2015 ICALEPCS
melbourne • australia

A Generic Framework for Rapid Development of OPC UA Servers

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

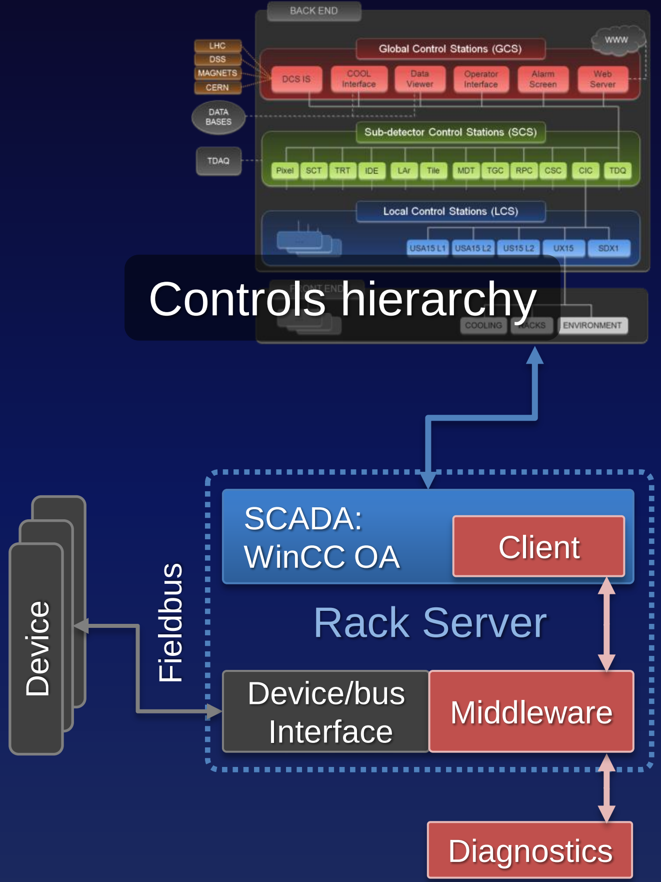
V. Filimonov
PNPI, Russia

D. Abalo Miron
University of Oviedo, Spain

a collaboration of ATLAS EXPERIMENT & CERN EN ENGINEERING DEPARTMENT

Motivation: Middleware Challenges

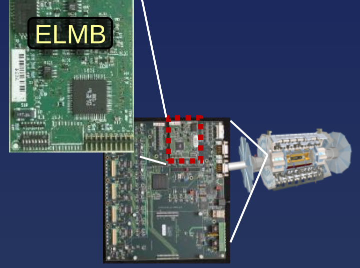
for Device Integration at LHC Detector Controls


Controls hierarchy

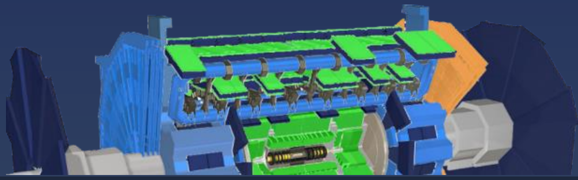
- ► Scale: 10^6 parameters, ~100 device types, >50 developers
- ► Standard middleware for back-end integration was OPC DA
- ➲ Limited to Windows platform, closed source, discontinued…

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



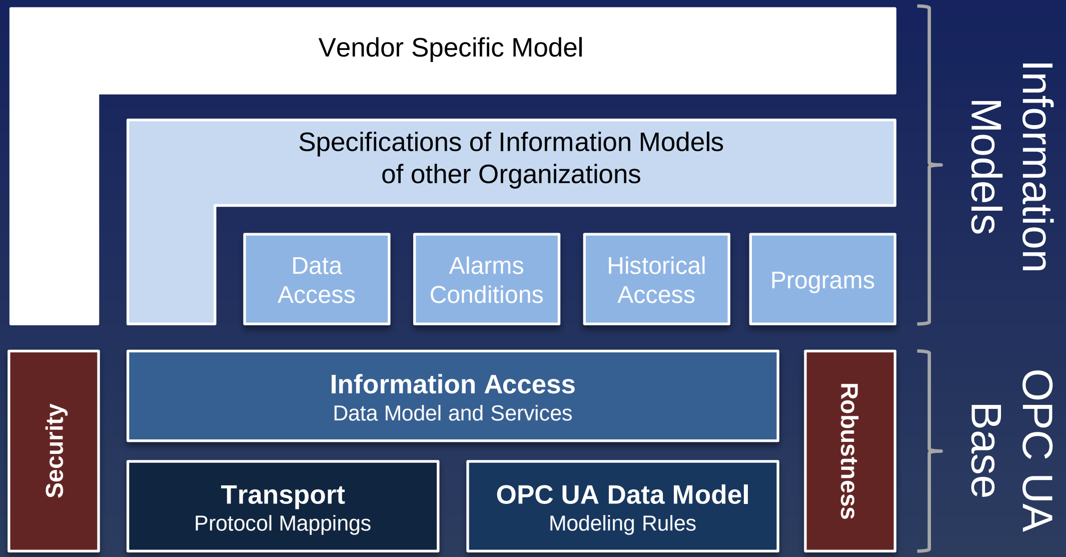
➲ Problems with stability, scalability, maintainability, diagnostics of existing systems and big effort for new systems



