SYSTEMATIC PRODUCTION OF BEAMLINE AND OTHER TURNKEY CONTROL SYSTEMS

Gasper Pajor*, Andrej Kosrmlj, Igor Verstovsek, Klemen Zagar, Rok Sabjan (Cosylab, Ljubljana)

Abstract

Turn-key oriented accelerator control system production is often quite complex and challenging. It involves software development as well as substantial project management effort and, almost always, an on-site installation. Most of the labs have developed solutions that to some extent support such processes, but are tailored to the lab's particular needs and environment. We could not recycle these solutions, as we had to keep the choices open for defining the naming convention, choosing the operating system, platform and even the control system.

Based on our experience with control systems, we have defined a complete set of processes that prescribe the highest level of quality and efficiency in all the project segments. To implement these processes, we have developed a number of tools for composing, configuring and deploying the control system software. Use of these tools enforces strict version control and traceability, enables centralized configuration of the system and largely reduces possibility of human errors. These tools also enable us to re-use well tested building blocks, leaving us more time for system-wide quality assurance.

BACKGROUND

The goals, which later transform into benefits, of the software versioning and deployment control process are fairly easy to identify. The process has to be strict where required yet user-friendly and flexible to accommodate various common practices. The result of such process is well defined and repeatable conglomerate of existing elements and particular configurations.

It is less clear, however, how to implement such a process as there are many open parameters to consider. We have identified the following steps:

- definition of the atomic building block (component),
- enumeration and configuration of the components comprising the control system,
- assembly of components into deployable collection of files.

If the interfaces are defined well enough, the above steps can be broken down in three separate processes. Each process can then have its own specialized tools and clearly defined scope.

SQL Database as Interface

We decided to utilize the SQL database as the interface between stages in the process and it proved to be a good choice. It forced us to design the interfaces and double check them at the very beginning to avoid costly database changes later on. As a result the interfaces have stayed the same for almost two years now and so far there has been no need for changing them.

There is another point to choosing the SQL database as an interface; practically every programming language has a library for communication with a database, so we have not limited the choice of implementation language of our tools with the choice of interface.

Apart from the role as an interface, the SQL database also acts as a central information repository, storing the data and providing the foundation of repeatability and traceability.

GENERIC COMPONENTS

The component as we defined it is a recyclable building block of a control system.The element could be anything from a bulk solution for particular subsystem to a single meaningful line that needs to be inserted somewhere. Needless to say, most of the components are somewhere in between.

What makes the components recyclable is the description of configurable parameters and mechanisms to adapt these parameters for each particular use. Apart from the usual meta information such as name, creator, version etc. we have defined groups of name-value-comment triplets. Tags are also associated with each group of triplets that define how the data contained within the group should be interpreted. All the meta information is initially entered in an xml file.

A dedicated tool, the *Orchestrator*, is used to automatically check the component for consistency before its files are committed to CVS and its meta information from the xml file is entered into the SQL database. At the same time the *Orchestrator* tags the CVS with appropriate version, which it also stores in the database. With this the *Orchestrator* guarantees synchronization of component versions in the CVS with the corresponding component information in the database. This consequently means easy extraction of particular component's files later in the process.

CONFIGURATION

The idea of configuration is based on the "classical" signal list that enumerates the relevant signals brought to control system and their properties, alarm limits etc. We decided to base our Signal List tool on Microsoft Excel, mainly to get all the spreadsheet functionality and flexibility out of the box. The built-in Visual Basic

*gasper.pajor@cosylab.com

for Applications, although somewhat peculiar at moments, provided all the programming strength we needed to implement the logic behind the spreadsheet cells.

The Signal List offers the available components to the user, lets the user configure the components' parameters and finally enables the user to store the current configuration to a *Configuration database*.

ASSEMBLY OF THE DEPLOYABLE FILE COLLECTION

The next stage of the process is the joining of components' files from the CVS and the configuration from the *Configuration Database* into a file system, ready to be deployed. To illustrate its function we call the tool that performs this task the *Weldor*.

