GetVolumeList();
   VARIANT value;

        VariantInit(&value);
        for(VolumeList)
   {
            //Create Object Path by adding key properties to the object path
            pClass->SpawnInstance(0, &pNewInst);
            SystemCreationClassName = getSysCreationalClassName();
            value->bstrVal =          StringToBSTR(SystemCreationClassName.c_str());
            pNewInst->Put(L"SystemCreationClassName",0,&value,0);
            // Deliver the class to WMI.
            pResponseHandler->Indicate(1, &pNewInst);
            pNewInst->Release();

  } // End of for Loop
  pClass->Release();
  pResponseHandler->SetStatus(0, WBEM_S_NO_ERROR, 0, 0);
  return WBEM_S_NO_ERROR;
}
```

Conventional Approach – Pros

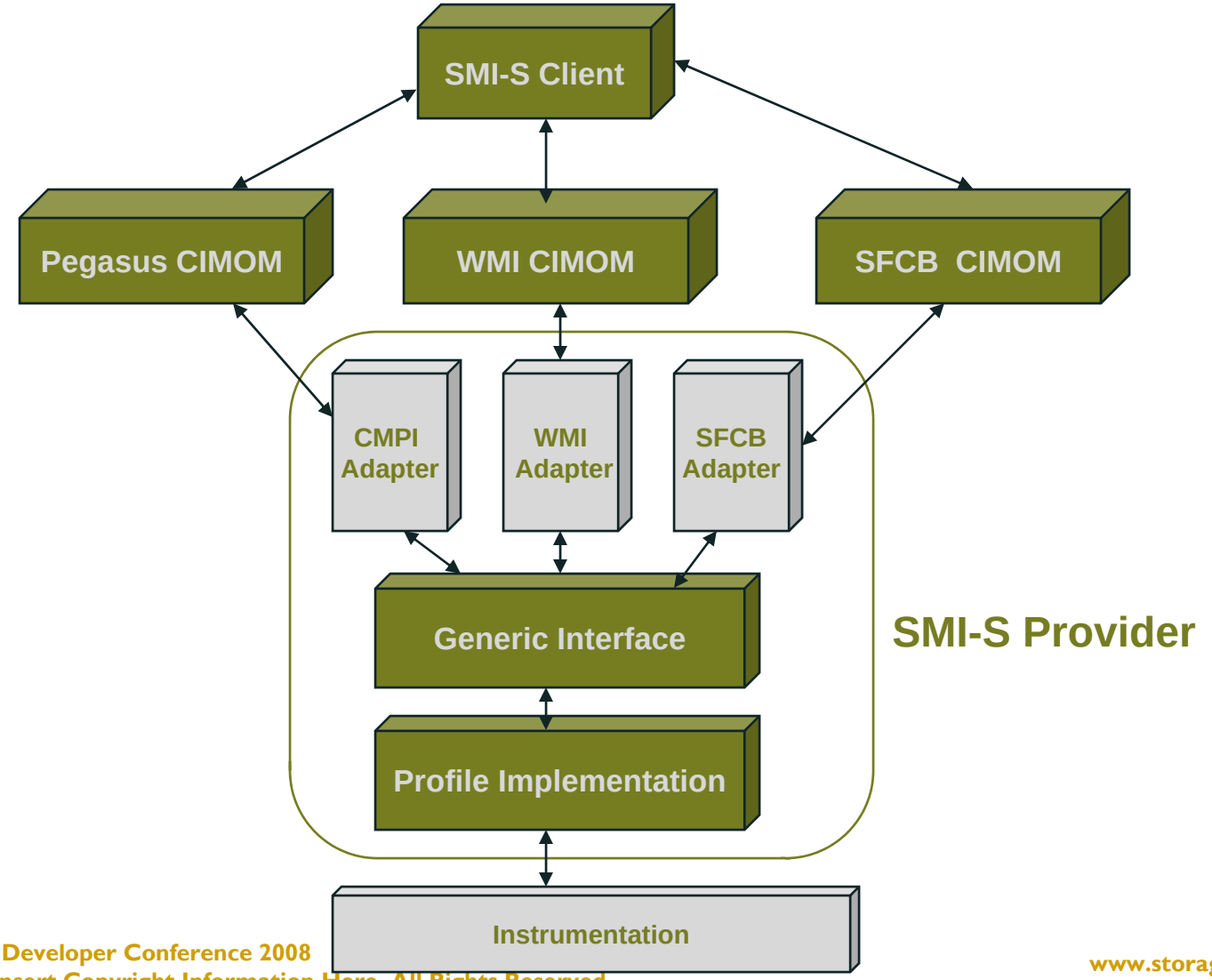
- If the SMI-S Provider needs to be developed only for a specific CIMOM (e.g. embedded Providers)
- Performance of using natively supported interfaces of a CIMOM is better

Conventional Approach – Cons

- Not all Management Broker (CIMOM) supports CMPI/WMI/Native interface.