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)



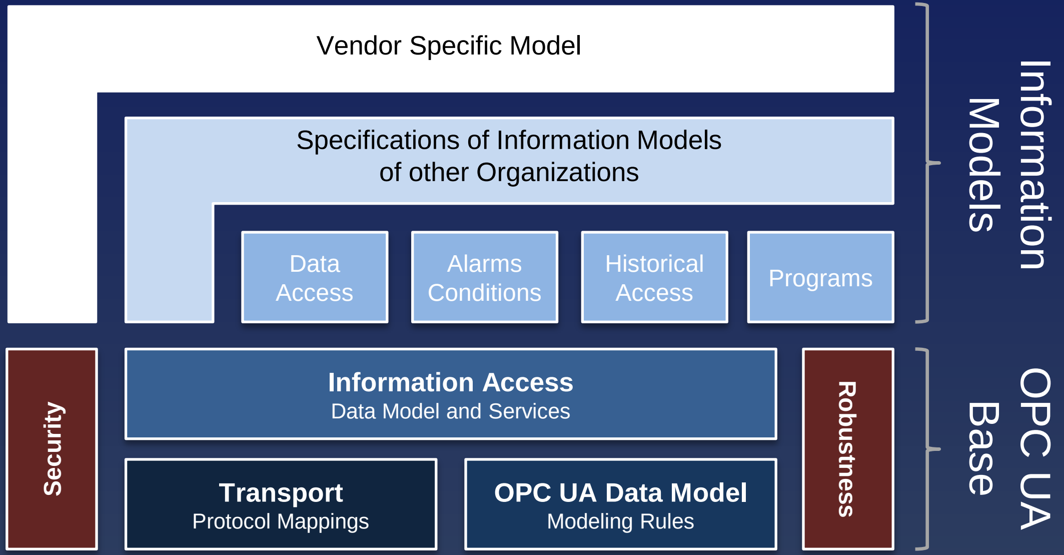
➲ Solves already some problems

▶ 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

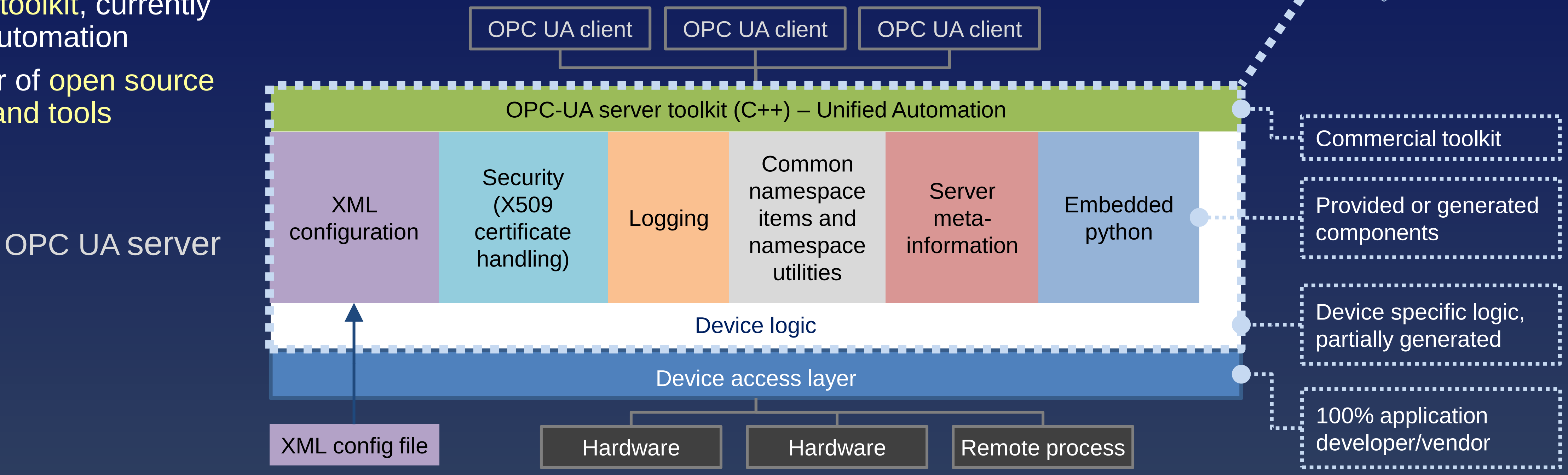
▶ Still requires expertise and effort in programming with OPC UA …

➲ Maybe provide development environment and generate OPC UA related code?

