CONTROL SYSTEM RELIABILITY REQUIRES CAREFUL SOFTWARE INSTALLATION PROCEDURES *

R. Müller†, R. Bakker, T. Birke, R. Lange, BESSY, Berlin, Germany

Abstract

The demands on availability and reliability of the control system reflects the stability needed by accelerators in production. On the other hand the system is continuously developed and new features have to be included and adapted. With growing complexity of the system testing for unforeseen side effects becomes increasingly difficult. Frequently control access to pieces of equipment essential for the operation is necessary. Maintenance periods suited for installations are mostly extremely short. This paper describes the release control system and the installation stages used at BESSY to provide a high level of control system transparency and stability.

1 ENVIRONMENT

The third generation light source BESSY II started user operation in 1999[1]. The control system in use at this accelerator is based on the EPICS toolkit[2]. This paper describes the control system maintenance and development strategies providing a high level of both development speed and operational reliability.

1.1 Network Topology

Essential prerequisite of manageable installation procedures is a proper separation of control system development area and the system used for control and operation of the accelerator. At BESSY this requirement is met by common measures like segmentation of the computer network.

Software is developed in a standard desktop environment configured around a file server in an easily accessible Internet domain (stage 0 in fig. 1). Here functionalities are tested for the first time at test stands. In contrast to that relatively open environment the control system in production runs on a highly protected, non-routable Intranet with autonomous file server, consoles and front end computers (IOCs). Independent network components as well as fail over configurations guarantee minimized interruption of service in case of hardware failures or other typical computer problems.

1.2 Test Stages

Within the control network there is a dedicated test machine. Access restrictions for this computer are modified

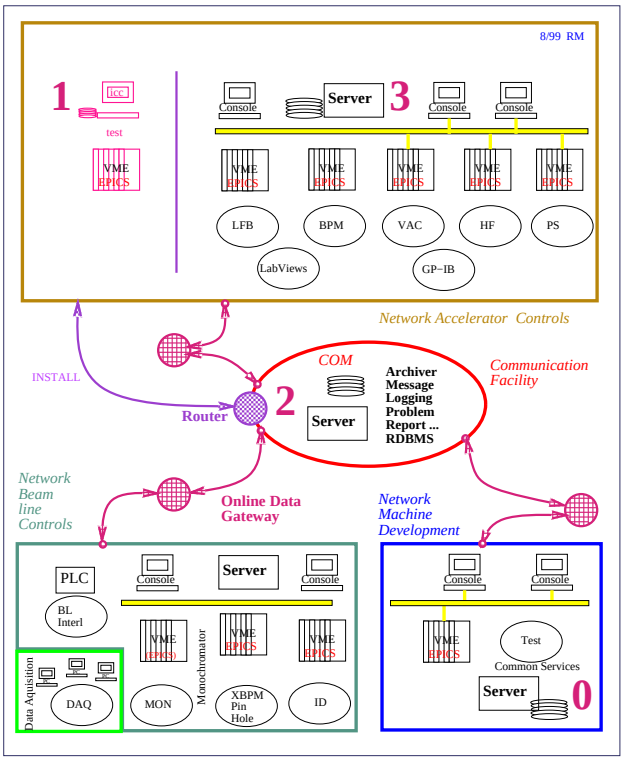

Figure 1: Sketch of the Network Topology.

to allow for downloads of new software releases from the development area. System configuration is an independent structural replication of the controls file server. Within the whole controls environment user accounts are mapped to functionalities like operator, maintainer etc. The transfer of developers ownership of modules to functionalities already takes place on the test machine. Access security allows full hardware control from this computer to administrator accounts with installing and testing permission.

If a new software module or version is scheduled for going into operation it first has to be installed on the test machine either in the boot area of the IOC computers or into the tree of console applications (stage 1 in fig. 1). As soon as a time slot for tests becomes available the new versions are tested by running console applications on the test machine or booting the IOCs from the test area. Discovered bugs are fixed, the improved version reinstalled and retested. Whenever a test time slot ends the switch back to the settled productional version requires in its worst case the reboot of the affected IOCs. If the new modules appear to be sufficiently mature the version repository is updated, the

* Funded by the Bundesministerium für Bildung, Wissenschaft Forschung und Technologie and by the Land Berlin

† Email: mueller@bii.bessy.de

modules are installed on the file server and become the new productional version (stage 2/3 in fig. 1).

