



December 3-6, 2007, Santa Clara Marriott, Santa Clara, CA

Python CIM Providers with PyWBEM

Bart Whiteley



Agenda

- Motivation
- Design Goals
- Benefits
- Architecture
- Interface Details
- Demonstration
- References

Motivation

- Current provider interfaces (CIMOM-specific C++, CMPI, CIMPLe, ...) expose too many details of the WBEM operations.
- Current provider interfaces require too much boilerplate code.
- Providers required more build management effort than they should.
- Inspired by the ease and power of the PyWBEM client API.
- Leverage the dynamic nature of Python.

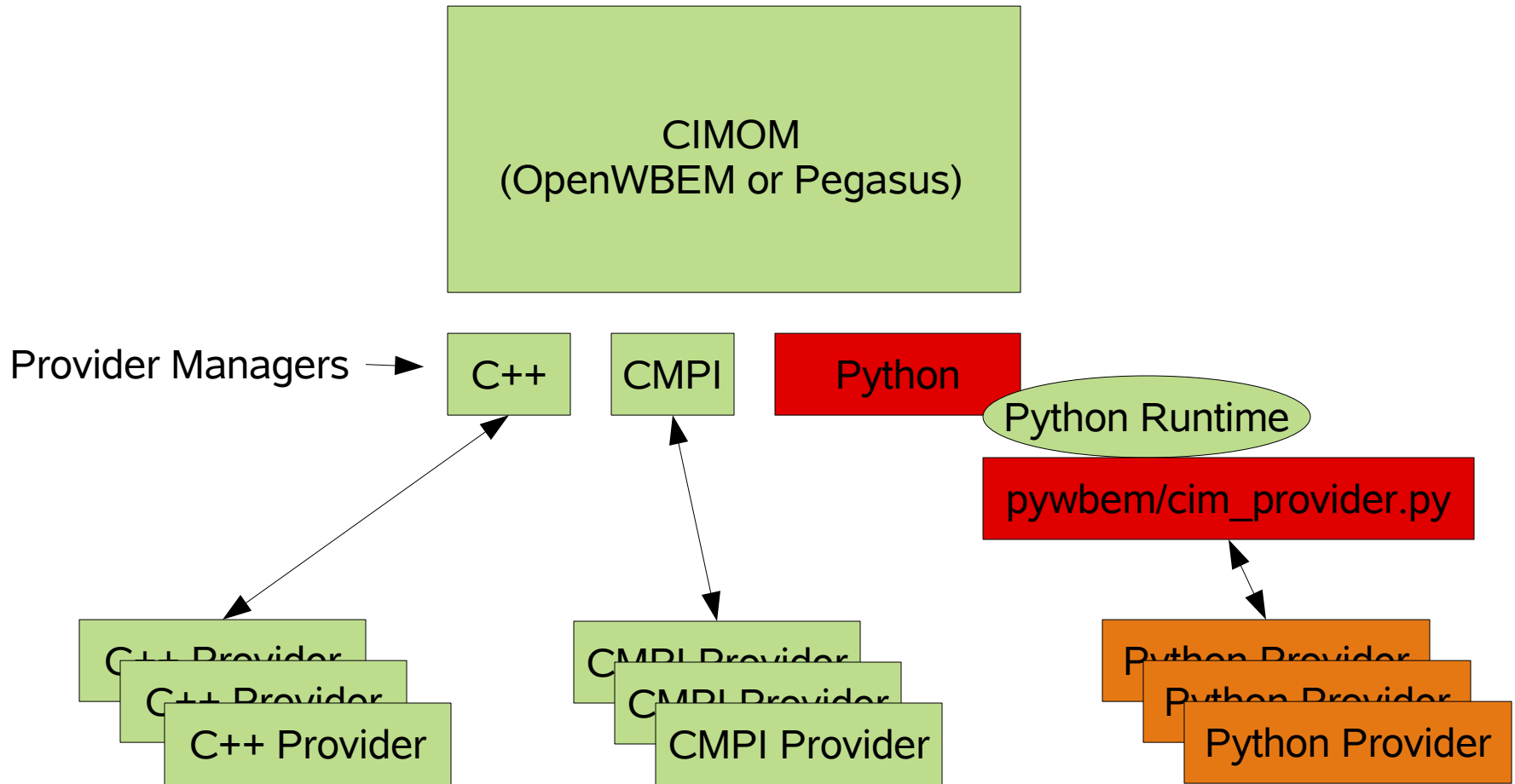
Design Goals

- Don't require provider writers to do anything that the provider interface could do for them.
- Ease of development.
- Ease of deployment.
- Leverage PyWBEM objects.
 - PyWBEM objects can be passed directly from PyWBEM client to python provider interface.
- Consistency with PyWBEM client API.
 - “Upcall” CIMOM handle matches `pywbem.WBEMConnection`.
- CIMOM neutral.
- Reduce operations.
- Reduce parameters.

