Automatic Generation of SMI Providers

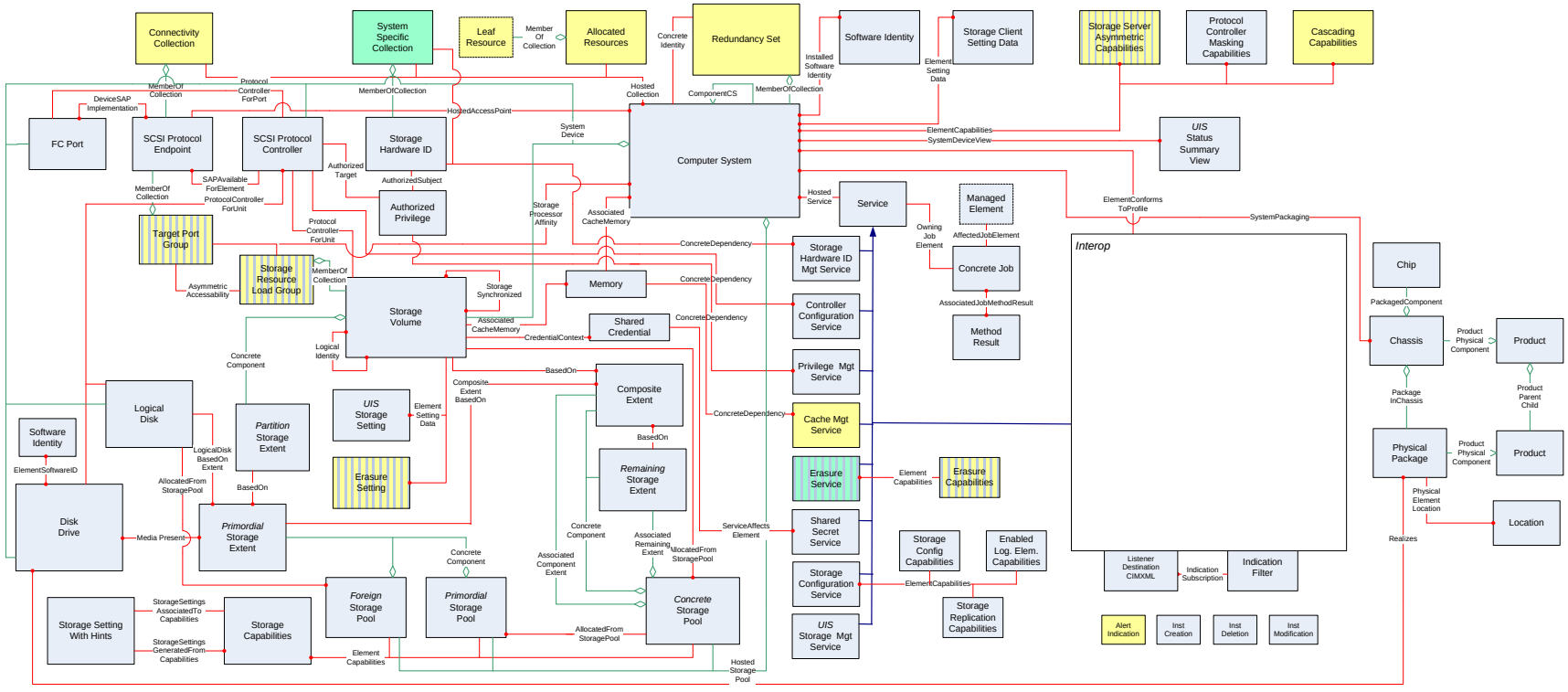
David Dodgson

Unisys

Objectives

- Learn why automatic generation is useful.
- Learn about the additions to Simple WBEM (cimple) used to make it work.
- Learn how CIM instances are stored and retrieved.
- Learn how the provider is built.

Why Use Automatic Generation?



Why Use Automatic Generation?

