A Generic Framework for Rapid Development of OPC UA Servers

P. Nikiel, B. Farnham, S. Schlenker, C.-V. Soare
CERN, Switzerland

V. Filimonov
PNPI, Russia

D. Abalo Miron
University of Oviedo, Spain

a collaboration of
Motivation: Middleware Challenges for Device Integration at LHC Detector Controls

- Scale: $10^6$ parameters, ~100 device types, >50 developers
- Standard middleware for back-end integration was OPC DA
  - Limited to Windows platform, closed source, discontinued...

1. Commonly supported COTS:
   - Power supplies, VME crates, PLCs...
   - Suppliers provide OPC DA servers

2. Custom devices:
   - Custom built electronics or front-end power supplies
   - Sub-system experts use solutions of their choice, significant effort in development and maintenance, and middleware expertise required
   - Developers have often limited software knowledge and change frequently

- Problems with stability, scalability, maintainability, diagnostics of existing systems and big effort for new systems
OPC Unified Architecture

Industrial machine-to-machine communication protocol for interoperability

- OO Information modeling capabilities
- Enhanced security, scalability
- Supports buffering, per-connection heartbeats and timeouts, discovery
- Multi-platform implementation, more lightweight ✪ embedding possible
- Commercial SDKs available with stack from OPC foundation
- Meanwhile also open source stack implementations (C, C++, Java, JS, Python)

Solves already some problems
- Still requires expertise and effort in programming with OPC UA …
OPC Unified Architecture

Industrial machine-to-machine communication protocol for interoperability

- OO Information modeling capabilities
- Enhanced security, scalability
- Supports buffering, per-connection heartbeats and timeouts, discovery
- Multi-platform implementation, more lightweight embedding possible
- Commercial SDKs available with stack from OPC foundation
- Meanwhile also open source stack implementations (C, C++, Java, JS, Python)

- Solves already some problems
- Still requires expertise and effort in programming with OPC UA …

- Maybe provide development environment and generate OPC UA related code?
Quick opcUA Server generator framework

A tool for rapid C++ server development

► Generates executable OPC UA server from target object-oriented information model

► Where does rapidity come from?
  - Automatic generation of OPC UA related source code
  - Establishing common architecture and convention
  - Provides many useful components to reduce development effort

► What does it base on?
  - OPC UA toolkit, currently Unified Automation
  - A number of open source libraries and tools

OPC UA server toolkit (C++) – Unified Automation

- XML configuration
- Security (X509 certificate handling)
- Logging
- Common namespace items and namespace utilities
- Server meta-information
- Embedded python

 OPC UA client

 OPC UA client

 OPC UA client

 Commercial toolkit

 Provided or generated components

 Device specific logic, partially generated

 100% application developer/vendor

15th ICALEPCS, 17-23th October 2015, Melbourne, Australia
Quick opcUA Server generation framework

A tool for rapid C++ server development

- Generates executable OPC UA server from target object-oriented information model
- Where does rapidity come from?
  - Automatic generation of OPC UA related source code
  - Establishing common architecture and convention
  - Provides many useful components to reduce development effort
- What does it base on?
  - OPC UA toolkit, currently Unified Automation
  - A number of open source libraries and tools

OPC UA server

OPC-UA server toolkit (C++) – Unified Automation

- XML configuration
- Security (X509 certificate handling)
- Logging
- Common namespace items and namespace utilities
- Server meta-information
- Embedded python

Commercial toolkit
Provided or generated components
Device specific logic, partially generated
100% application developer/vendor

XML config file
Hardware
Hardware
Remote process

OPC UA client
OPC UA client
OPC UA client

15th ICALEPCS, 17-23th October 2015, Melbourne, Australia
Modus Operandi

Developer benefits:

- **Design file** can be created using provided XSD schema
- Roughly 50-90% of code can be generated
- User sections of **Device Logic** stubs are well separated, merging tool simplifies re-generation after design changes or quasar upgrades
- **CMake** based build system with pre-built toolchains for several platforms
- **Configuration file** can be created using generated XSD schema

---

Get+understand model of target device/system

Fill/edit *Design File*

Generate UA address space + configuration module

(Re-)generate **Device Logic** stubs and variable handling

(Re-)implement user sections of **Device Logic**

Choose platform, build server + test binaries

Fill **Configuration File**

Test, evaluate …

Device model is OK

Device Logic is OK

Generate SCADA types, instances, UA addressing

END
## Design – Example

<table>
<thead>
<tr>
<th>DESIGN FILE</th>
<th>TEXTUAL CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;class name=&quot;PowerSupplyChannel&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;cachevariable name=&quot;current&quot; dataType=&quot;Float&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>/class&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;class name=&quot;PowerSupply&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;sourcevariable name=&quot;state&quot; dataType=&quot;Int&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;hasobjects class=&quot;PowerSupplyChannel&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>/class&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONFIGURATION FILE</th>
<th>TEXTUAL CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;PowerSupply name=&quot;powerSupply1&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;PowerSupplyChannel name=&quot;channel1&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;PowerSupplyChannel name=&quot;channel2&quot;/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>/PowerSupply&gt;</code></td>
</tr>
</tbody>
</table>

### VISUALIZATION

- **Design – Example**
  - **TEXTUAL CONTENT**:
    - `<class name="PowerSupplyChannel">`
    - `<cachevariable name="current" dataType="Float"/>`
    - `</class>`
    - `<class name="PowerSupply">`
    - `<sourcevariable name="state" dataType="Int"/>`
    - `<hasobjects class="PowerSupplyChannel"/>`
    - `</class>`
  - **Configuration File**:
    - `<PowerSupply name="powerSupply1">`
    - `<PowerSupplyChannel name="channel1"/>`
    - `<PowerSupplyChannel name="channel2"/>`
    - `</PowerSupply>`

