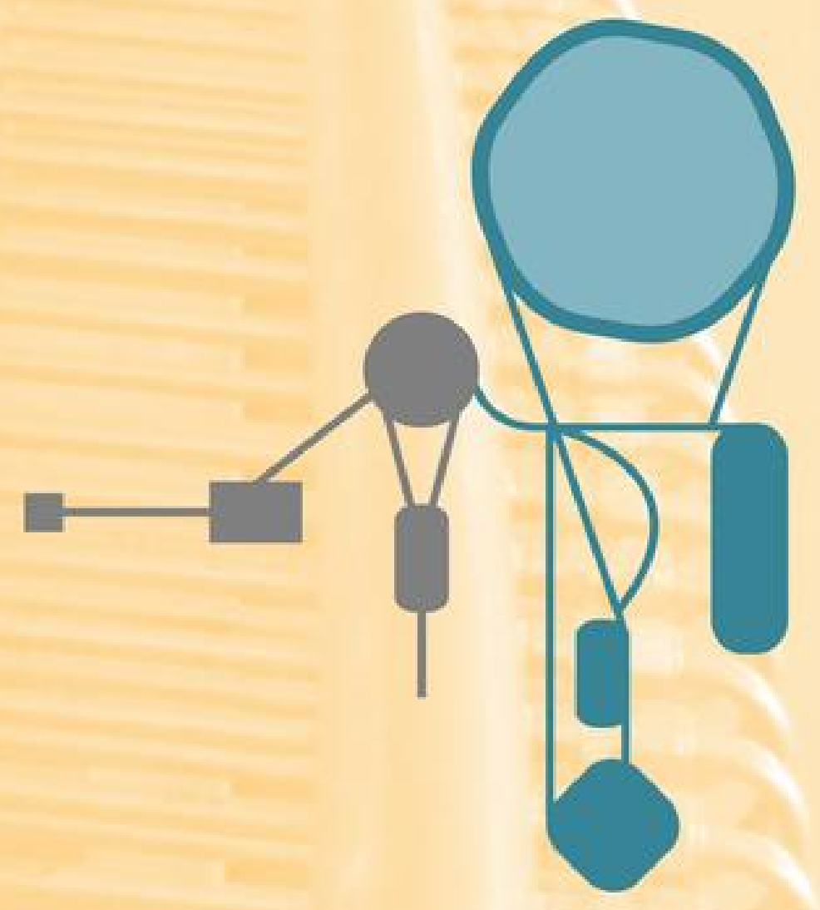
Extendend Application Fields for the Renovated GSI Control System

