CHEP 2015
21st International Conference on Computing in High Energy and Nuclear Physics
April 13-17, 2015, Okinawa, Japan

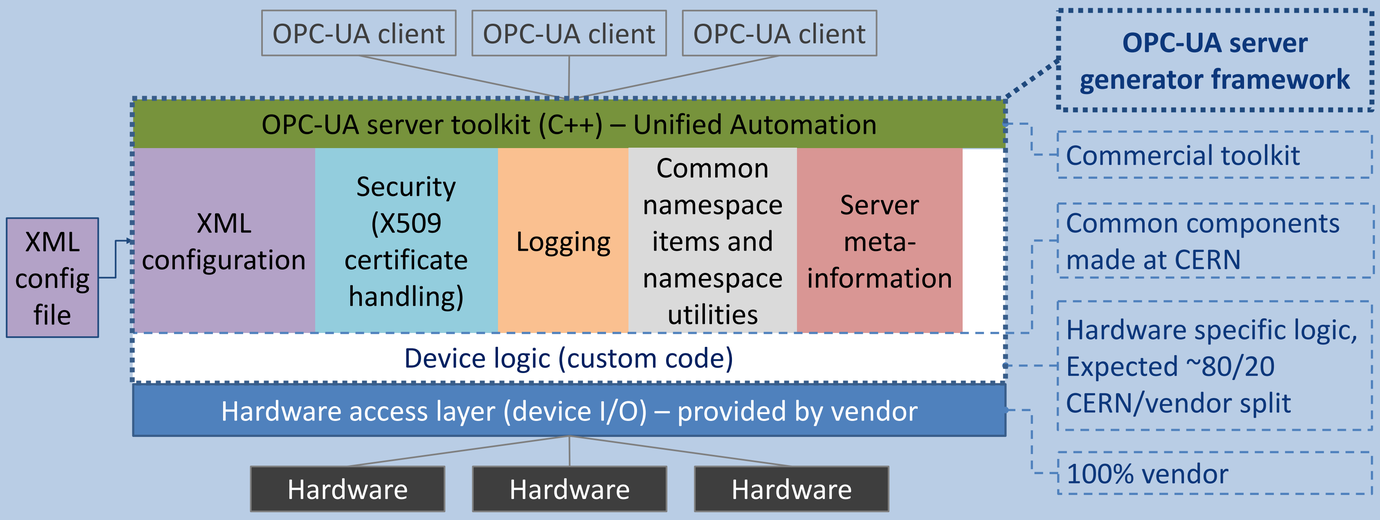
A Generic Framework for Rapid Development of OPC UA Servers

Piotr Nikiel, Ben Farnham, Stefan Schlenker (CERN, Geneva, Switzerland)
Viatcheslav Filimonov (PNPI, Gatchina, Leningrad District, Russia)

Abstract

This paper describes a new approach for generic design and efficient development of OPC UA servers. Development starts with creation of a design file, in XML format, describing an object-oriented information model of the target system or device. Using this model, the framework generates an executable OPC UA server application, which exposes the per-design OPC UA address space, without the developer writing a single line of code. Furthermore, the framework generates skeleton code into which the developer adds the necessary logic for integration to the target system or device.

The big picture of OPC UA based communication



- To monitor and control hardware via OPC-UA requires an OPC-UA server; the server provides a software proxy for the hardware which is compatible with any standard OPC-UA client.
- Internally to the server, the hardware is often represented by a hardware access layer; a device specific software component with a proprietary (i.e. non-standard) interface.
- The OPC-UA server publishes an interface (the *address space*), in standard OPC-UA vocabulary, this represents the functionality of the underlying hardware.
- The approach is platform-independent; this project supports Windows and Linux for X86 and X86-64 CPUs and Linux on ARM CPUs

Describing the structure and object layout of the server using design file

Example **design file:**

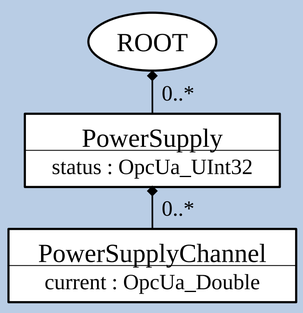
```
<class name="PowerSupplyChannel">
  <cachevariable name="current" dataType="Float"/>
</class>

<class name="PowerSupply">
  <sourcevariable name="state" dataType="Int"/>
  <hasobjects class="PowerSupplyChannel"/>
</class>
```

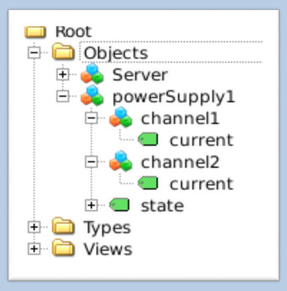
Example **config file:** (corresponding to design above)

```
<PowerSupply name="powerSupply1">
  <PowerSupplyChannel name="channel1">
  <PowerSupplyChannel name="channel2"/>
</PowerSupply>
```