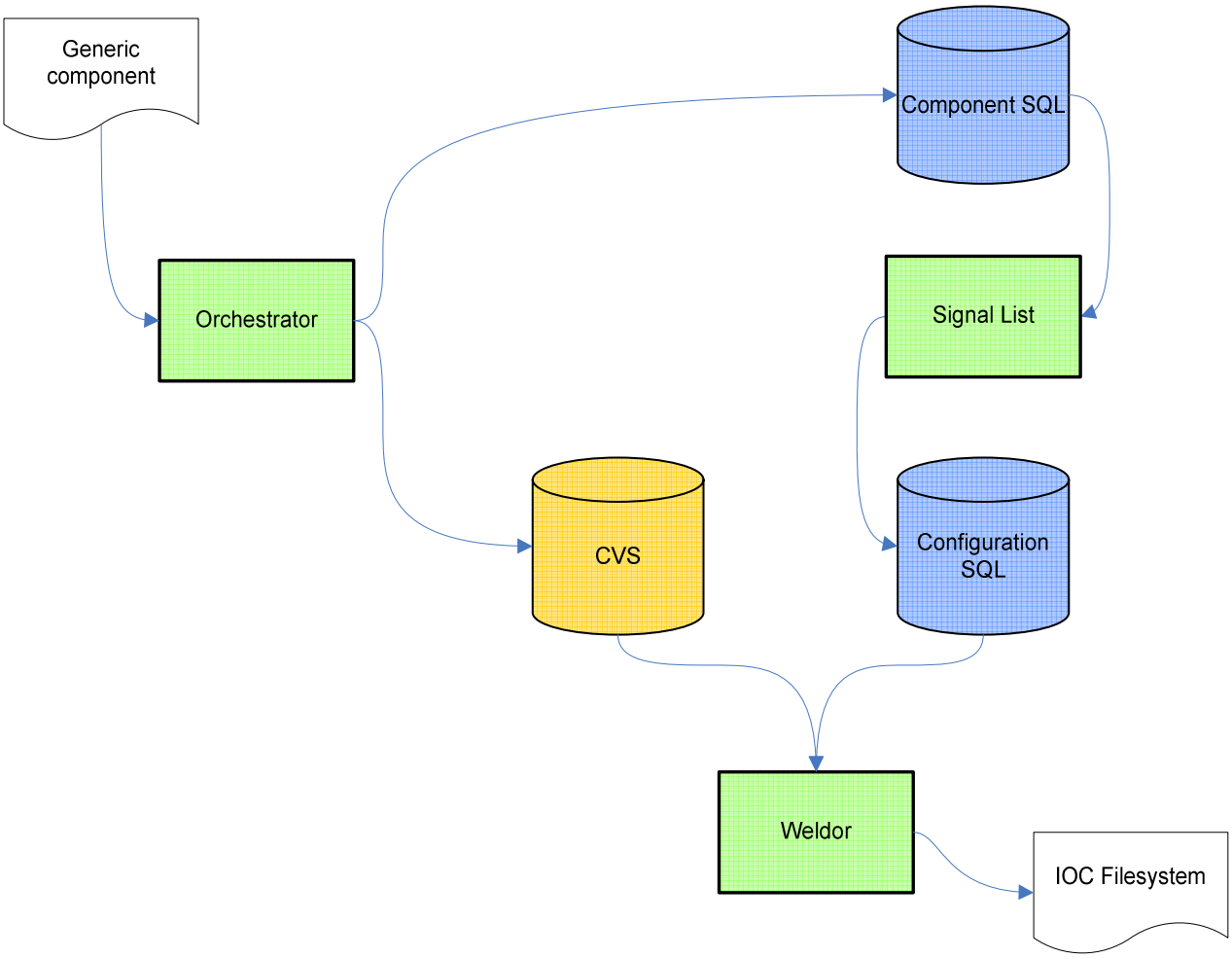

Figure 1: Generator toolset and flow diagram.

The palette of available components (and their versions, of course), is fetched from the *Components database*. Once added to the *Signal List*, the component is represented by a section of cells on the Excel sheet. User can then change the default parameters by editing respective cells.