K. Höppner, L. Hechler, K. Herlo, P. Kainberger, U. Krause, S. Matthies

GSI, Darmstadt, Germany



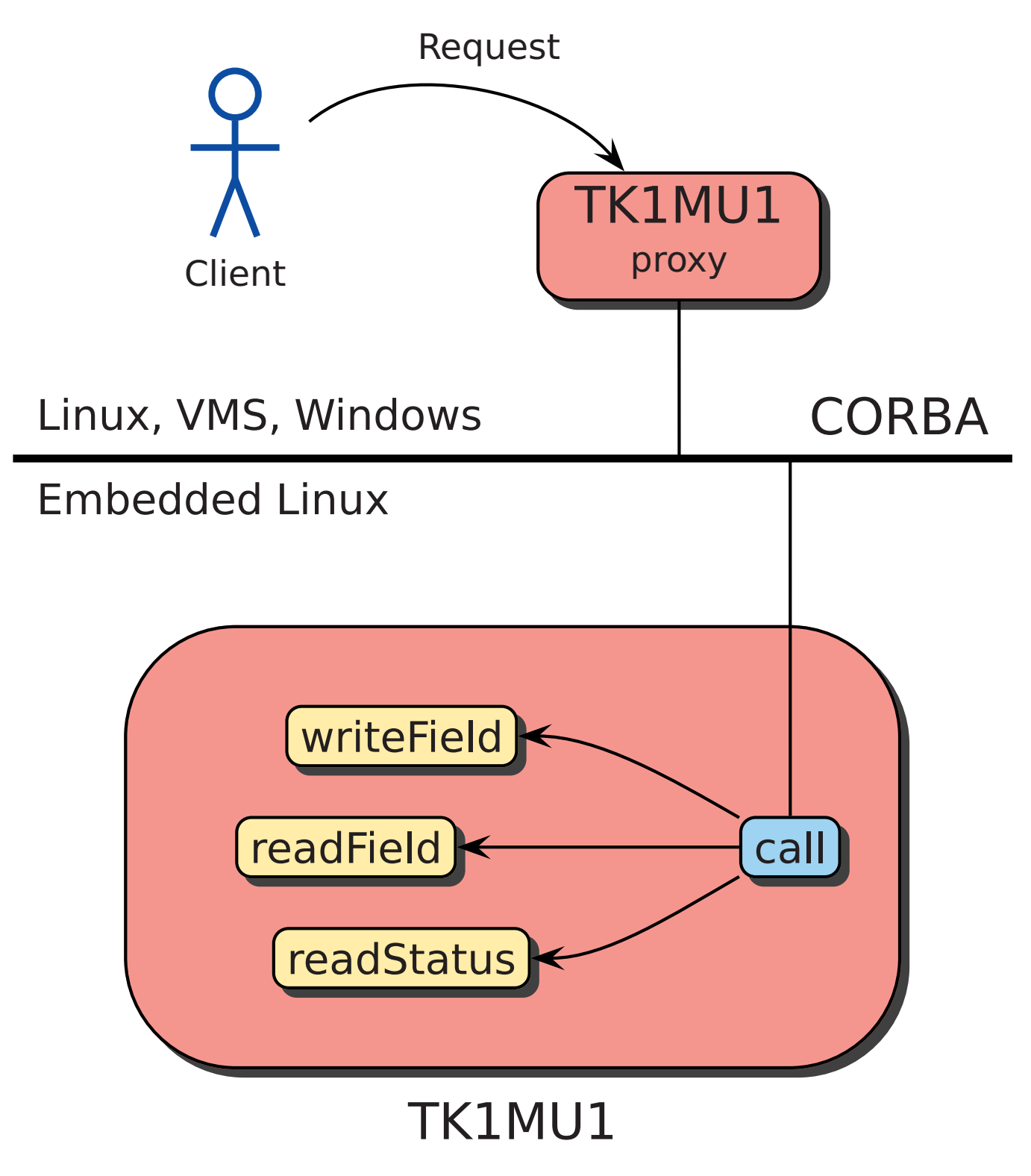
FAIR

Abstract

The current GSI control system uses a very monolithic approach that made it difficult to extend the system to other than the original platforms (VME front ends and OpenVMS on the application level). For the present renovation project of the communication layers, flexibility was a major design criterion. Front-end and application levels are connected via CORBA middleware, giving free choice for using various system architectures and programming languages on both levels. While most of the current front-end software will be ported to the existing VME front-end environment, now running Linux, the new system can integrate devices running on various architectures and operating systems into the new GSI control system. To model equipment functionality as independently as possible, generating adapter code from a well-defined XML description of device models is now under development. This will make the task of porting the existing 65 device models (including around 3000 properties) to the new modular approach easier.