- Change in Interface spec may require changes in the Provider code.
- Every developer must be aware of syntax and semantic of interface. For e.g developer should be well versed with Management Instrumentation (MI)API's . MI can be CMPI, WMI or native. In case of WMI, developer should have working knowledge of COM/DCOM and ATL. ATL/COM programming has its own challenges that may distract focus on implementing providers.
- Support of multiple CIMOM interface will require considerable efforts, resources and also provider code will not get reused extensively. For e.g., CMPI provider code will not get reused much in WMI provider.
- Decrease in code reusability across multiple CIMOM interfaces will proportionally increase the development.
- Maintaining different code bases for Provider if need to support OpenPegasus on Linux and WMI on Windows.
- Testing efforts will also increase if Provider has to support multiple CIMOM interfaces.

What's the solution?

Interface Adapter Architecture



Interface Adapter Architecture

- *Localize the Management Instrumentation (CMPI/WMI/Native) primitive data types and classes at the interface adapter layer.*
- *Profile implementer will work on generalized data types and classes.*

Profile implementer will collect the desired properties and their values and send it to the Provider Adapter framework, which will then deliver it to CIMOM.

Interface Adapter Architecture – Pros

- *Improved "time to market" factor.*
- *Profile code can be reused.*
- *Performance will be nearly same for all interfaces.*
- *Efforts required for adding support for new CIMOM interface will be restricted to the adapter layer only. Test efforts are also reduced since there is no need to test different providers only the new adapter layer needs to be tested.*
- *Availability of Provider development tools like CIMPLE/SimpleWBEM*

Interface Adapter Architecture – Cons

- *Design efforts are high as compared to the conventional approach.*
- *Profile implementer does not have access to Management Brokers (CIMOM) API.*
- *Existing CMPI/Native based providers that are already developed cannot be reused.*

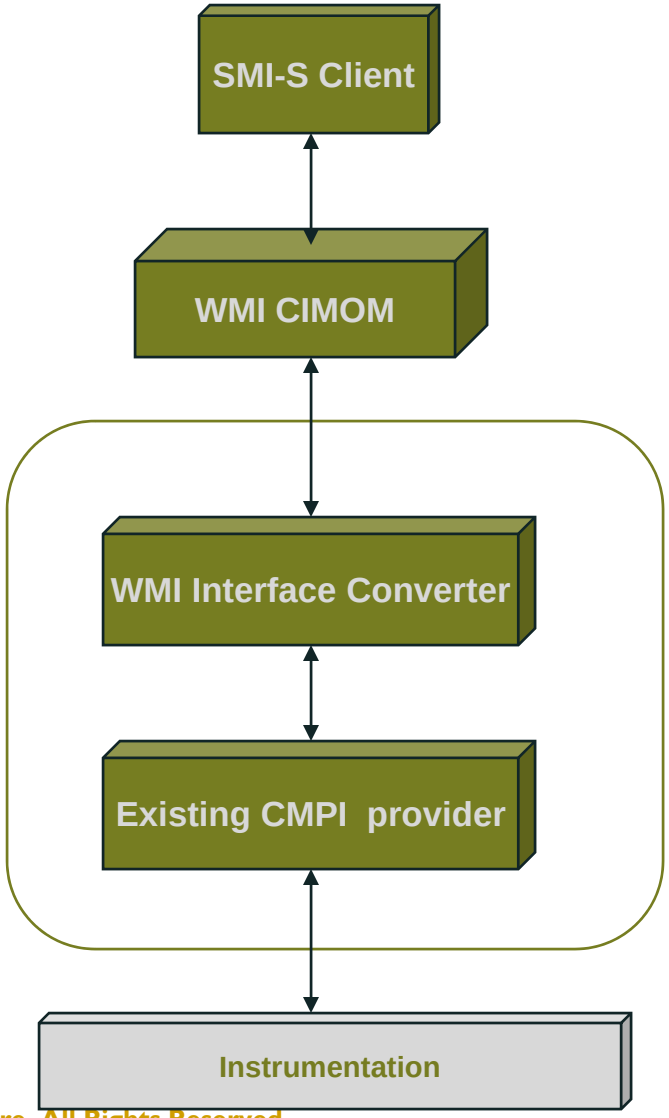
Interface Adapter Architecture – EnumInstanceNames Sample Code

Generic Interface for Profile Implementation

```
Function StorageVolumeProviderEnumInstanceNames(..)
