
# CIMPLE

## A new kind of provider environment

Mike Brasher, Karl Schopmeyer

**Inova Development**

December 6, 2005

Version 1.0

MANAGEMENT DEVELOPERS CONFERENCE

# Introduction

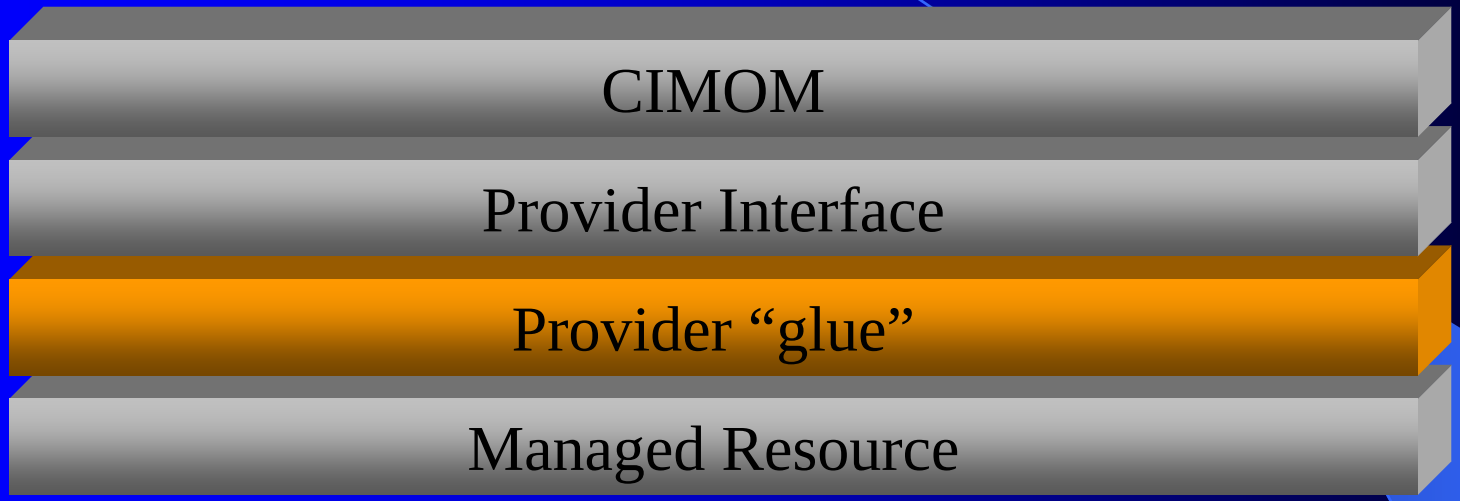
MANAGEMENT DEVELOPERS CONFERENCE

# What is CIMPLE?

- **CIMPLE** is an environment for developing providers with three key advantages.
  - They are very easy to build.
  - They are small.
  - They are fast.
  - They are more reliable.

MANAGEMENT DEVELOPERS CONFERENCE

# Conventional Provider




The provider is mostly “glue” that maps a managed resource to a provider interface. **CIMPLE** eliminates the need to develop much of the provider.

MANAGEMENT DEVELOPERS CONFERENCE

# Key simplifications

- Eliminates the need to implement several of the provider operations.
- Generates real classes in the target language from MOF classes.
- Generates the provider skeleton and CIM interface automatically.

MANAGEMENT DEVELOPERS CONFERENCE

# Provider Operations

- Get-instance
- Get-instance-names*
- Enum-instance
- Enum-instance-names*
- Get-property*
- Set-property*
- Create-instance
- Modify-instance
- Delete-instance
- Associators*
- Associator-names*
- References*
- Reference-names*
- Method-stubs*

*CIMPLE provides these operations

MANAGEMENT DEVELOPERS CONFERENCE

# Class generation

**CIMPLE** generates classes in the target language from MOF classes.

```
class Fan
{
    [key] string DeviceID;
    uint64 Speed;
    uint64 DesiredSpeed;

    uint64 SetSpeed(
      [in] uint64 speed);
};
```

```
class Fan
{
    Value<string> DeviceID;
    Value<uint64> Speed;
    Value<uint64> DesiredSpeed;
    CIMPLE_CLASS(Fan);
};
```

MANAGEMENT DEVELOPERS CONFERENCE

# Provider skeleton generation

- Generates skeletons for instance operations.