- **Visualization**
  - **runtime OPC UA client**
  - **quasar-generated diagram**
Design – Example

Schema-aware XML editor (Eclipse plugin)

**DESIGN FILE**

```
<?xml version="1.0" encoding="UTF-8"?>
<design
  class_root
  xmlns:projectShortName="PowerSupply"
  xmlns:id="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/Design Design.xsd"
  class="PowerSupply"
    name="powerSupply1"
      channel1"
      channel2"
</PowerSupply>
```

**CONFIGURATION FILE**

```
<ds:design
  objects="PowerSupplyChannel"
    name="current"
      dataType="Float"
    
    name="state"
      dataType="Int"
    
    class="PowerSupplyChannel"
      instanceUsing="configuration"
    
    class="PowerSupplyChannel"
      addressSpaceWrite="forbidden"
    
    name="state"
      addressSpaceRead="asynchronous"
    
    addressSpaceReadUseMutex="asynchronous"
    
    d:root
      synchronous
```

**VISUALIZATION**

Quasar-generated diagram

**Runtime OPC UA client**

Design – Example
Components & Tools

Embedded python
Use python scripts in device logic. User writes in safe language.

Variable-based scripts for processing in in/out direction.

Global scripts with address space access.

XML configuration

Generated schema, simple creation.

Validation tool, verify design constraints.

Generated loader for object instantiation and runtime access to configuration.

Logging
Provides API and exchangeable back-end.

Component based.

Tools

Design visualization: UML generator

Platform toolchains: Linux x86_64, i686, ARM (Raspbian), ARM (Zynq), Windows 32/64

Easy RPM generator

Generated program to test full address space.

Documentation: doxygen

Software management: consistency checker helps using versioning system.

Protocol components

CAN devices and interfaces

SNMP module

IPv6 module

Server meta-information
# Items, memory usage, thread pool size, run time …

More to come…
Components & Tools

Embedded python

- Use python scripts in device logic
- User writes in **safe language**
- Variable-based scripts for processing in in/out direction
- Global scripts with **address space access**

### XML configuration
- Generated schema → simple creation
- Validation tool → verify design constraints
- Generated loader for object instantiation and runtime access to configuration

### Logging
- Provides **API** and exchangeable **back-end**
- Component based

### Protocol components
- **CAN** devices and interfaces
- **SNMP** module
- **IPbus** module

More to come…
State and Usage

Quasar v1.0
- Available for collaborators via SVN
- Documentation: inline documentation and video tutorials
- Export to GitHub in progress (free open source license)

Collaboration with equipment vendors
- Several vendors interested on using quasar for their hardware in collaboration with CERN experts
- Should facilitate problem diagnostics and maintenance

Quasar-made servers
- Three servers in production in ATLAS experiment controls
- >5 in test stage or development, to be used for new projects or replacing deprecated OPC DA solutions
- Several users across CERN, provided positive feedback

CANopen via CAN
IPbus via TCP/IP
SNMP via TCP/IP
VME crates via CAN
FPGA board via CAN
S7 TSPP PLC via TCP/IP
CAEN HV power supplies via TCP/IP
Iseg HV power supplies via TCP/IP
Rad-hard ASIC monitoring via optical link
FPGA (Zynq) via TCP/IP
HV-Micro via CAN
Conclusions

► generates OPC UA servers from information model

► Development and maintenance effort greatly reduced due to:
  ● **Coherency**: design file as single point of input
  ● **Knowledge requirements** on OPC UA layers or SDKs minimal
  ● **Programming reduced to device logic** in C++, python
  ● Lots of pluggable components
  ● Multiple platforms supported out-of-the-box
  ● Higher controls layer integration facilitated

► External equipment suppliers are willing to use it

⇔ Looks promising that we can meet the middleware challenges!
Conclusions

► generates OPC UA servers from information model

► Development and maintenance effort greatly reduced due to:
  ● Coherency: design file as single point of input
  ● Knowledge requirements on OPC UA layers or SDKs minimal
  ● Programming reduced to device logic in C++, python
  ● Lots of pluggable components
  ● Multiple platforms supported out-of-the-box
  ● Higher controls layer integration facilitated

► External equipment suppliers are willing to use it

♫ Looks promising that we can meet the middleware challenges!
Transforming Information Model

Model condensed into *Design File* using OO approach

- **Classes, relations** between classes
- **Variables** which belong to classes, main types
  - *Cache* variables: in-memory data access
  - *Source* variables: asynchronous and synchronous device access
- Various class and variable **attributes+properties** such as data type, read-only or writable, ...

**Code and schema generation**

- Based on XSLT transforms
Internal handling of variables (generated) – Sequence diagrams

Hardware

Device

Device updates data

Device Logic Object

handleUpdate()

Device-specific message or function call

SourceVariable

Device

device replies to the request

Device Logic Object

handleUpdate()

Device specific reply message Or asynchronous function call

Read():

Prepare C.V.
Send request

Wait on C.V.

C.V. is notified

Generated AddressSpace

Address Space Object

setSomeValue()

update values

SourceVariable

IO Manager

beginRead()

New IO Job (separate thread)

finishRead()

value sent back

Address Space

read value

Internal handling of variables (generated) – Sequence diagrams

15th ICALEPCS, 17-23th October 2015, Melbourne, Australia

Stefan Schlenker, CERN