{
   // Initialization
  Get volume list
   for( each Volume)
   {
            //Create new instance of generic object path "newObjPath" and insert all
                  //key properties.
                  SystemCreationClassName = getSysCreationalClassName();
                  newObjPath.AddKey("SystemCreationClassName“,
                                                SystemCreationClassName.c_str(),STRING);

    }
                  // Return the  object path to Provider Framework.
                  return generic object path;
}
```

Interface Converter Architecture



SMI-S Provider

Interface Converter Architecture

- *Convert or adapt CMPI interface to WMI interface. Adapting CMPI to Native interface is also possible. However this combination is rare.*

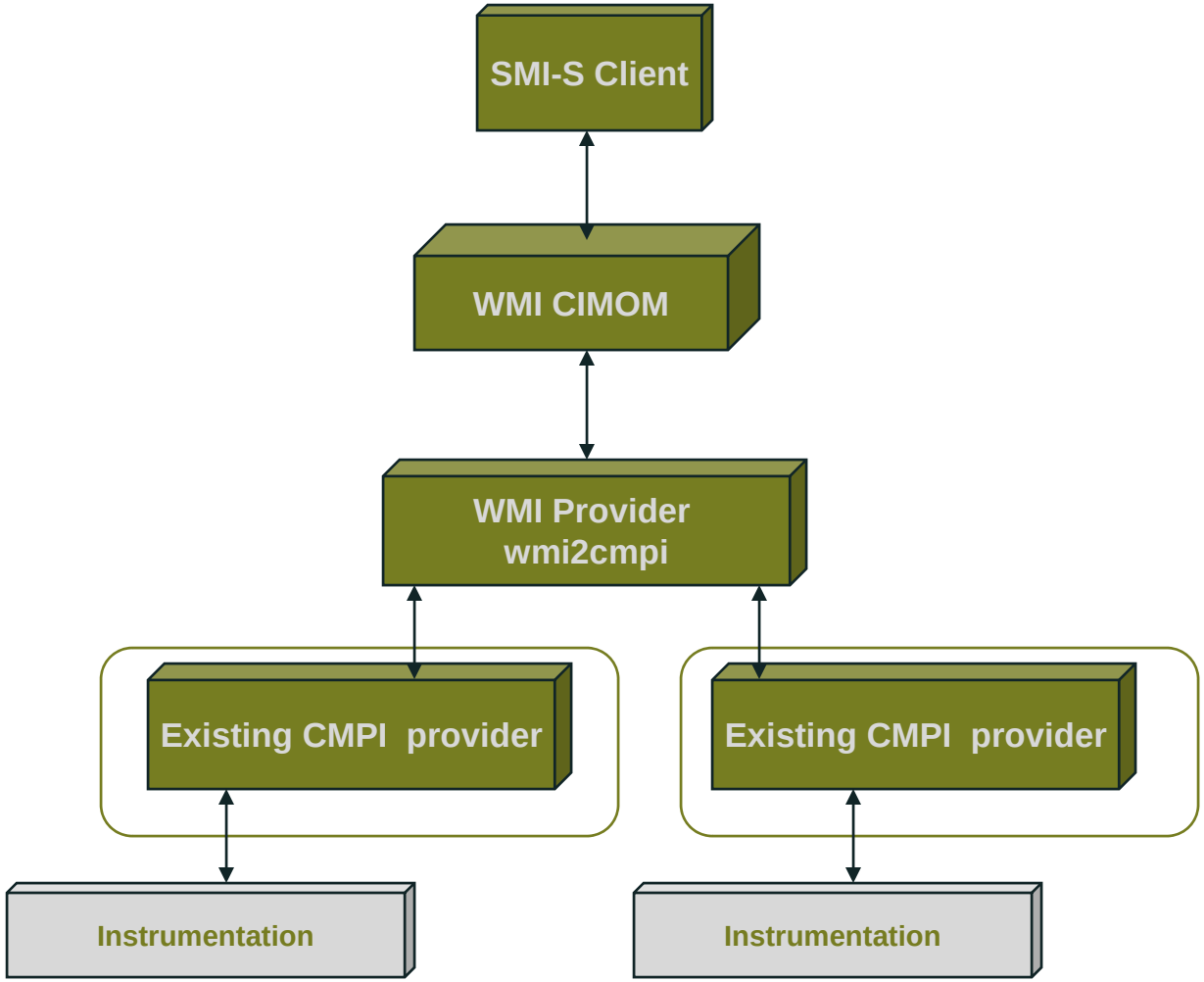
Interface Converter Architecture – Pros

- *The existing CMPI/Native based providers can be reused.*
- *Maintaining separate code base for different provider is not needed.*
- *Test efforts are reduced since there is no need to test different providers only converter layer need to be tested.*

Interface Converter Architecture – Cons

- *Adapting WMI interface to CMPI interface across platform i.e., Windows to Linux is not possible.*
- *Slow performance as request and result needs to be translated from CMPI interface to WMI and vice-versa.*
- *CMPI and WMI interface may not have one to one mapping. Such situations are difficult to handle and are source of errors.*

WMI2CMPI



WMI2CMPI - Pros

- *The existing CMPI/Native based providers can be reused.*
- *Maintaining separate code base for different provider is not needed.*

WMI2CMPI - Cons

- *Slow performance as request and result will be translated from WMI Provider to CMPI Provider and vice-versa by the wmi2cmpi adapter layer, which is a WMI Provider.*