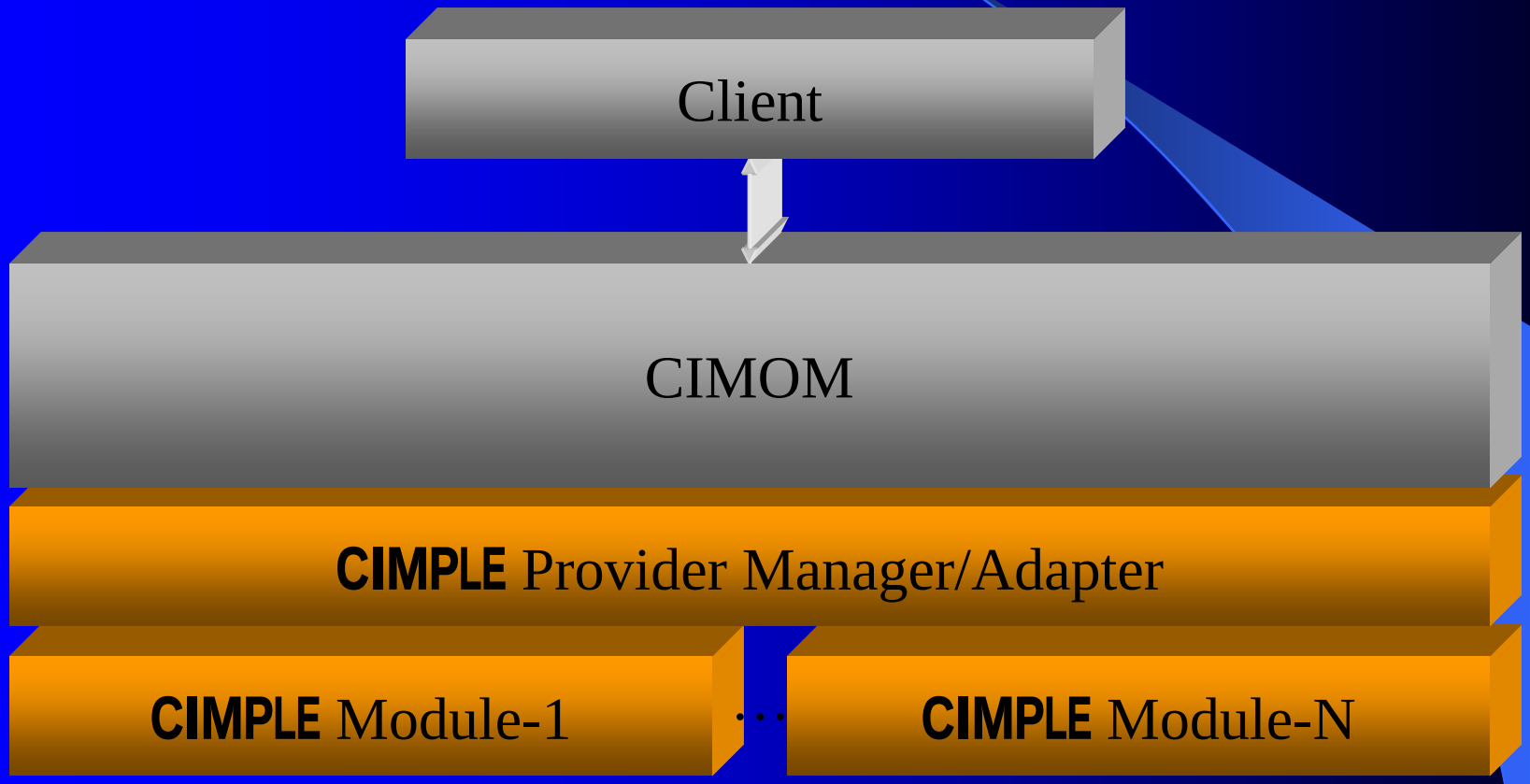
```
Get_Instance_Status Fan_Provider::get_instance(Fan* inst)
{
    return GET_INSTANCE_UNSUPPORTED;
}
```

- Generates method signatures and method dispatchers.

```
Invoke_Method_Status Fan_Provider::SetSpeed(
    const Fan* inst,
    const Value<uint64>& DesiredSpeed,
    Value<uint32>& return_value)
{
    return INVOKE_METHOD_UNSUPPORTED;
}
```

