OpenCPI - Open Component Portability Infrastructure

An Open Source Infrastructure for Heterogeneous/Embedded Component-Based Systems and Applications

OpenCPI Application Control Interface (ACI)

Topics for This Briefing

- Introduction to the ACI
- Creating a New Application
- Simple Example
- Compiling an ACI Application
- Accessing the Data Plane
- Using Python with the ACI

Introduction to the ACI

- The Application Control Interface (ACI) is an application launch and control API for executing XML-based OpenCPI applications within a C++ or Python program
- Useful when **`ocpirun`** cannot supply the necessary programmatic and/or dynamic creation of control of an XML-based application; for example, when:
  - The contents of the application XML (OAS) need to be constructed programmatically or some of its attributes need to be dynamically set
  - The C++ main program needs to directly connect to the ports of the running application and send or receive data to/from it
  - An XML-based application needs to be run repeatedly (perhaps with configuration changes) in the same process
  - Component property values need to be read or written dynamically during application execution
- Useful for standalone applications or test fixtures

Creating a New ACI Application

- ACI applications are placed under the **`applications/`** subdirectory of an OpenCPI project
- New ACI applications are created with the **`ocpidev`** command:

  `ocpidev create application <application-name>`
- For example, for an ACI application named **`demo-aci-app`**, the command:

  `ocpidev create application demo-aci-app`

  creates:
  - **`demo-aci-app/`** - the ACI application's directory
  - **`demo-aci-app.cc`** – the main program for the ACI application
  - **`demo-aci-app.xml`** – the OAS for the ACI application
  - **`applications/application.xml`** – the top-level XML file that can be used to specify options that apply to all applications (this file is only created if it doesn't exist when the command is run)

Simple ACI Application Example

```
#include "OcpiApi.hh"
namespace OA = OCPI::API;
...
int main()
{
  try {
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    app.start();
    app.wait();
    app.finish();
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

Compiling an ACI Application

- Compiling an ACI application is similar to building other OpenCPI assets
- The **`--rcc_platform`** option to **`ocpidev build`** can be used to build for various software platforms
- From the **`demo-aci-app`** directory:

```
ocpidev build --rcc-platform centos7 \
              --rcc-platform xilinx19_2_aarch32
```

- From the project directory:

```
ocpidev build application demo-aci-app \
   --rcc-platform centos7 \
   --rcc-platform xilinx19_2_aarch32
```

Using the Control Plane

- Control plane operations in OpenCPI consist of getting and setting properties of components
- In the ACI, there are a few different options for getting and setting properties depending on performance, accuracy, and simplicity requirements
- The sample code excerpts in the next four slides show the general technique for these options (creating a runnable application is beyond the scope of this briefing)

Getting Properties – Option 1

```
...
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    std::string value;
    app.getProperty("bias", "biasValue", value);
    std::cout << value << "\n";
...
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

- Simplest interface
- All string-based, so slowest for real-time processing

Getting Properties – Option 2

```
...
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    unsigned int ivalue = app.getPropertyValue<unsigned int>("bias", "biasValue");
    std::cout << ivalue << "\n";
...
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

- Template based and simple
- Can have C++ "promoting" issues if you don't explicitly state the type using **`getPropertyValue<`*`type`*`>()`**
- Gives proper native types, which is sometimes more convenient

Getting Properties – Option 3

```
...
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    OA::Property biasValue(app, "bias.biasValue");
    std::cout << biasValue.getULongValue() << "\n";
...
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

- Fastest, lowest level, "I know exactly what to expect," with possible optimizations to direct memory-mapped access

Setting Properties

- Here are three options for setting a property that are exactly analogous to the three methods for getting a property value presented on the previous slides

```
...
    OA::Application app("myapp.xml");
    app.initialize();
    app.setProperty("bias", "biasValue", "15");

    app.setPropertyValue<unsigned int>("bias", "biasValue", 16);

    OA::Property biasValue(app, "bias.biasValue");
    biasValue.setULongValue(17);
...
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

Accessing the Data Plane

- Data can be written into or read from external ports of an OpenCPI application using the ACI's **`ExternalPort`** and **`ExternalBuffer`** classes
- Useful for interfacing with other applications external to OpenCPI that use a different protocol or transport layer
- Leaves the data formatting and opcode up to the user for increased flexibility

Accessing the Data Plane (cont.)

```
#include "OcpiApi.hh"
namespace OA = OCPI::API;
...
int main()
{
  try {
    OA::Application app("demo-aci-app.xml");
    app.initialize();
    OA::ExternalPort &toApp = app.getPort("in");
    OA::ExternalPort &fromApp = app.getPort("out");
    app.start();
    while (true) {
      ...
    }
  catch (std::string &e) {
    std::cerr << "Error: " << e << std::endl;
  }
}
```

Example of Sending Data *to* an Application

```
size_t sendBytesToApp(uint8_t *sendPtr, size_t sendSize, ExternalPort &port)
{
  uint8_t *dataPtr = NULL;
  size_t dataSizeBytes = 0;
  OA::ExternalBuffer *dataBuffer = NULL;

  // Get a buffer to fill
  do {
    dataBuffer = port.getBuffer(dataPtr, dataSizeBytes);
  } while(!dataBuffer);

  // Fill the buffer
  const size_t transferSizeBytes = std::min(sendSize, dataSizeBytes);
  memcpy(dataPtr, sendPtr, transferSizeBytes);

  // Actually push N bytes on the port with opcode 0
  dataBuffer->put(transferSizeBytes, 0);
  return transferSizeBytes; // Tell caller
}
```

Example of Getting Data *from* an Application

```
size_t getBytesFromApp(uint8_t *recvPtr, size_t recvSize, ExternalPort &port)
{
  uint8_t *dataPtr = NULL;
  size_t dataSizeBytes = 0;
  uint8_t opCode;
  bool endOfData;
  OA::ExternalBuffer *dataBuffer = NULL;

  // Get a buffer of data
  do {
    dataBuffer = port.getBuffer(dataPtr, dataSizeBytes, opCode, endOfData);
  } while(!dataBuffer);

  // Fill the out buffer with the data
  const size_t transferSizeBytes = std::min(recvSize, dataSizeBytes);
  memcpy(recvPtr, dataPtr, transferSizeBytes);

  // Release the buffer
  dataBuffer->release();
  return transferSizeBytes; // Tell caller
}
```

Using Python with the ACI

- OpenCPI provides Simplified Wrapper and Interface Generator (SWIG) bindings for the ACI, which allow users to access the ACI from within a Python interpreter
- ACI programs can be written in Python, with the ACI used directly by importing the **`opencpi.aci`** module
- Supported on development hosts only, not on any cross-compiled or embedded systems
- Mostly a 1:1 translation from C++ ACI functions to Python, with some some exceptions due to differences in language features (See the *OpenCPI Application Development Guide* for details)
- Python ACI capability is preliminary and not all ACI methods are supported

Simple Python ACI Example

```
import opencpi.aci as OA
try:
    app = OA.Application("demo-aci-app.xml")
    app.initialize()
    app.start()
    app.wait()
except Exception as e:
   print("Error: " + str(e))
```

For other examples, see the
OpenCPI Application Development Guide