- *CMPI and WMI interfaces may not have one to one mapping. Such situations are difficult to handle and are source of errors.*

What do I do with all these ?

- Depending on requirements (current/future), schedule and long-term view decide on an approach that is valid for your SMI-S Providers
- Storage Hardware vendors whose wares are being given to multiple OEMs are better off choosing the Interface Adapter Architecture
- Storage Hardware meant for a specific environment can choose to have only native interface (CMPI if supported natively) based SMI-S Providers

SOW Update

- SMI-S Provider Performance should be such that any enumeration does not take more than ***x*** time.

Performance Improvements

- SMI-S Providers that fetch data from the storage hardware based on any request from client are inherently slow.
- Implement Caching Mechanisms in the SMI-S Provider to improve performance.
- An example for doing cache management in SMIS- Provider:
  - Creation of SMI-S Provider managed element cache at SOD
  - Update cache asynchronously at regular intervals or based on configuration changes (via CIM Methods)
  - If supported by storage hardware/firmware update the cache based on events received for device status changes or configuration changes

Low Hanging Fruits

- Vendor or OEM specific extensions to SMI-S Profiles creates interoperability issues

References

- SMI-S - http://www.snia.org/tech_activities/standards/curr_standards/smi/
- WBEM - http://www.dmtf.org/standards/wbem/
- WMI - http://msdn.microsoft.com/en-us/library/aa394582(VS.85).aspx
- CMPI v1.0 - http://www.opengroup.org/bookstore/catalog/c051.htm
- CMPI v2.0 - http://www.opengroup.org/bookstore/catalog/c061.htm
- OpenPegaus Project - http://www.openpegasus.org/
- CIMPLE Tool - http://simplewbem.org/

Queries