When the users finish with the configuration, they can export the data from the *Signal List* to *Configuration Database*. The exported data contains information about components and their attributes, which is all that is required for composing the deployable collection of files.

Weldor can be seen as a script that takes a single parameter as its input: the "build ID" i.e. the configuration id number in the *Configuration Database*. The rest of the *welding* is performed automatically in the following five steps:

- Fetching data from *Configuration Database*
- Fetching the components' files from CVS
- Applying naming convention
- Welding
- Invoking the "next stage"

Fetching the data from the *Configuration Database* is based on the build ID. The data then fills Weldor's data model, which is available in all the next steps.

For each component the exact version (tag) of its files is checked out from the CVS and placed into appropriate local directory.

The next step can be seen as somewhat EPICS specific, but it could be in fact used in any control system. The problem we faced on numerous occasions is that almost every lab or site has its own naming convention standards. This is even more emphasized in control systems with flat variable space, where naming convention has to provide clues on where and what the variable represents.

To handle this issue, we decided to introduce *generic names* in the components' files, which are replaced with particular names, formatted and named according to the target lab's naming convention. We implemented this procedure with extensive use of regular expressions.

An example for EPICS channel name transformation for Labs A and B would be:

Name in generic component:

```
$(DEVICE)$(VOLTAGE)$(GET)
```

Transformed name for Lab A:

```
$(DEVICE):VOLT_MONITOR
```

In instance for Lab B:

```
$(DEVICE)::Get-Voltage
```

When all the naming is handled, the *welding* takes place, when the data from *Configuration Database* and the files are properly combined. A separate directory is created and filled with appropriate components' files for each IOC. Then the groups of name-value-comment are applied to the files, according to each group's specification. The simplest group handler types are find/replace, insert and similar, whereas the more complex include the complete logic for accumulating the data and later generating code for low-level controllers (e.g. DeltaTau PMAC [1]).

The result of this step are the IOC directories prepared to be compiled and deployed. The files that are not IOC specific (e.g. GUI applications, documentation etc.) are stored separately.

The "next stage" step is specific for each target platform and can include anything from a simple cleanup to preparation and execution of compiling and deploying.

Although Weldor could have been programmed in a scripting language we have decided to do it in Java. This decision was especially welcome when the need for more advanced handlers emerged. The object oriented approach also provided foundations for easy switching to alternative implementations of each step.

BENEFITS

Although the development cycle is usually longer and transforming existing piece of software into recyclable form takes some time, we have achieved our goal and benefits exceed the mentioned drawbacks by far.

The system-wide testing is usually last on the project schedule, which often means its scope is reduced by the oncoming deadline. The described system provides controlled and efficient re-use of the components, eliminating the need to test the well-tested units each time over. This consequently leaves us more time for system-wide testing and quality assurance.

The process and tools are designed to eliminate many possibilities for human errors. The "atomic" operation of consistency check, CVS commit, CVS tag and new database entry can be set as an obvious example.

FUTURE CONSIDERATIONS

We have already created some by-products of our system, such as the *Documentatore* tool that performs the naming convention routines on Microsoft Word documents and the *Miner* tool that provides database analysis such as logical diff-like comparison between two build IDs. Both of these tools will probably see some improvements.

We plan to change our version control system from CVS to subversion in the near future. That will require some alternative implementation of the parts that now communicate with CVS.

Although Excel is a fantastic tool for spreadsheet manipulation we have not used its functionality as much as we anticipated. This and the fact that other tools are much more often used in Linux than in Windows is something we might have to consider. For now, we use VMware with Windows running as a virtual system inside Linux.

CONCLUSION

We have used this system on all of our larger integration projects since the toolset has been developed. Some of these projects have already been successfully finished while others are well on their way there.

The described tools and processes fit well with our (also in-house developed) project management toolset CPM [2]. This setup, together with accumulated know-how, allows us to provide control system integration and turn-key control system solutions of highest quality.

REFERENCES

[1] Delta Tau Data Systems, Inc. http://www.deltatau.com/

[2] I. Verstovsek et al.: "Management System Tailored to Research Institutes", ICALEPCS 2007