Quick opcUA Server 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 client

OPC UA client

OPC UA client

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

Device logic

Device access layer

XML config file

Hardware

Hardware

Remote process

Commercial toolkit

Provided or generated components

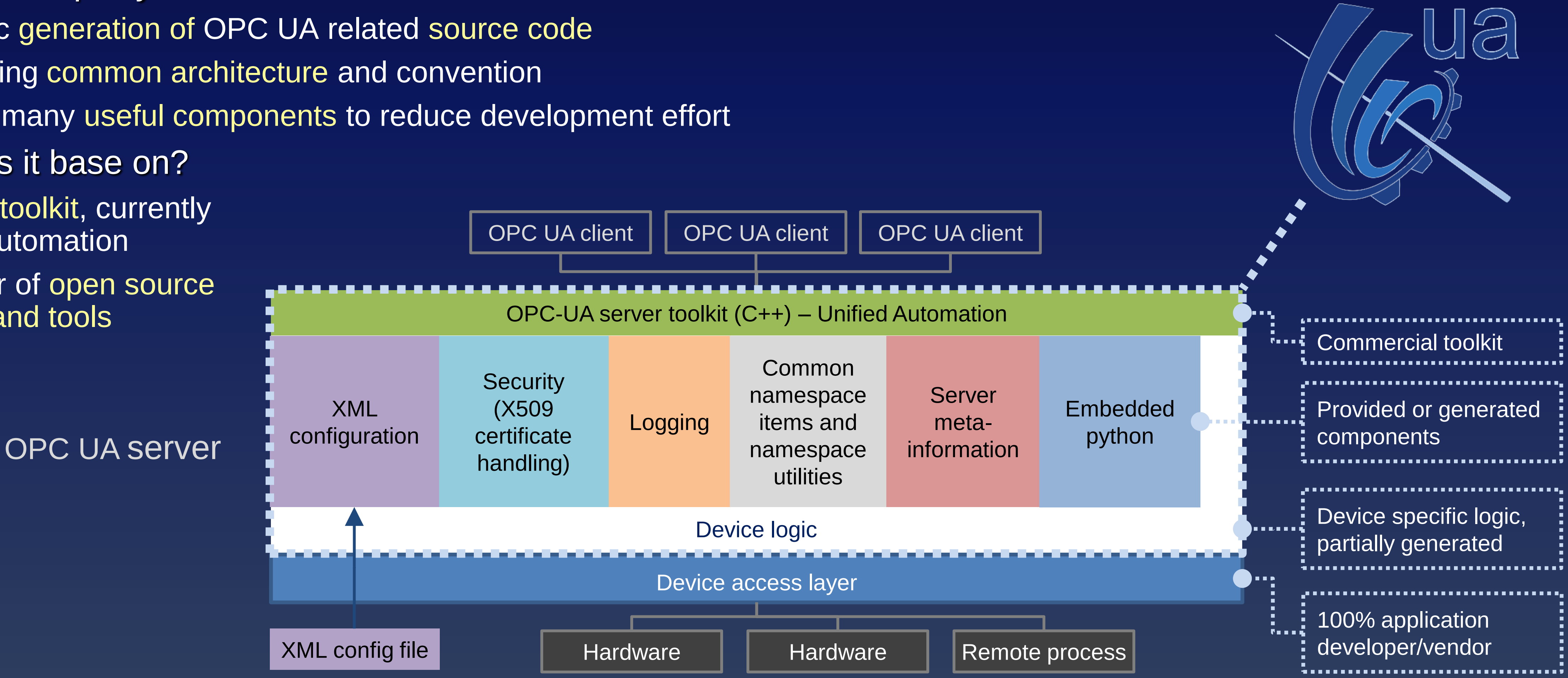
Device specific logic, partially generated