Benefits

- Easy to debug.
- Provider writers can focus on solving the problem of instrumentation.
- Greatly reduced time to market.
- Less opportunity for error.
- Reduced LOC
 - Process Provider
 - C++: 3600
 - CMPI: 3338
 - Python: 800
 - File System Provider
 - C++: 2900
 - CMPI: 1383 (subset of classes/features)
 - CIMPLe: 1230 + 8650 generated
 - Python: 440

Architecture Diagram



Architecture

- Implemented as a provider manager for OpenWBEM and Pegasus.
- Converts native CIMOM objects to PyWBEM objects.
- Loads Python modules (the providers).
- Provides “upcall” CIMOM handle and other CIMOM services to providers through Python extensions.
- Python abstraction layer (`cim_provider.py`) maps WBEM operations from CIMOM to simplified, more friendly Python provider interface.

Interface Concepts

- Combine similar operations into one method.
 - EnumerateInstanceNames, EnumerateInstances --> `enum_instances()`.
 - Associators, AssociatorNames, References, ReferenceNames --> `references()`.
 - CreateInstance, ModifyInstance --> `set_instance()`
- Pass to the provider a “model” of the object(s) the provider should return.
 - The model is completed as much as possible by the interface.
 - The model conveys more information, such as PropertyList.
- Eliminate the need for providers to dispatch extrinsic methods.

'model' Parameter

- A `pywbem.CIMInstance` passed to instance and association methods.
- Attributes:
 - `path` – A `pywbem.CIMInstanceName` representing the `ObjectPath` of the instance.
 - `classname` – The name of the class.
 - `properties` – A case-insensitive dictionary containing the properties.
- Methods:
 - `update_existing(<mapping>)` -- Update the values of properties, iff the properties are already present in the instance.
- Behaves like a Python dictionary.
 - `model['propname']` returns/assigns the property **value**.
 - `model.properties['propname']` returns/assigns the `pywbem.CIMProperty` instance.

'model' continued.

- Properties in the model correspond to the PropertyList parameter from the request, if present.
 - If a property is not present in the model, no need to compute and set the property value on the instance.
 - `update_existing()` method is useful for this.
- Providers can chose to ignore the PropertyList information, and set all of the properties.
 - The provider interface will filter for you.
 - This is configurable.
- In the case of `get_instance`, Key properties are set before calling into the provider.
- The provider should update the model, and return or yield it, depending on the method.

pywbem.CIMProvider Class

- Python base class for Instance, Association, and Method providers.
- A single subclass of pywbem.CIMProvider can service multiple CIM classes.
- A single Python provider module can contain multiple subclasses of pywbem.CIMProvider.
- CIMProvider methods:
 - `__init__()`
 - `get_instance()`
 - `enum_instances()`
 - `set_instance()`
 - `delete_instance()`
 - `references()`
 - Extrinsic methods.

get_instance

```
def get_instance(self, env, model, cim_class):
```

- Return the requested pywbem.CIMInstance.
- If the model does not correspond to a valid instance, raise pywbem.CIMError(pywbem.CIM_ERR_NOT_FOUND).
- Otherwise, set additional properties on the model, and return the model.
- Keyword arguments:
 - env – Provider environment
 - model – A template of the pywbem.CIMInstance to be returned. The key properties are set corresponding to the requested instance. If the request included a PropertyList, only properties in the PropertyList are present in the model.
 - cim_class – A pywbem.CIMClass representing the CIM class whose instance is requested.

enum_instances

```
def enum_instances(self, env, model, cim_class,  
    keys_only):
```

- Python generator used to enumerate instances.
- For each instance, update the properties on the model, then yield the model.
- Keyword arguments:
 - env – Provider Environment.
 - model – A template of the pywbem.CIMInstances to be generated. Similar to get_instance.
 - keys_only – True if request was EnumerateInstanceNames. False if request was EnumerateInstances.
 - cim_class – A pywbem.CIMClass representing the CIM class whose instance is requested.

set_instance

```
def set_instance(self, env, instance, previous_instance,  
    cim_class):
```

- Create or Modify an instance, or raise a `pywbem.CIMError` with an appropriate `pywbem.CIM_ERR_*`.
- Keyword arguments:
 - `env` – Provider environment.
 - `instance` – A `pywbem.CIMInstance` representing the new instance.
 - `previous_instance` – `None` if doing a `CreateInstance`, otherwise the `pywbem.CIMInstance` representing the old instance prior to modification.
 - `cim_class` – A `pywbem.CIMClass` representing the CIM class whose instance is requested.

delete_instance

```
def delete_instance(self, env, instance_name):
```

- Delete the specified instance, or raise a `pywbem.CIM_Error` with an appropriate `pywbem.CIM_ERR_*`.
- Keyword arguments:
 - `env` – Provider environment
 - `instance_name` – A `pywbem.CIMInstanceName` representing the instance to be deleted.

references

```
def references(self, env, object_name, model, assoc_class,  
              result_class_name, role, result_role, keys_only):
```

- Handles all association operations (Associators, AssociatorNames, References, ReferenceNames).
- Python generator that yields instances of `pywbem.CIMInstance` representing CIM instances of the association class.
- Keyword arguments:
 - `env` – Provider environment
 - `object_name` – The target object
 - `model` – A template of the `pywbem.CIMInstance` to be returned.
 - `result_class_name`, `role`, `result_role` – See DSP0200.
 - `keys_only` – True if handling `ReferenceNames`.
 - `assoc_class` – A `pywbem.CIMClass` representing the CIM association class whose instance is requested.