Migration Status

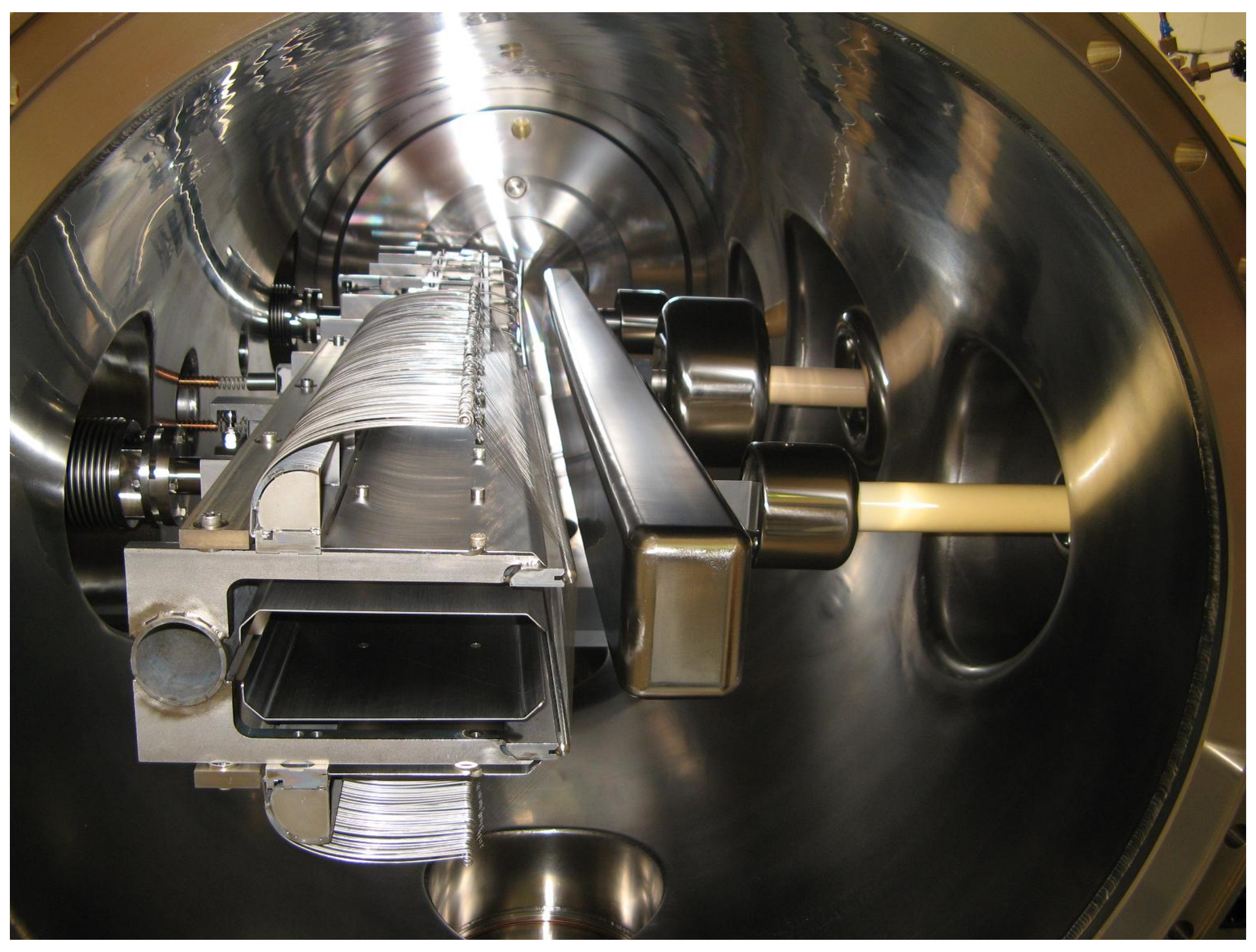
- ☛ Present GSI control system uses VME-bus controllers on front-end level: M680*xx* based realtime equipment controllers (EC) and non-realtime device presentation or supervising controllers (DPC/SC), communication with operating level via in-house network protocols
- ✔ In progress: replacement of supervising controllers with PowerPC based VME boards, running embedded Linux.
- ✔ First use in real life accelerator controls in summer 2007
- ✔ Replacement of middleware level with CORBA
- ✔ New object oriented property classes, but reuse of present code (user support routines—USR)
- ✔ Extending the scope of the control system to non-VME based device servers
- ✔ Device server API in C++
- ✔ Client API in C++, Java, and Python