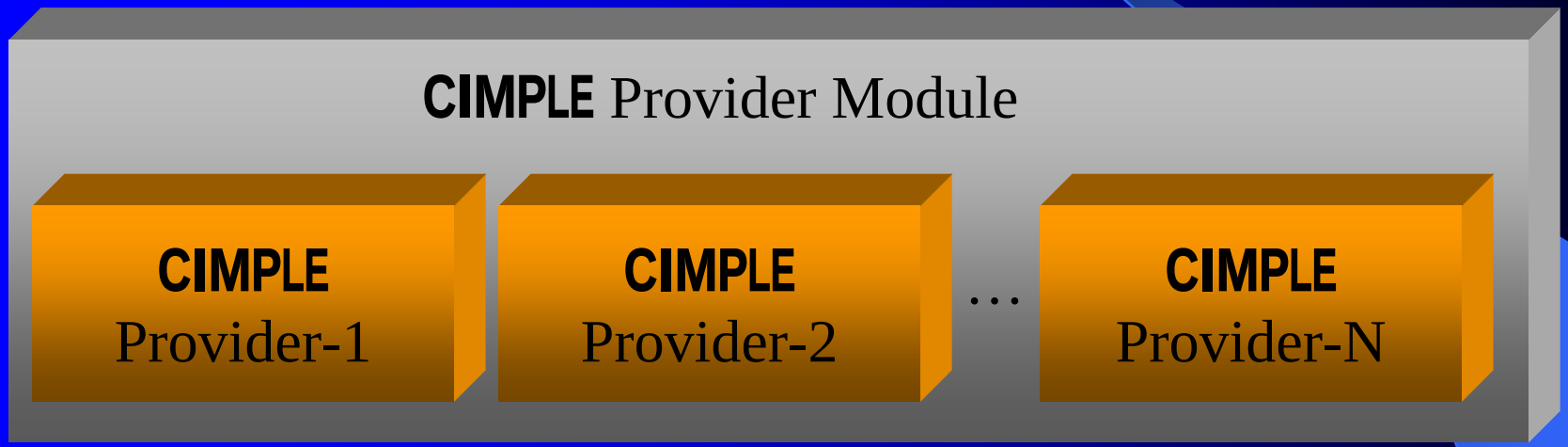
MANAGEMENT DEVELOPERS CONFERENCE

# Overall **CIMPLE** Architecture



MANAGEMENT DEVELOPERS CONFERENCE

# CIMPLE Module Architecture



MANAGEMENT DEVELOPERS CONFERENCE

# Motivation

MANAGEMENT DEVELOPERS CONFERENCE

# The Problem

- Convention providers are:
  - **hard** to develop and maintain.
  - **error prone** – errors caught at run-time rather than compile-time.
  - too **large** for some environments (embedded).
  - too **slow** for some hardware (embedded).
- Management of the provider infrastructure dominates the provider development effort.

MANAGEMENT DEVELOPERS CONFERENCE

# Dynamic interfaces

```
Uint32 GetSpeed(CIMInstance& fan) throw(Exception)
{
    Uint32 pos = fan.findProperty("speed");

    if (pos == PEG_NOT_FOUND)
        throw Exception("not found");

    try
    {
        Uint32 speed;
        fan.getProperty(pos).getValue().get(x);
        return speed;
    }
    catch(...)
    {
        throw Exception("type mismatch");
    }
}
```

MANAGEMENT DEVELOPERS CONFERENCE

# What if Fan were a real class?

```
inline Uint32 GetSpeed(const Fan& fan)
{
    return fan.Speed.value;
}

// Or just ‘fan.Speed.value’
```


# Pegasus/CIMPLE

```
void Pegasus_example()
{
  try
  {
    CIMInstance inst("TheClass");
    inst.addProperty(CIMProperty("u", "hello"));
    inst.addProperty(CIMProperty("v", Uint32(99)));
    inst.addProperty(CIMProperty("w", Boolean(true)));
    inst.addProperty(CIMProperty("x", Real32(1.5)));

    Array<Uint32> y;
    y.append(1);
    y.append(2);
    y.append(3);
    inst.addProperty(CIMProperty("y", y));
    Uint32 pos = inst.findProperty("v");

    if (pos != PEG_NOT_FOUND)
    {
      Uint32 v;
      CIMProperty prop = inst.getProperty(pos);
      prop.getValue().get(v);
    }
  }
  catch(...)
  {
  }
}
```

```
void CIMPLE_example()
{
  TheClass* inst = TheClass::create();
  inst->u.value = "hello";
  inst->v.value = 99;
  inst->w.value = true;
  inst->x.value = 1.5;
  inst->y.value.append(1);
  inst->y.value.append(2);
  inst->y.value.append(3);
  uint32 v = inst->v.value;
  destroy(inst);
}
```