- SMI Profiles supported:
  - SNIA : Storage Virtualizer
  - SNIA : Array
  - SNIA : Server
  - UIS : Stealth DAR
- Classes supported:
  - 102 CIM classes
  - 9 SNIA classes
  - 31 UIS classes
  - Total of 142 classes

Open Source Support

- Pegasus
  - Used to provide CIM support
  - No changes to source
- Simple WBEM (cimple)
  - Used to create provider
  - Source additions made to support automatic generation

Automatic Generation

- Datastore Class
  - Provides CIM support
- Tools
  - Writes full provider code, not just skeleton
  - Supports instances, associations, and indications
- Build Procedures
  - Creates provider dll

Datastore Class

- This class provides an in-memory repository for all CIM instances.
- It provides the CIM interfaces.
  - Get, Delete, Create, Modify
  - Enumerate, EnumerateNames
  - Associators, AssociatorNames, References, ReferenceNames
- References are updated when stored to point to instances internal to the datastore.

Datastore Class

- It handles multiple namespaces.
- It handles indications on Create, Modify, and Delete.
  - The user derives a function object from the IndicationRegistration class.
  - The function object is registered.
  - The signal method of the function object is called when the indication occurs.
- The datastore can be printed to a file.

Datastore Class

- The Datastore is a collection of Namespaces.
- A Namespace is a set of Containers plus a set of Class maps.
- A Class map is a set of Containers for that class.
- A Container is a reference-counted object pointing to the data for an instance.
- Creates and Deletes are recursive, storing the container in the Namespace set and by each of its base classes.

Datastore Class

- A lookup is done by its class.
- Each set is a binary tree to speed look-up.
- Instances are compared by common keys. If they have no common keys, they are compared by class name or container index.
- Enumerations are a traversal of that class's tree.

Tools

- genclass is used to generate class information.
- genprovD generates provider code using the Datastore class.
- genmodD generates the provider module using the Datastore class.
- genmak generates the provider make file.

Provider Example


```
Get_Instance_Status CIM_MediaPresent_Provider::get_instance(
    const CIM_MediaPresent* model,
    CIM_MediaPresent*& instance)  {
   Instance *p;
   trace( "CIM_MediaPresentGetInstance", cast<Instance*>(model) );
   switch ( ds.get( model, p ) )  {
     case Datastore::DATASTORE_OK:
          instance = cast<CIM_MediaPresent*>(p);
          return GET_INSTANCE_OK;
     case Datastore::DATASTORE_NOT_FOUND:
     case Datastore::DATASTORE_INVALID_NAMESPACE:
     case Datastore::DATASTORE_INVALID_CLASS:
          return GET_INSTANCE_NOT_FOUND;
   /* Other error cases */
   };
   return GET_INSTANCE_FAILED;
}
```

Provider Example


```
Enum_References_Status
   CIM_MediaPresent_Provider::enum_references(
    const Instance* instance, const CIM_MediaPresent* model,
    const String& role,
   Enum_References_Handler<CIM_MediaPresent>* handler)
{  Array<Instance*> arr;
   trace( "CIM_MediaPresentEnumReferences",
        const_cast<Instance*>(instance) );
   switch ( ds.references( instance, arr, "CIM_MediaPresent",
        role.c_str() ) )
   {    case Datastore::DATASTORE_OK:
        for ( size_t i = 0; i < arr.size(); ++i )
          handler->handle( cast<CIM_MediaPresent*>(arr[i]) );
        return ENUM_REFERENCES_OK;
       case Datastore::DATASTORE_NOT_FOUND:
       /* Error handling */
   };
   return ENUM_REFERENCES_FAILED;
}
```

Provider Example


```
Modify_Instance_Status
   CIM_MediaPresent_Provider::modify_instance(
    const CIM_MediaPresent* model,
    const CIM_MediaPresent* instance)
{  Instance *p;
   CIM_MediaPresent* target;
   trace("CIM_MediaPresentModifyInstance",cast<Instance*>(model));
   switch ( ds.get( model, p ) )
   {
   case Datastore::DATASTORE_OK:
        target = cast<CIM_MediaPresent*>(p);
        break;
   case Datastore::DATASTORE_NOT_FOUND:
   /* Error handling */
   };
```

Provider Example


```
    copy( target, instance, model );

    switch ( ds.modify( target ) )
    {
    case Datastore::DATASTORE_OK:
         return MODIFY_INSTANCE_OK;
    case Datastore::DATASTORE_NOT_FOUND:
    case Datastore::DATASTORE_INVALID_NAMESPACE:
    case Datastore::DATASTORE_INVALID_CLASS:
         return MODIFY_INSTANCE_NOT_FOUND;
    case Datastore::DATASTORE_ACCESS_DENIED:
         return MODIFY_INSTANCE_ACCESS_DENIED;
    case Datastore::DATASTORE_INVALID_PARAMETER:
         return MODIFY_INSTANCE_INVALID_PARAMETER;
    };
    return MODIFY_INSTANCE_FAILED;
}
```

Build Procedures

- Classlist.txt.
  - This list contains all the classes to be generated.
- Provlist.txt
  - This list contains all the class providers to be generated.