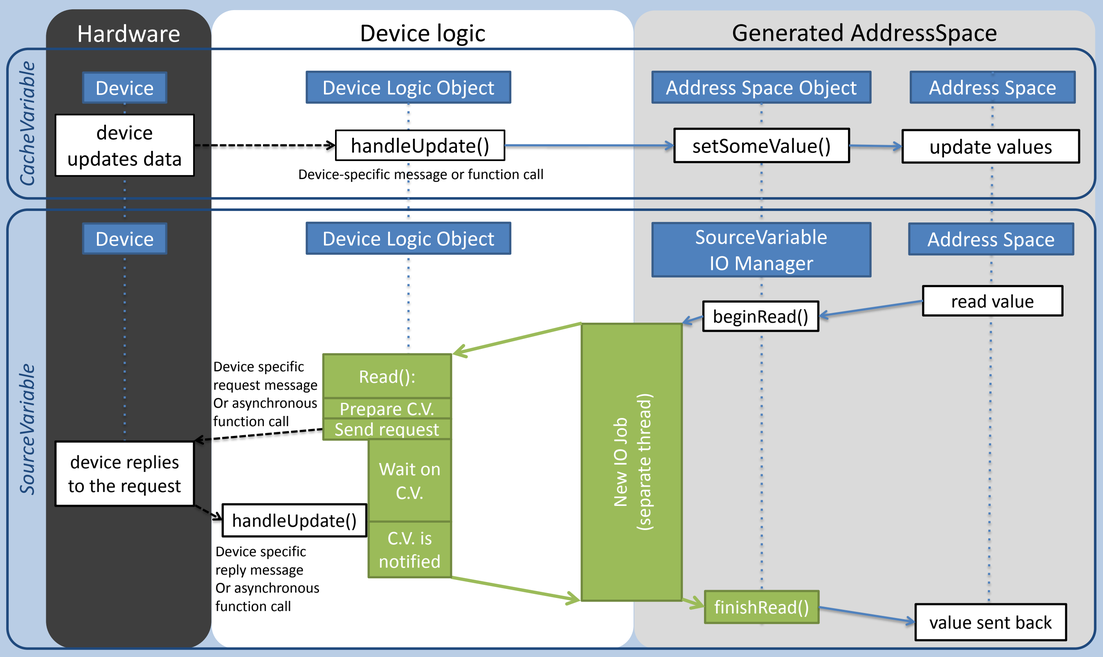
UML visualization diagram (auto-generated)



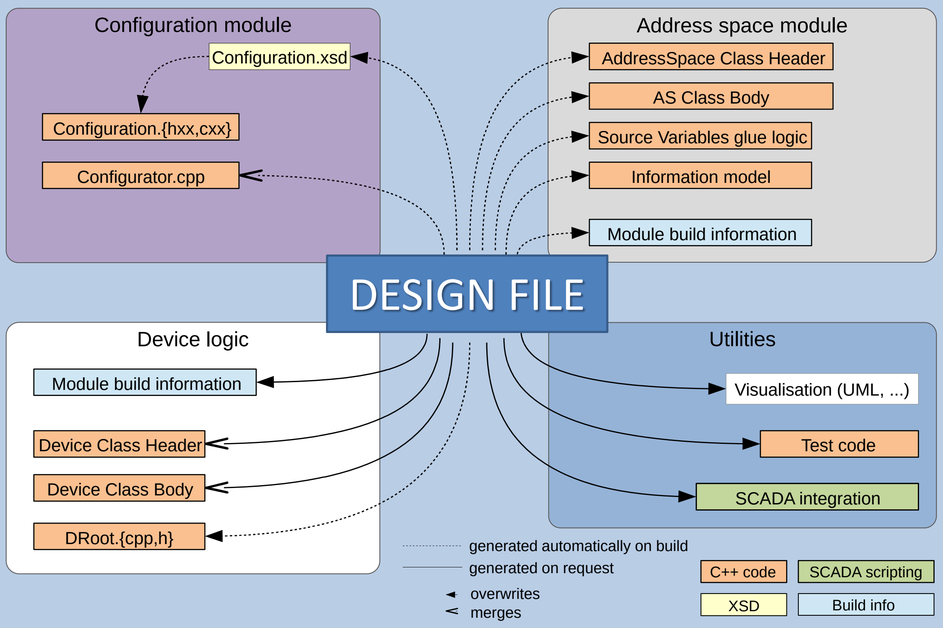
OPC UA Client view (here: UaExpert)



Internal handling of variables (auto-generated) Sequence diagrams



Generation of majority of C++ source code, configuration schema, integration and development tools



- Only custom user code has to be written by a developer
- Whole Address Space module is 100% automatically generated
- Stubs of Device Logic source code are generated, enabling developers to provide factual implementation without writing "a skeleton"
- Configuration module is 100% generated including configuration file schema and configuration file loader and validator
- Generated code, generated stubs as well as overall architecture promotes creation of robust software (type safety, adherence to standards, appropriate error handling) as well as well-performing software (multi-threaded processing is natively supported, see above)
- Technologies used: XML, XSLT, Python, shell scripts

Additional features

- Open source: the Framework is under GPL license (though requiring non-open source UA toolkit)
- Build system: A robust build system (based on Cmake) is included. It is integrated with design file handling to automate development tasks.
- SCADA Integration: The framework generates OPC UA bindings to Siemens WinCC OA SCADA.
- Software management tools: The framework is capable of maintaining consistency of the source code, raising alerts when e.g. custom code was forgotten to be added into version control system.
- Target deployment: The framework seamlessly integrates with RPM builder (for Linux).

Conclusions

The approach proved to be successful for many OPC UA servers created for ATLAS DCS (already in production) and beyond. It has been chosen to integrate 3rd party products into various control systems at CERN in cooperation with the suppliers. In this approach, design and development efforts are hugely reduced, and the percentage of generated code can reach up to 90%. Applying a common approach also helped to easily delegate development tasks to growing group of developers and to reduce maintenance costs.