100% application developer/vendor

Quick opcUA Server generAtion fRamework

A tool for rapid C++ server development

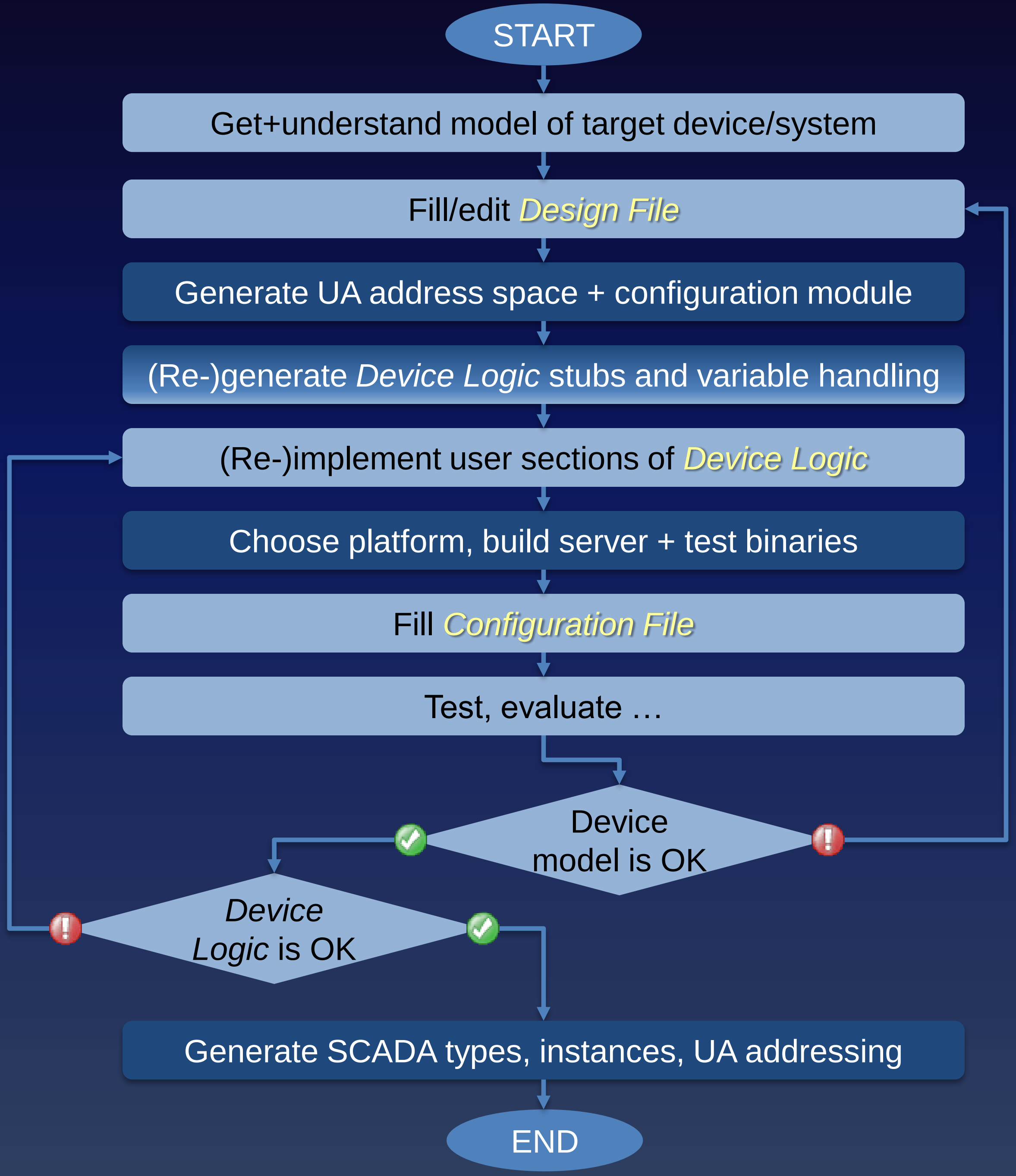
- Generates executable OPC UA server from target object-oriented information model
- Where does rapidity come from?
  - Automatic generation of OPC UA related source code
  - Establishing common architecture and convention
  - Provides many useful components to reduce development effort
- What does it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


OPC UA client
OPC UA client
OPC UA client
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
Device logic
Device access layer
XML config file
Hardware
Hardware
Remote process
Commercial toolkit
Provided or generated components
Device specific logic, partially generated
100% application developer/vendor
ua

Modus Operandi

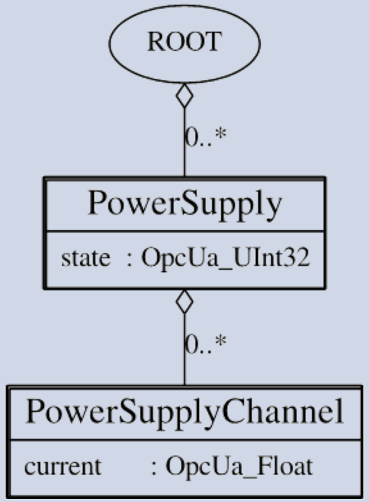
Developer benefits:

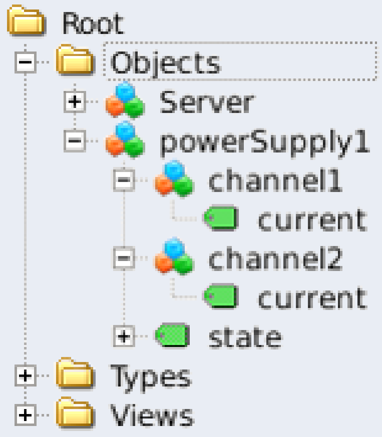
- *Design file* can be created using provided XSD schema
- Roughly 50-90% of code can be generated
- User sections of *Device Logic* stubs are well separated, merging tool simplifies re-generation after design changes or quasar upgrades
- CMake based build system with pre-built toolchains for several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


START
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

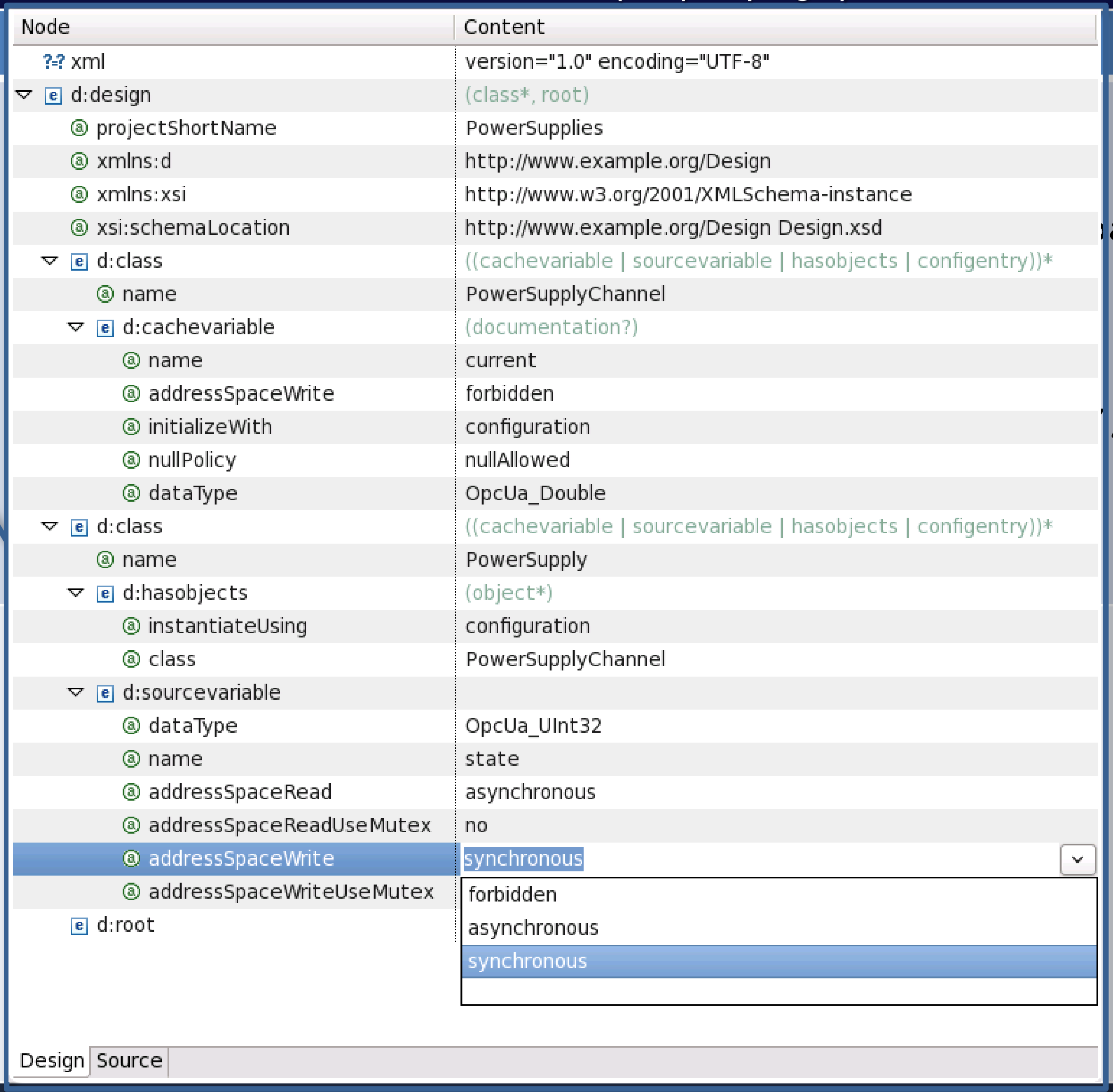
	TEXTUAL CONTENT	VISUALIZATION

DESIGN FILE	`<class name=“PowerSupplyChannel”>`
	`   <cachevariable name=“current” dataType=“Float”/>`
	`</class>`
	