Device Access in the new control system via CORBA; reuse of existing USR code

Septum Motor Control: M-Box

- ☛ New injection septum in the heavy ion synchrotron (SIS) to be installed November 2007
- ☛ Simultaneous movement of motor axes required due to limited mechanical distortion of anode and cathode.
- ✔ Moving anode and cathode with Cosylab's M-Box, integrating a PMAC (programmable multi axis controller) motor control and a MicroIOC, an embedded system based on Intel CPU running Debian Linux
- ✔ Device Manager on x86 platform, in the past used for testing and simulation, is brought into real life accelerator controls.



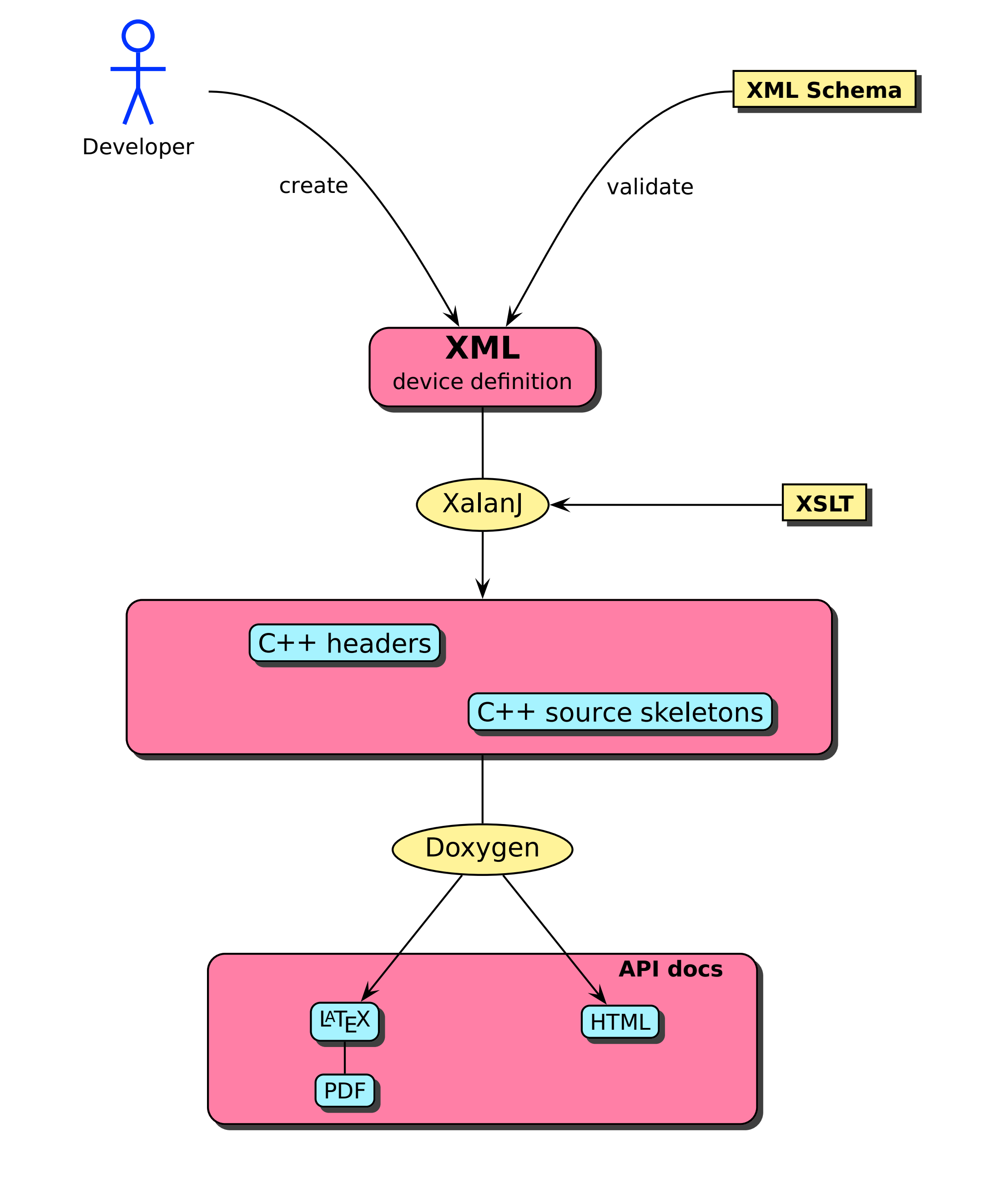
Photograph of the new septum

Automatic Code Generation

- ☛ GSI control system supports approximately 65 equipment models, implementing 3000 properties.
- ☛ While major part of USR code may be reused, the wrapper classes for the object oriented representation of equipment models and their properties have to be implemented for the migration to the new control system.
- ✔ We invented a code generator based on a XML presentation of equipment models
- ✔ Code generator provides support for the pulse-to-pulse mode of GSI accelerators and the needs for restricting access to device properties during cancer therapy operation.
- ✔ Created files:
  - ● C++ header files for device properties and their actions (read, write, call)
  - ● C++ USR code skeletons for property actions, to be filled with existing USR code
  - ● In-code API documentation to be processed with Doxygen, generating HTML and LaTeX/PDF format.
- ✔ Code generation based on a XSLT stylesheet processed with XalanJ, providing a flexible, platform independent Java based solution.

Example of a XML device definition

```xml
<eqmod>
    <name>MX</name>
    <creator>KlausHoeppner</creator>
    <version>09.01.01</version>
    <description>Multiplexed Dipole
                   Magnet</description>
    <variant id="1">PERMANENT_SIS</variant>
    <variant id="2">SHARED_SIS</variant>
    <variant id="3">PERMANENT_UNI</variant>
    <property category="master">
        <name>INIT</name>
        <description>Initialize</description>
        <action type="call" medlock="all"/>
    </property>
    <property category="slave">
        <name>CURRENTS</name>
        <description>Current Set
                  Value</description>
        <action type="read" medlock="none"/>
        <action type="write" medlock="vrtacc"/>
        <data type="Float32">
            <value name="current">Current (Set)</value>
        </data>
    </property>
</eqmod>
```