MANAGEMENT DEVELOPERS CONFERENCE

# Speed/size comparison

| | Pegasus | CIMPLE | Improvement |
|---|---|---|---|
| **Speed*** | 11.04 seconds | 0.6 seconds | 18 times faster |
| **Object size** | 2102 bytes | 240 bytes | 9 times smaller |

* One million iterations

MANAGEMENT DEVELOPERS CONFERENCE

# Advantages

MANAGEMENT DEVELOPERS CONFERENCE

# CIMPLE advantages

- **Easier** – eliminates busy work, allowing one to concentrate on the core problem.
- **Smaller** – reduces object size and memory usage.
- **Faster** – improves performance.
- **Reliable** – catches at compile time rather than run-time.
- **Interoperable** – could potentially run under any CIMOM.

MANAGEMENT DEVELOPERS CONFERENCE

# Easier

- Development easier with ***first class objects***.
- Fewer instance provider operations to implement.
- No need to implement association operations.
- Method signatures automatically generated.
- Tools automate complex activities:
  - **genclass –** generates C++ classes from MOF.
  - **genprov –** generates provider skeletons.
  - **regmod –** registers provider with Pegasus.
  - **testmod –** pre-loads and tests providers.

MANAGEMENT DEVELOPERS CONFERENCE

# Smaller

- **CIMPLE** providers have smaller **object size**.
- **CIMPLE** providers use **heap size**.

MANAGEMENT DEVELOPERS CONFERENCE

# CIMOM vs. provider size

- The object size of the CIMOM is a fixed cost.
- The cost of the provider is a variable cost (it increases as you add or enhance providers).

MANAGEMENT DEVELOPERS CONFERENCE

# Faster

- Use of first-class objects reduces overhead by one order of magnitude (no longer necessary to search property lists).

MANAGEMENT DEVELOPERS CONFERENCE

# Reliable

Due to first class objects:

- more errors can be detected at compile time.
- complexity is substantially reduced.

MANAGEMENT DEVELOPERS CONFERENCE

# Interoperable

- **CIMPLE** was designed to work with any CIMOM (or environment).
- **CIMPLE** provider can be loaded by Pegasus today.

MANAGEMENT DEVELOPERS CONFERENCE

# Developing the provider

MANAGEMENT DEVELOPERS CONFERENCE

# CIMPLE provider development

1. Define MOF class.
2. Generate class (**`genclass`**).
3. Generate provider skeleton (**`genprov`**).
4. Implement operations.
5. Pre-test provider (**`testprov`**).
6. Register provider with Pegasus.

MANAGEMENT DEVELOPERS CONFERENCE

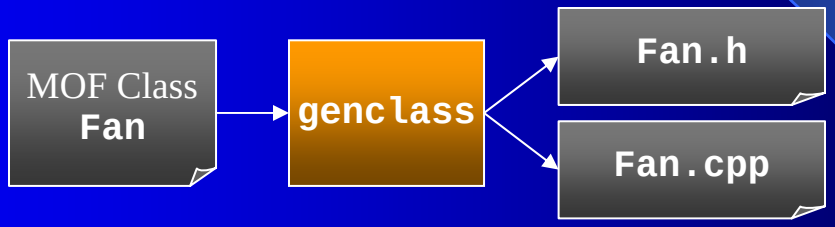
# Define the MOF class

```
// repository.mof

class Fan
{
    [key] string DeviceID;
    uint64 Speed;
    uint64 DesiredSpeed;
    uint32 SetSpeed([in] uint64 DesiredSpeed);
};
```

MANAGEMENT DEVELOPERS CONFERENCE

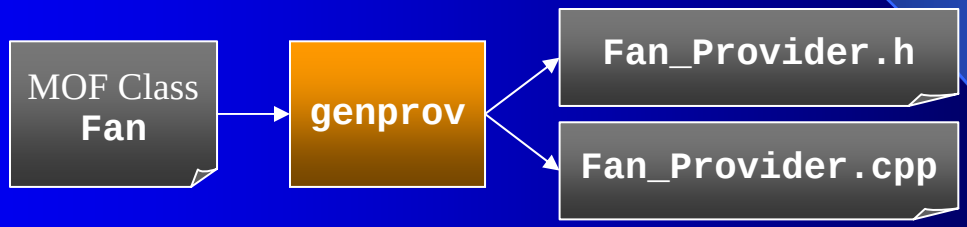
# Generate the class




```
% genclass Fan
Created Fan.h
Created Fan.cpp
% _
```