	`<class name=“PowerSupply”>`
	`   <sourcevariable name=“state” dataType=“Int”/>`
	`   <hasobjects class=“PowerSupplyChannel”/>`
	`</class>`
		 ROOT
		0..*
		PowerSupply
		state : OpcUa_UInt32
		0..*
		PowerSupplyChannel
		current : OpcUa_Float *quasar-generated diagram*

CONFIGURATION FILE	`<PowerSupply name=“powerSupply1”>`
	`   <PowerSupplyChannel name=“channel1”/>`
	`   <PowerSupplyChannel name=“channel2”/>`
	`</PowerSupply>`
		 Root
		Objects
		Server
		powerSupply1
		channel1
		current
		channel2
		current
		state
		Types
		Views *runtime OPC UA client*

Design – Example

Schema-aware XML editor (Eclipse plugin)



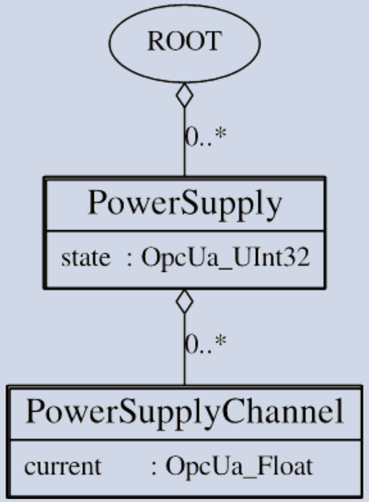
Node	Content
?-? xml	version="1.0" encoding="UTF-8"
d:design	(class*, root)
projectShortName	PowerSupplies
xmlns:d	http://www.example.org/Design
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	http://www.example.org/Design Design.xsd
d:class	((cachevariable | sourcevariable | hasobjects | configentry))*
name	PowerSupplyChannel
d:cachevariable	(documentation?)
name	current
addressSpaceWrite	forbidden
initializeWith	configuration
nullPolicy	nullAllowed
dataType	OpcUa_Double
d:class	((cachevariable | sourcevariable | hasobjects | configentry))*
name	PowerSupply
d:hasobjects	(object*)
instantiateUsing	configuration
class	PowerSupplyChannel
d:sourcevariable	
dataType	OpcUa_UInt32
name	state
addressSpaceRead	asynchronous
addressSpaceReadUseMutex	no
addressSpaceWrite	synchronous
addressSpaceWriteUseMutex	
d:root	

VISUALIZATION

DESIGN FILE

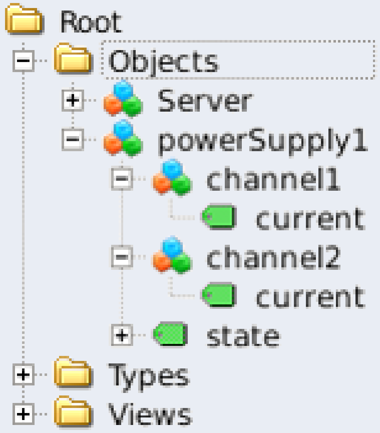
at"/>

/>



quasar-generated diagram

CONFIGURATION FILE



runtime OPC UA client

Components & Tools

XML configuration

Generated schema ➲ simple creation

Validation tool ➲ verify design constraints

Generated loader for object instantiation and runtime access to configuration

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**

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

IPbus module

Server meta-information

# Items, memory usage, thread pool size, run time …

More to come…

Components & Tools

XML configuration

Generated schema ➲ simple creation

Validation tool ➲ verify design constraints

Generated loader for object instantiation and runtime access to configuration

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**

Logging

Provides **API** and **exchangeable back-end**

Component based

Protocol components

CAN devices and interfaces

SNMP module

IPbus module

S

# Items, me

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*

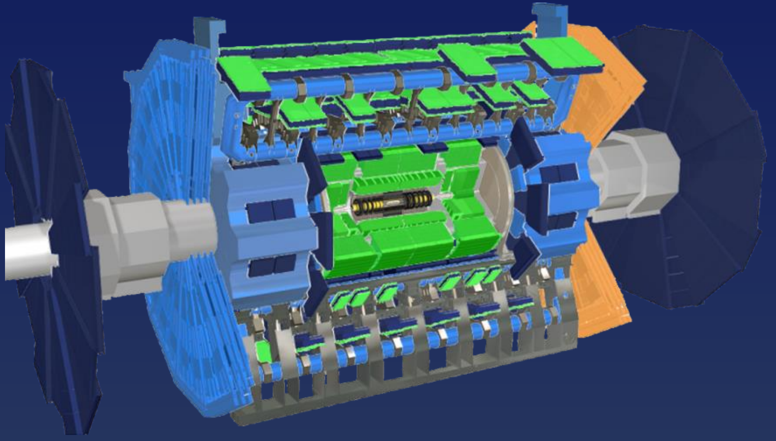
Rad-hard ASIC monitoring *via optical link*