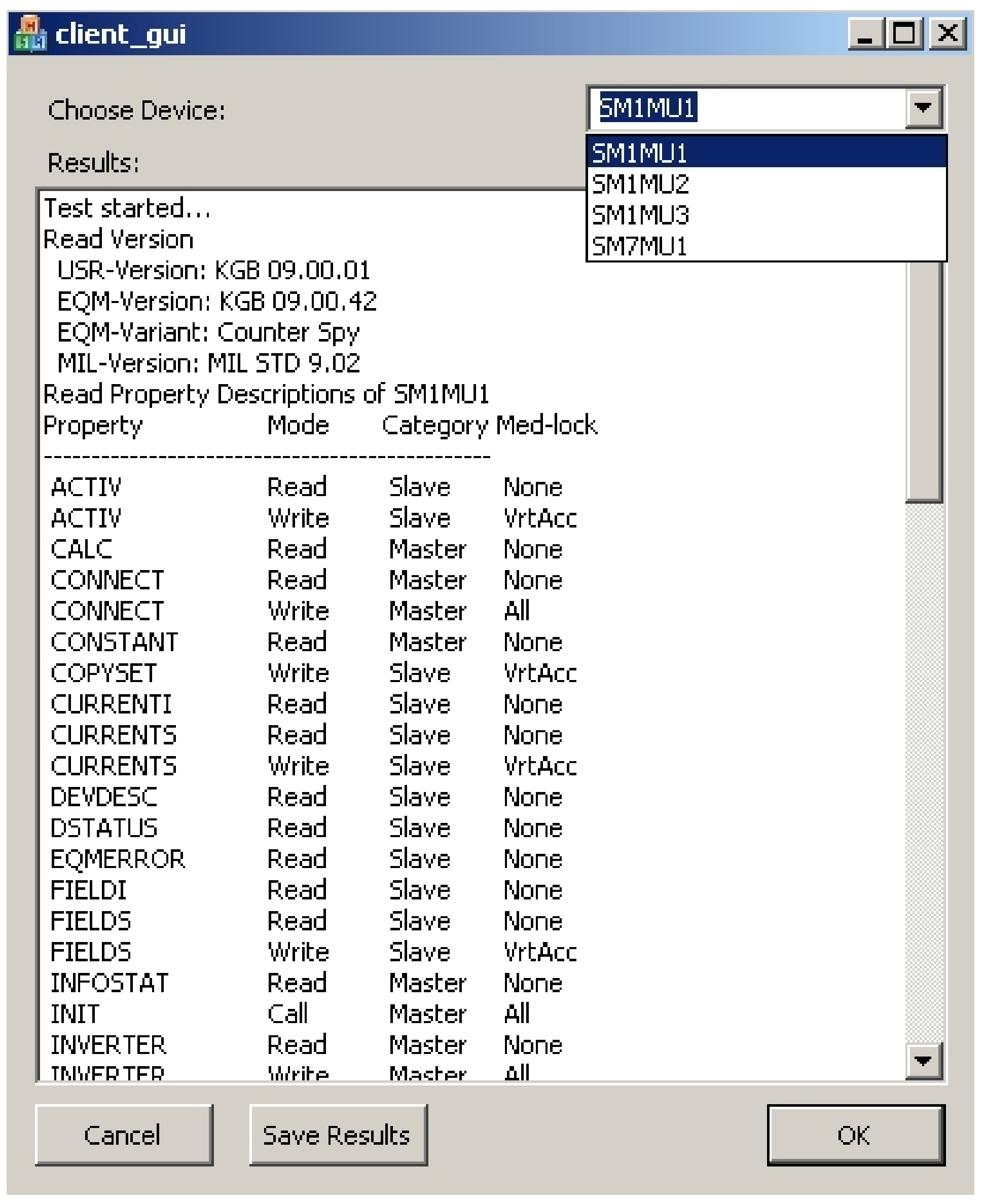


Workflow for generation of C++ sources and API documentation

Windows Client and Server API

C++ client and server code was ported to run on MS Windows:

- ✔ Identification of system dependencies in C++ code
- ✔ Definition of wrapper classes with common layer for system dependent calls, providing the same interface for message queueing, signal handling, and system logging for Windows and Linux
- ✔ Library providing some Unix functionality on Windows



```
Test started...
Read Version
 USR-Version: KGB 09.00.01
 EQM-Version: KGB 09.00.42
 EQM-Variant: Counter Spy
 MIL-Version: MIL STD 9.02
Read Property Descriptions of SM1MU1
Property          Mode      Category Med-lock
-----------------------------------------------
 ACTIV             Read      Slave     None
 ACTIV             Write     Slave     VrtAcc
 CALC              Read      Master    None
 CONNECT           Read      Master    None
 CONNECT           Write     Master    All
 CONSTANT          Read      Master    None
 COPYSET           Write     Slave     VrtAcc
 CURRENTI          Read      Slave     None
 CURRENTS          Read      Slave     None
 CURRENTS          Write     Slave     VrtAcc
 DEVDESC           Read      Slave     None
 DSTATUS           Read      Slave     None
 EQMERROR          Read      Slave     None
 FIELDI            Read      Slave     None
 FIELDS            Read      Slave     None
 FIELDS            Write     Slave     VrtAcc
 INFOSTAT          Read      Master    None
 INIT              Call      Master    All
 INVERTER          Read      Master    None
 INVERTER          Write     Master    All
```

Screenshot of a Windows Client