1.3 Multi-homed Communication Facility

Similar to accelerator controls dedicated networks for the beamline systems and major user groups are presently in the phase of being set up. Independency of these areas are desirable for operational security and availability reasons but require additional infrastructure.

Data Switch-Yard: For numerous correlation and monitoring tasks online data have to be passed between the different networks. E.g. the beam intensity measured by the accelerator control system has to be available also at the beamlines. Equally the read-outs of the diagnostic XBPM systems in the beamline area are needed by the orbit stability control modules within the accelerator segment. The data transfers between the networks is provided by gateway programs that control transfer of access security and impact on network bandwidth. It is foreseen to run these gateway processes on dedicated multi-homed communication facilities (Fig. 1).

Versioning of Production Systems: At BESSY there are several layers (core system, toolkit libraries, generic and custom applications) and streams of development (accelerator, ID, beamline). If a subsystem has to go out of sync and certain versions of the involved software have to be frozen today this is mostly done by copying the required modules to separate locations and using this extra tree for the build. With increasing complexity of the systems it becomes essential to replace this error-prone and inflexible procedure and maintain a uniform and transparent versioning of all production quality software that allows an easy roll-back to preceding versions. The central communication facility will provide the required file server functionality for this common repository (stage 2 in fig. 1).

2 SOFTWARE DEVELOPMENT

Guiding ideas behind the common and mandatory source code development structure have been roll-back and branch-merging capability, transparency and flexibility as well as cooperative maintainability.

2.1 Application Development Environment

Unified directory trees with preconfigured Makefiles, install locations and handles to the proper version of the core modules are provided to the application developer by a central script (*makeBaseApp*). The common structure eases migration to newer releases of the base modules and the transfer of modules to another developer.

2.2 Source Code Control System

Certain intermediate or stable stages of a software module are preserved by the check in into a source code repository. At BESSY the package *Concurrent Versions System* (CVS)[3] is used. CVS features concurrent access, conflict resolution and branch merging strategies. Logging of source file changes and symbolically tagging of certain releases simplifies a secure roll-back to a previous version. Access and protection is controlled by standard Unix file permissions. Developers cooperating on a certain package have a common group identification. Core developers have to pay additional attention to licensing terms and utilize a pseudo user identification. For inter-division exchange it is planned to set up a CVS server to allow for secure and networked code maintenance and retrieval.

2.3 Module Encapsulation

Shared libraries are used to facilitate independent bug fixing and maintenance of modules used by several console applications. Typically the interface definition of a module is not changed. Then it only requires to install the improved version of the shared library to fix all applications that use the module. On the back side a careful bookkeeping of all affected software components is required if the interface definition has to be changed and the installation of the new shared library is required.

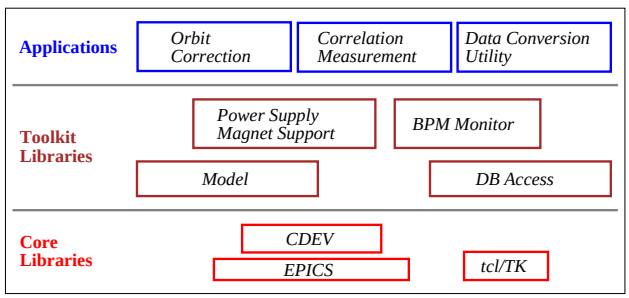

Figure 2: Vertical Dependencies: Module Layers.

2.4 Installation Environment and Procedures

A very convenient feature of the Application Development Environment (See 2.1) is the support of installation and distribution procedures. For each module *Makefiles* allow to install binaries, libraries, include and resource files as well as scripts controlling environment variables into the appropriate location. Distribution stages like the installation into the developers working directory, into the global development area, onto the test machine and finally into the production area file server are easily addressable. The utilized distribution mechanism *rdist* takes care of the actions necessary for a synchronization of the module.

No structural solution is available yet for vertical inter-module dependencies. If a base or intermediate module is modified it requires the definition of a coordinated rebuild and installation campaign to get all affected applications updated. Even if the whole installation process has to run through the test stages described in 1.2 the vulnerability of the procedure requires versioning measures.