FPGA (Zynq) *via TCP/IP*

HV-Micro *via CAN*

Conclusions

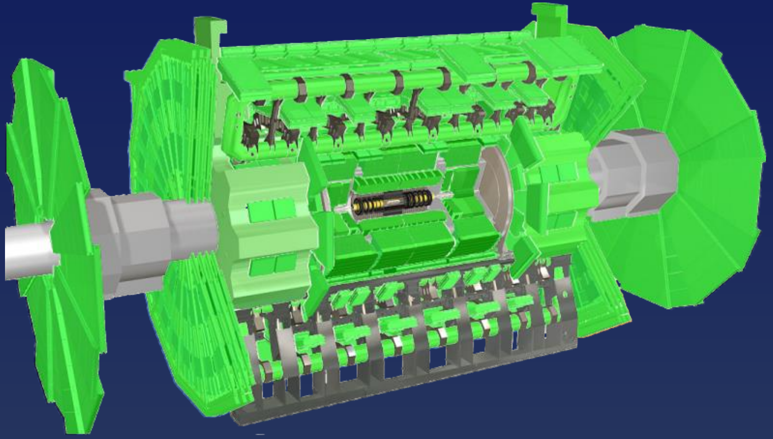
- quasar generates OPC UA servers from information model
- Development and maintenance effort greatly reduced due to:
  - Coherency: design file as single point of input
  - Knowledge requirements on OPC UA layers or SDKs minimal
  - Programming reduced to device logic in C++, python
  - Lots of pluggable 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

- quasar 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!

BACKUP

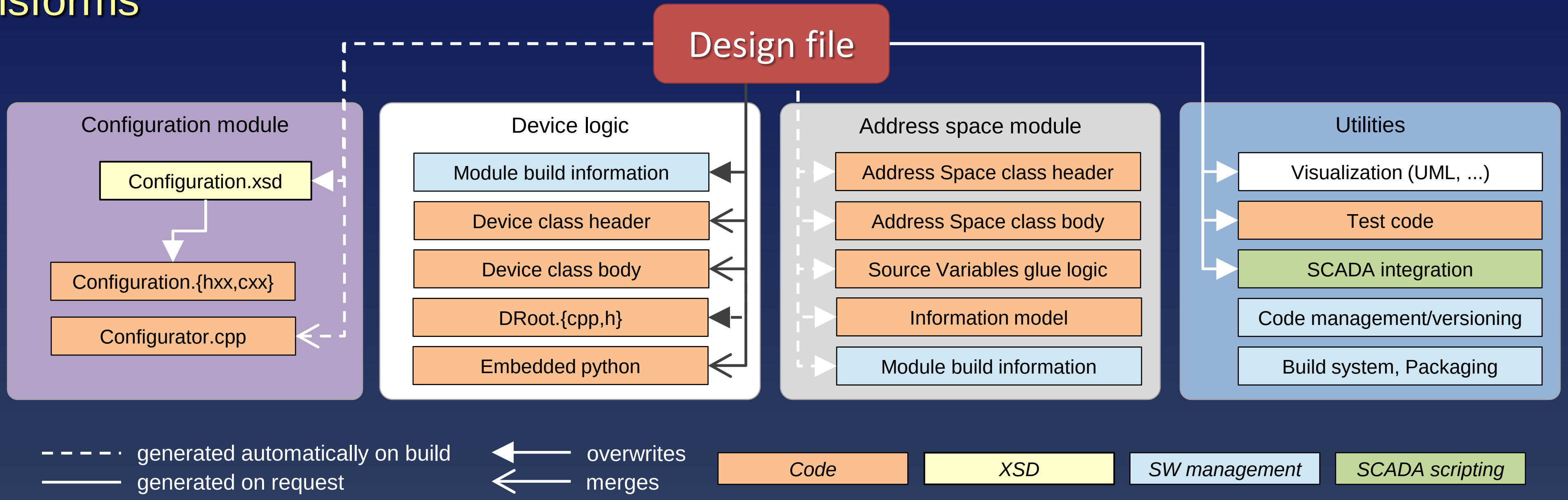
Transforming Information Model

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



Design file

Configuration module
- Configuration.xsd
- Configuration.{hxx,cxx}
- Configurator.cpp

Device logic
- Module build information
- Device class header
- Device class body
- DRoot.{cpp,h}
- Embedded python