Some class providers require code to be written manually. These are generally methods for services. They are automatically generated the first time and then removed from Provlist.txt.

Buildlist.bat

- Used to generate a file (Classlist.cpp) with all the class cpp files.

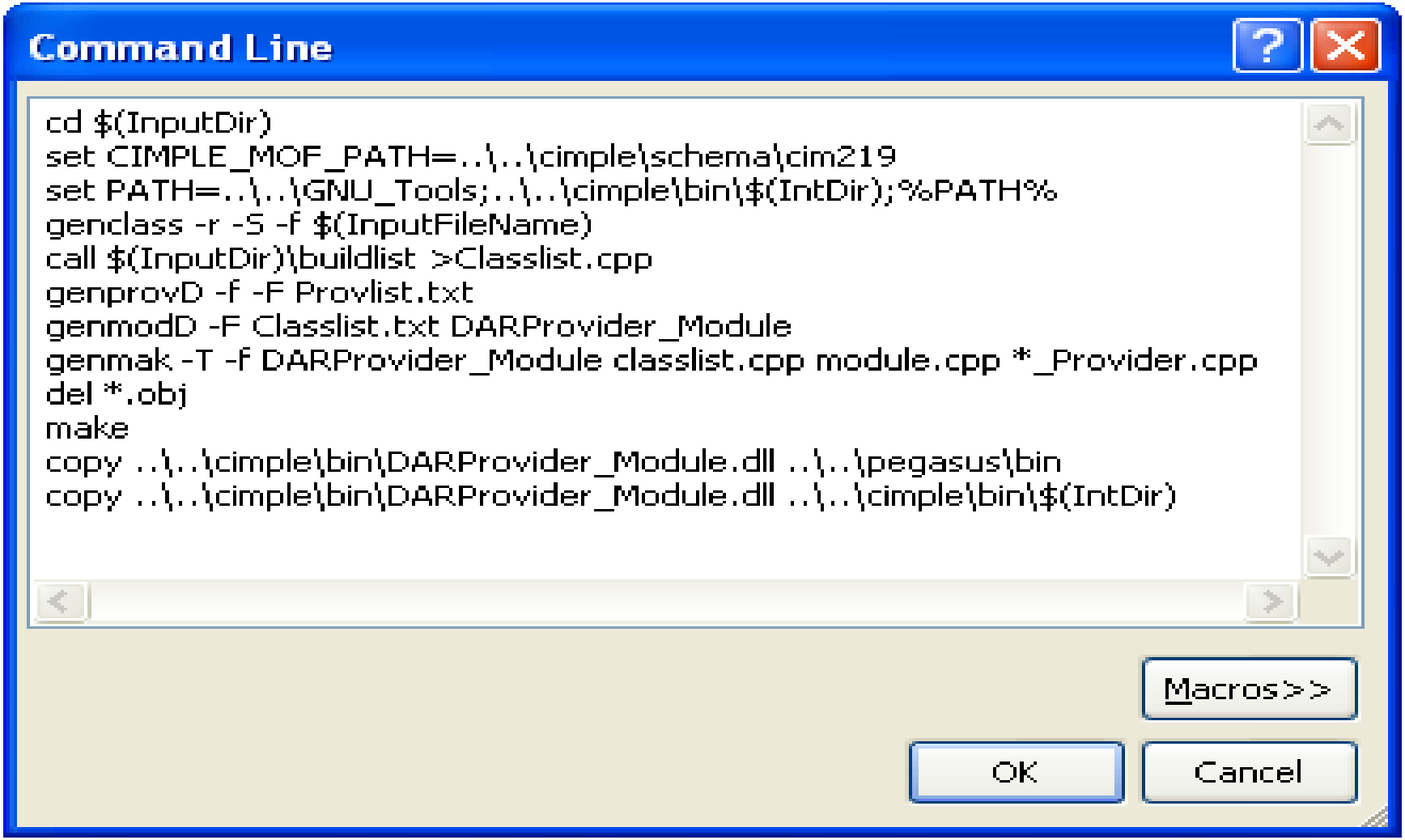
```
for /F %%f in (.genclass) do @echo #include "%%f"
```

