‘NEARLY’ GENERIC MONITORING AND CONTROL PROGRAMS MAINTAIN BEAM QUALITY

R.Bakker, T.Birke, B.Kuske, R.Müller
BESSY, Bld. 14.51, Rudower Chaussee 5, D-12489 Berlin, Germany

Abstract
For beam focusing and steering at BESSY II applications are used that are generic enough to describe and correct all accelerator sections at BESSY II by single program instances. E.g. the program ‘Orbit’ provides the appropriate execution modes to measure and correct the beam trajectory in the fast cycling booster synchrotron (different energy levels), in the storage ring (closed orbit and single turn measurement mode) and in the different transfer lines. Basic ingrediends are an OO modelling toolkit, an ORACLE database providing the configuration data connecting magnets and power supplies and a well separated interface to the GUI. Physics methods dynamically adapt to the specific system. Device bookkeeping and screen update is generally based on callbacks. As residual non-generic program elements the data navigation features, like display options, meaningful operation modes etc., remain and reflect the specific system differences.

1. INTRODUCTION

Third generation light sources have to deliver beam quality beyond todays technically achievable passive stability. Requirements for well defined beam conditions are typically met by control system applications performing periodic orbit recenterings and optics readjustments. Despite the fact that programs in use at other light sources or ‘particle factories’ are comparable with respect to functionality, precision and reliability there is no common or sharable approach available yet.

The following description of the beam steering and shaping applications in use at BESSY II is focused on the software modules and interfaces, not on achieved results or operational performance. These programs are not yet sharable but generic in the sense that single program instances describe and control all different accelerator sections at BESSY II. The orbit measurement and correction program for example reads and steers the beam trajectories in the fast cycling booster synchrotron (different energy levels), in the storage ring (closed orbit and single turn measurement mode) and in the different transfer lines.

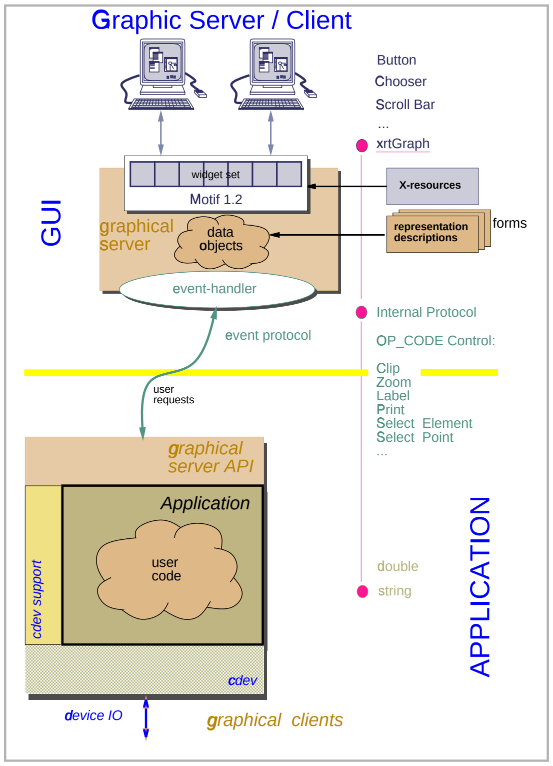
2. GENERAL PROGRAM ANATOMY

2.1 Program Components

The application environment at BESSY II partly determines the fundamental layout of the programs:

2.11 Graphical User Interface (GUI)

User interaction and data presentation is handled by a dedicated GUI server [1]. Applications are graphical clients using the API to this server, free from graphical code and not aware of any GUI element. User requests (type *button, chooser*) or data updates (type *meter, message*) are mediated by changes of the corresponding application variables (type *double, string*). The actual representation instances (look and feel, layout, multiplicity etc.) of the application variables are described independently and externally in ‘form’ files (see Fig. 1). The data objects linking representation and application entities are maintained within the graphics server. The only exception of a not fully decoupled graphical presentation is imposed



Goemon: C++ Model Toolkit

Class Categories:

Component: Inheritance
Element - Drift - Quad - …
- Marker - Sextupole - …
- BPM - ...

Machine: Inheritance
Beamline - Ring - Storage Ring …
- Transfer Line - ...

Accelerator Object:

Section Composition: Overloaded Operator +

E-Section$_N$ = Elem$_A$ + Elem$_B$ + …
Length, Type

B-Section$_N$ = B-Elem$_A$ + B-Elem$_B$ + …
Strenght, Transfer Functions

Beamline = B-Section$_1$ + B-Section$_2$ + ...

Member Functions:

Matrix / Vector:	Mathematics
Get / Set:	Model Set Points
Calculation:	Optical Functions

Fig. 1: Elements of the BESSY Development Environment: GUI Server/Client (left) and C++ Model Toolkit (right)

by the complex *xrtGraph*[2] widget used for 2D diagrams (examples in fig. 5, 6): Here a simple one-by-one mapping of widget and application data sets is not possible: a ‘command and notify’ string is used in an OP-code type manner to control basic widget functionalities (e.g. attach string array of labels to arrays of a datasets) or to report results of user actions (e.g. selection of a data region).

2.12 Modelling Toolkit

A C++ modelling toolkit (Goemon)[3] features inheritance of single lattice components as well as of various beam line types (see Fig. 1). Different models are easily set up using overloaded assignment operators to model smaller sectors of an accelerator structure and collate them to the section under study. Sequences of elements of certain type and length are turned into accelerator objects by adding magnetic strength and appropriate transfer functions. Member functions of the accelerator object allow to manipulate the model (set points, displacements, etc.) and to perform optics calculations.