Method Providers

- Providers don't need to dispatch methods.
- Just implement a python method matching the signature of the CIM method.
 - Method name is prefixed with “cim_method_”
 - Method IN parameters are prefixed with “param_”
- Return a 2-tuple containing the return value, and a dictionary with the OUT parameters.

Method Provider Example

- MOF:

```
uint32 RequestStateChange([IN] RequestedState,  
    [IN(false), OUT] CIM_ConcreteJob REF Job,  
    [IN] datetime TimeoutPeriod);
```

- Python:

```
def cim_method_requeststatechange(self, env,  
    object_name, method, param_requestedstate,  
    param_timeoutperiod):
```

- object_name – A pywbem.CIMInstanceName or pywbem.CIMClassName indicating the target object.
- method – A pywbem.CIMMethod representing the CIM method definition.
- param_requestedstate and param_timeoutperiod are the IN parameters.
- Return a 2-tuple containing the return value (pywbem.Uint32) and the dictionary
{'job':pywbem.CIMInstanceName('CIM_ConcreteJob', ...)}

Module Functions

- Required:
 - `get_providers(env)`
 - Returns a dictionary mapping CIM class names to instances of Python provider classes.
- Optional:
 - `init(env)`
 - Optional. First method called.
 - `shutdown(env)`
 - Called when the provider is unloaded.
 - `can_unload(env)`
 - Return True if the provider can be unloaded.
 - Indication related methods.
 - `handle_indication`, `activate_filter`, `authorize_filter`, `deactivate_filter`.
 - Not covered in this presentation.

Provider Environment

- The first parameter passed to most functions or methods.
- Contains CIMOM services made available to providers.
- `env.get_logger()`
 - Return a logger object.
- `env.get_cimom_handle()`
 - Return an “upcall” CIMOM handle.
 - “Upcall” CIMOM handle interface resembles WBEM operations (DSP0200).
 - “Upcall” CIMOM handle interface is compatible with `pywbem.WBEMConnection`.



Provider Environment Example

```
logger = env.get_logger()  
logger.log_debug('Debug Info')  
ch = env.get_cimom_handle()  
other_inst = ch.GetInstance(inst_path,  
    LocalOnly=False, IncludeQualifiers=False,  
    IncludeClassOrigin=False)
```

Code Generation

- Writes much of the provider for you.
- Generated code is specific to the CIM class being instrumented.
 - Intrinsic method stubs determined by the nature of the class.
 - Property names and types.
 - Extrinsic methods
 - Method names.
 - Parameter names and types.
 - Return types.
- Promotes consistent patterns across providers.
- Comments in the code teach how to write Python providers.
- Code generation is optional (compare CIMPLe).
- Greatly reduces ramp-up time.
 - Just follow the instructions and fill in the blanks.

codegen

`codegen(cim_class)`

- `cim_class` is a `pywbem.CIMClass`
 - Make sure it has all of the properties, methods, and qualifiers.
- Returns a 2-tuple containing:
 - Syntactically correct Python provider module code, ready to run (though it won't do much).
 - Provider registration MOF.

Debugging

- Provider manager checks the timestamp of provider modules, and reloads if timestamp is newer.
 - Effortless to try the latest changes to provider code.
- Errors from providers, including syntax errors, are sent as stack traces to the client application.
- Debugging becomes quick and easy.
 - Nothing to rebuild.
 - Nothing to reinstall.

Demonstration

A live demo that includes:

- Generating provider module code using `pywbem.codegen`.
- Implementing the instance-related methods and extrinsic methods.
- Deploying and using the provider.
- Similar to the QuickStart Guide on the wiki.
 - <http://pywbem.wiki.sourceforge.net/Provider+QuickStart>

Summary

- The easiest way to write and maintain providers.
- Simplified interface.
- Easy debugging.
- Easy deployment.
- CIMOM neutral.

References

- <http://pywbem.sourceforge.net/>
- <http://pywbem.wiki.sourceforge.net/>
- <http://pywbem.wiki.sourceforge.net/Provider+QuickStart>
- <http://omc.svn.sourceforge.net/viewvc/omc/pybase/trunk/>
- <http://omc.svn.sourceforge.net/viewvc/omc/pyprofiles/>
- http://www.dmtf.org/standards/published_documents/DSP200.html
- <http://www.python.org/dev/peps/pep-0008/>