MANAGEMENT DEVELOPERS CONFERENCE

# Generate provider skeleton




```
% genprov Fan
Created Fan_Provider.h
Created Fan_Provider.cpp
% _
```

MANAGEMENT DEVELOPERS CONFERENCE

# get_instance method

```
// Fan_Provider.cpp:

Get_Instance_Status Fan_Provider::get_instance(Fan* inst)
{
    // Only one fan in this system.

    if (inst->DeviceID.value == “FAN01”)
    {
        if (!inst->Speed.null)
            inst->Speed.value = _get_fan_speed(1);

        if (!inst->DesiredSpeed.null)
            inst->DesiredSpeed.value = 0;

        return GET_INSTANCE_OK;
    }

    return GET_INSTANCE_NOT_FOUND;
}
```

MANAGEMENT DEVELOPERS CONFERENCE

# SetSpeed method

```
// Fan_Provider.cpp:

Invoke_Method_Status Fan_Provider::SetSpeed(
    const Fan* inst,
    const Value<uint64>& DesiredSpeed,
    Value<uint32>& return_value)
{
    // Only one fan in this system.

    if (inst->DeviceID == “FAN01”)
    {
        _set_desired_speed(1, DesiredSpeed);
        return_value.value = _get_fan_speed(1);
        return INVOKE_METHOD_OK;
    }

    return INVOKE_METHOD_FAILED;
}

// Note: Pegasus version of SetSpeed() is 87 lines of code.
```

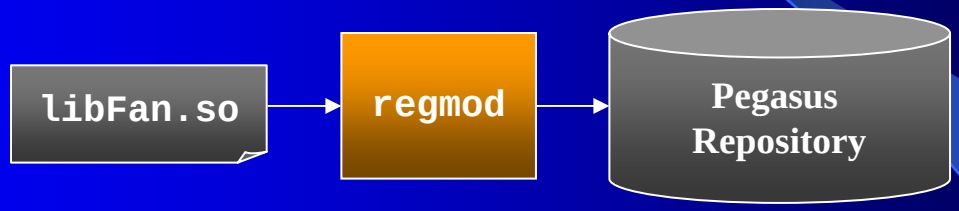
MANAGEMENT DEVELOPERS CONFERENCE

# Pre-test the provider


```
% testprov libcmplfan.so
% _
```

MANAGEMENT DEVELOPERS CONFERENCE

# Register module with Pegasus




```
% regmod libcmplfan.so
% _
```

MANAGEMENT DEVELOPERS CONFERENCE

# Provider size comparison

| | Pegasus | CIMPLE | Savings |
|---|---|---|---|
| libcmplfan.so | 26,021 bytes | 6,664 bytes | 75 percent |
| load() | 3,837 bytes | 502 bytes | 87 percent |
| get_instance() | 1,480 bytes | 242 bytes | 84 percent |
| SetSpeed() | 6,598 bytes | 72 bytes | 99 percent |

**Note**: savings increases with the complexity of the provider

MANAGEMENT DEVELOPERS CONFERENCE

# Conclusion

MANAGEMENT DEVELOPERS CONFERENCE

# Status

Complete:

- Instance providers
- Method providers
- Association providers

Work in progress:

- Indication providers
- CMPI adapter
- Query providers
- Access control

MANAGEMENT DEVELOPERS CONFERENCE

# Possible **CIMPLE** Applications

- Provider engine for **SMASH**.
- Provider engine for **WSManagement**.
- Integration with other CIMOMs.
- First-class-object-based client interface.
- In-memory repository.
- Basis for a CIMOM.

MANAGEMENT DEVELOPERS CONFERENCE

# Obtaining **CIMPLE**

- **CIMPLE** can be downloaded from **http://www.cimple.org**.

MANAGEMENT DEVELOPERS CONFERENCE

# Contact us


- **Karl Schopmeyer – k.schopmeyer@attglobal.net**
- **Mike Brasher – mike.brasher@austin.rr.com**