During commissioning many new installations and modifications took place in parallel. Therefore full copies of the accelerator control system in production have been saved routinely to be able to return to a previous, operational version if the reason of an unexplainable malfunctioning could not be found in due time. This procedure quickly turned out to become unmanageable. In the present stabilized situation at least certain installation blocks are not overwritten, but moved into the attic to provide a minimal version history. As already sketched in 1.3 it is planned to set up a complete versioning of binaries, libraries and configurations that ever went into production.

3 CONFIGURATION MANAGEMENT

The most difficult part of a 'living' control system is a systematic, consistent and trouble free maintenance of the various configuration data and their interdependencies.

3.1 Reference Data Repository

At BESSY a relational database serves as a central reference data repository. Configured around the notion of a 'device' and its name this DB is supposed to hold all required conversion factors, geometrical data, connections and relations[4].

The whole DB is handled similar to the applications. Independent database instances are set up in the development area, on the test machine and in the production area. By stepwise replication the DB has to follow the test stages that are mandatory for the applications (see 1.2). As an additional back-up strategy the periodic export of the whole DB to an external mass storage system takes care of the unique requirement on consistency, integrity and availability of the DB content.

3.2 Propagation of Device Modifications

Even if a large fraction of the configuration data are generated by scripts from the DB there is still a number of hand-edited files and dependencies that require manual interaction.

If e.g. a new device is added to the system or a device is modified the 'horizontal' dependencies of the various tools have to be checked to be able to identify affected entities. If the new device belongs to an existing device class a re-run of the configuration scripts covers most of the adaptation work. If a new device class is added the generating scripts have to be adapted and the net of correlations and dependencies has to be adapted (Fig. 3).

4 PROBLEM TRACKING

Complexity of the system, lack of time and limited availability of the required hardware prevent the clear exclusion of problems newly induced with the modifications.

In order to avoid unforeseen side effects a mailing list mandatory for the controls and application developers has been set up. Pending new installations or possible problems are announced on the list to prevent misunderstandings or need for additional activities.

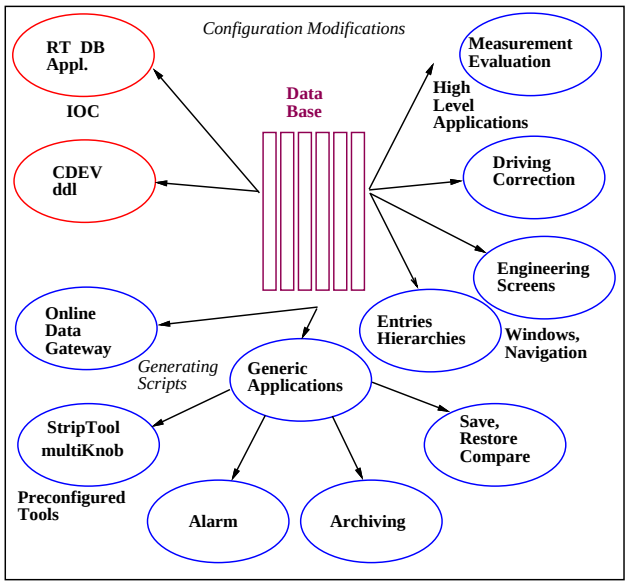

Figure 3: Horizontal Dependencies: Propagation of Modifications.

The *gnats* problem reporting system has been configured to cover the classical service areas (power supplies, RF, vacuum, controls) and notify the person(s) in charge of the subsystem via email. Operators are encouraged to use this tool as a checkable, reliable and transparent tool to transmit complaints, observations or comments. A tcl/Tk and a WEB interface help to create and submit reports, check the progress on a subject etc.

5 SUMMARY

The combination of version control, staged installation procedures and partly automatized configuration management turned out to be a manageable compromise of effort and efficiency. The resulting availability, consistency and reliability justifies today's unquestionable confidence into the control system.

6 REFERENCES

[1] R. Bakker et al., 'Status and Commissioning Results of BESSY II', Proceedings of the 1999 PAC, New York, 1999

[2] R. Bakker et al., Proceedings of the 1998 EPAC, Stockholm, 1998, p.1676

[3] CVS has been developed by Per Cederqvist et. al. and is available under the GNU General Public License

[4] R. Bakker, T. Birke, B. Kuske, B. Martin, R. Müller, Proceedings of the 1997 ICALEPCS, Beijing, 1997, p.407