Make

- Make target

```
repository:
  genclass -r -S -f Classlist.txt
  call buildlist >Classlist.cpp
  genprovD -f -F Provlist.txt
  genmodD -F Classlist.txt ClientProvider_Module
  genmak –F MakeProvider -f ClientProvider_Module
   classlist.cpp module.cpp *_Provider.cpp
  $(MAKE) –f MakeProvider
  copy ClientProvider_Module.dll,
   $(subst /,\,$(WITH_PEGASUS_OPT))\bin
```

Visual Studio Custom Build



Steps to Add a Class

- Make sure the .mof is defined, if not in a CIM class then in your own .mof file. Execute cimmof to update the repository with your mof definition.
- Update Classlist.txt and Provlist.txt with the new class.
- Build the provider DLL and copy it under the Pegasus bin directory.
- Execute regmod to register your provider if any class definitions have changed.
- Add custom code as needed to the class provider .cpp file and remove it from Provlist.txt.

Custom Code Example


```
Invoke_Method_Status PrintDeviceList(
        const UIS_StorageManagementService* self,
        const Property<boolean>& Verbose,
        const Property<String>& Filename,
        const Property<String>& Label,
        Property<uint32>& return_value);
Invoke_Method_Status ClearDS(
        const UIS_StorageManagementService* self,
        Property<uint32>& return_value);
Invoke_Method_Status TraceProvider(
        const UIS_StorageManagementService* self,
        const Property<boolean>& Trace,
        Property<uint32>& return_value);
```

Custom Code: PrintDeviceList


```
//UIS_StorageManagementService_Provider::PrintDeviceList
{  FILE* out = NULL;
   time_t timeval;
   return_value = 1;
   trace( "ServicePrintDeviceList",cast<Instance*>(self) );
   out = fopen( Filename.value.c_str(), "a+" );
   if ( out )
   {
       time( &timeval );
       fprintf(out,"\n** %s Open Log at %s\n",
              Label.value.c_str(),
              asctime( localtime( &timeval ) ) );
       ds.fprint( out, Verbose );
```

Custom Code: PrintDeviceList


```
        time( &timeval );
        fprintf( out, "\n** %s Close Log at %s\n",
                Label.value.c_str(),
                asctime( localtime( &timeval ) ) );
        fflush( out );
        fclose( out );
        return_value = 0;
    };
    return INVOKE_METHOD_OK;
}
```

Custom Code: ClearDS


```
Invoke_Method_Status
   UIS_StorageManagementService_Provider::ClearDS(
     const UIS_StorageManagementService* self,
     Property<uint32>& return_value)
{
     ds.clear();
     return_value = 0;
     return INVOKE_METHOD_OK;
}
```

Custom Code: TraceProvider


```
//UIS_StorageManagementService_Provider::TraceProvider(
{  time_t timeval;
   char buffer[80];
   time( &timeval );
   _snprintf(buffer,sizeof(buffer),"\n** Trace %s at %s\n",
       Trace.value ? "ON" : "OFF",
       asctime( localtime( &timeval ) ) );
   traceflag = traceflag || Trace.value;
   trace( buffer, NULL );
   traceflag = Trace.value ? 1 : 0;
   return_value = 0;
   return INVOKE_METHOD_OK;
}
```

Summary

- Automatic generation is very useful when dealing with a large number of classes, particularly if their MOF files have already been defined.
- The addition of the Datastore class to Simple WBEM and updates to the generation tools make automatic generation possible.
- Build tools such as make or Visual Studio can build the provider in a single command.

Links

- Unisys Corporation
  http://www.unisys.com
- Simple WBEM
  http://www.simplewbem.org
- Pegasus
  http://www.openpegasus.org

Questions

??