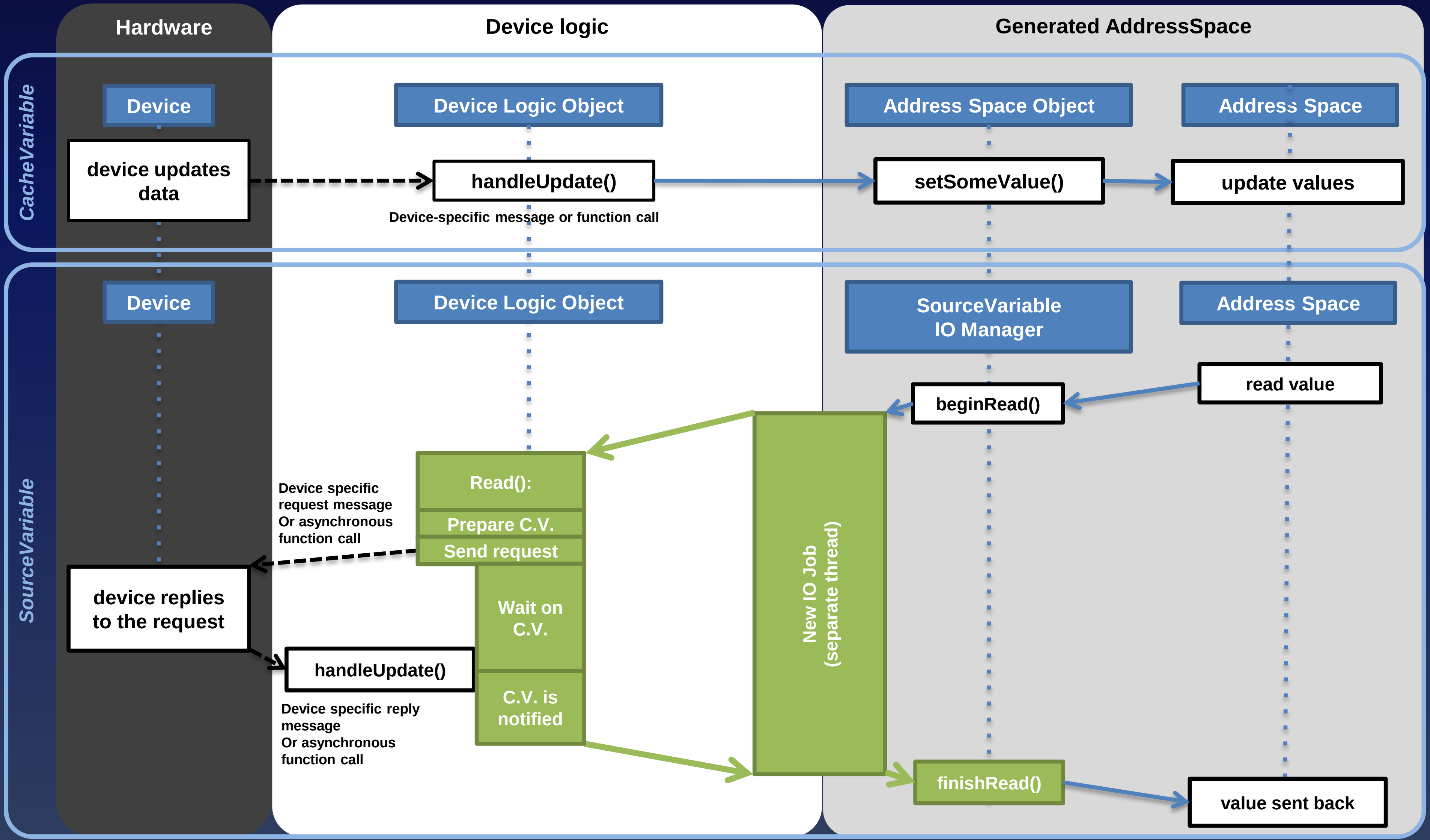
Address space module
- Address Space class header
- Address Space class body
- Source Variables glue logic
- Information model
- Module build information

Utilities
- Visualization (UML, ...)
- Test code
- SCADA integration
- Code management/versioning
- Build system, Packaging

- - - - generated automatically on build
—— generated on request
◀— overwrites
<— merges

Code | XSD | SW management | SCADA scripting

Internal handling of variables (generated) – Sequence diagrams


Hardware
Device logic
Generated AddressSpace
CacheVariable
Device
Device Logic Object
Address Space Object
Address Space
device updates data
handleUpdate()
setSomeValue()
update values
Device-specific message or function call
SourceVariable
Device
Device Logic Object
SourceVariable IO Manager
Address Space
read value
beginRead()
New IO Job (separate thread)
Read():
Prepare C.V.
Send request
Device specific request message Or asynchronous function call
Wait on C.V.
device replies to the request
handleUpdate()
C.V. is notified
Device specific reply message Or asynchronous function call
finishRead()
value sent back