2.13 Unique Reference Data

A central ORACLE database holds the reference repository providing a unique data source for all relevant configurations[4]. Data are set up and maintained by the equipment responsible person (Fig. 2). Export of data to the download area of the IOC’s or to cache files used by generic applications is done by programs or scripts (Fig. 3). Dynamic programs use DB queries (OCI) for their configuration. For the model toolkit element sequence, length and position are extracted from the database.

2.14 Support Utilities

The C++ model object may be passed to utility packages by its pointer and queried or modified by the (optical) member functions. Magnet - power-supply wiring (Fig. 3), conversion factors, device I/O,

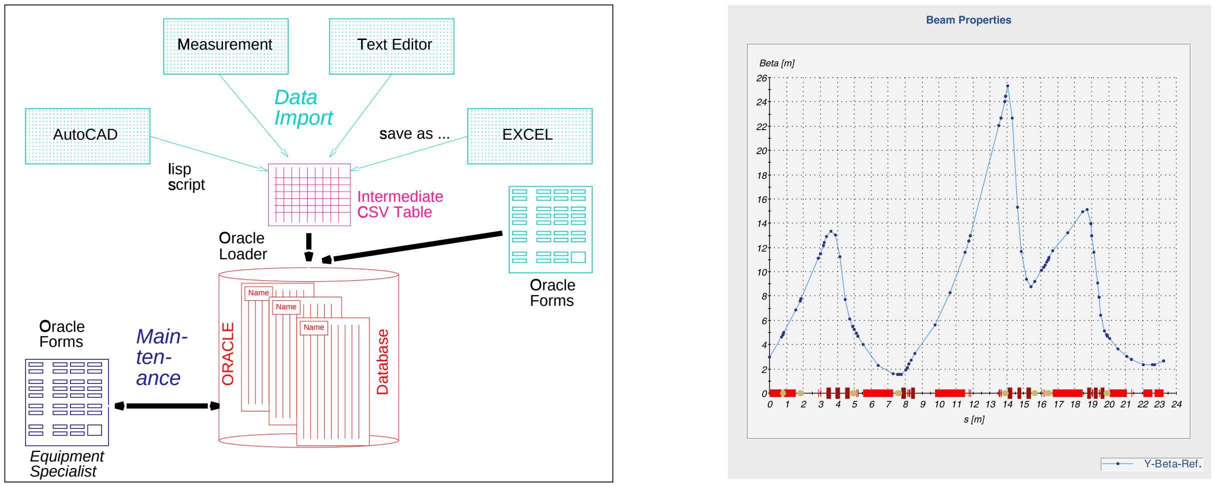


Fig. 2: Role of the Reference Database: Data are Imported from Genuine Sources. Ownership is Preserved. Update is done by the Equipment Responsible Person (left), For Model Construction Element Sequence and Length is used (right)

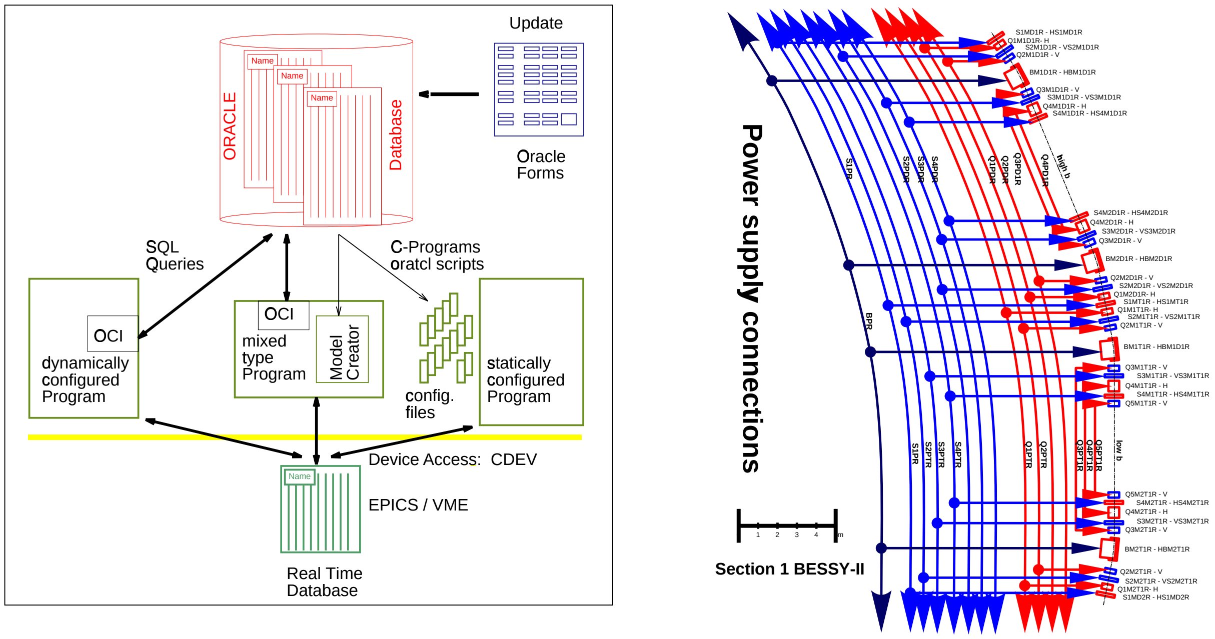


Fig. 3: Role of the Reference Database: Data Export Mechanisms for various Generic Applications (left). Mapping between Control System and Model (Wiring, Conversion Factors, device I/O, Synchronized Model/Power Supply Update) (right)

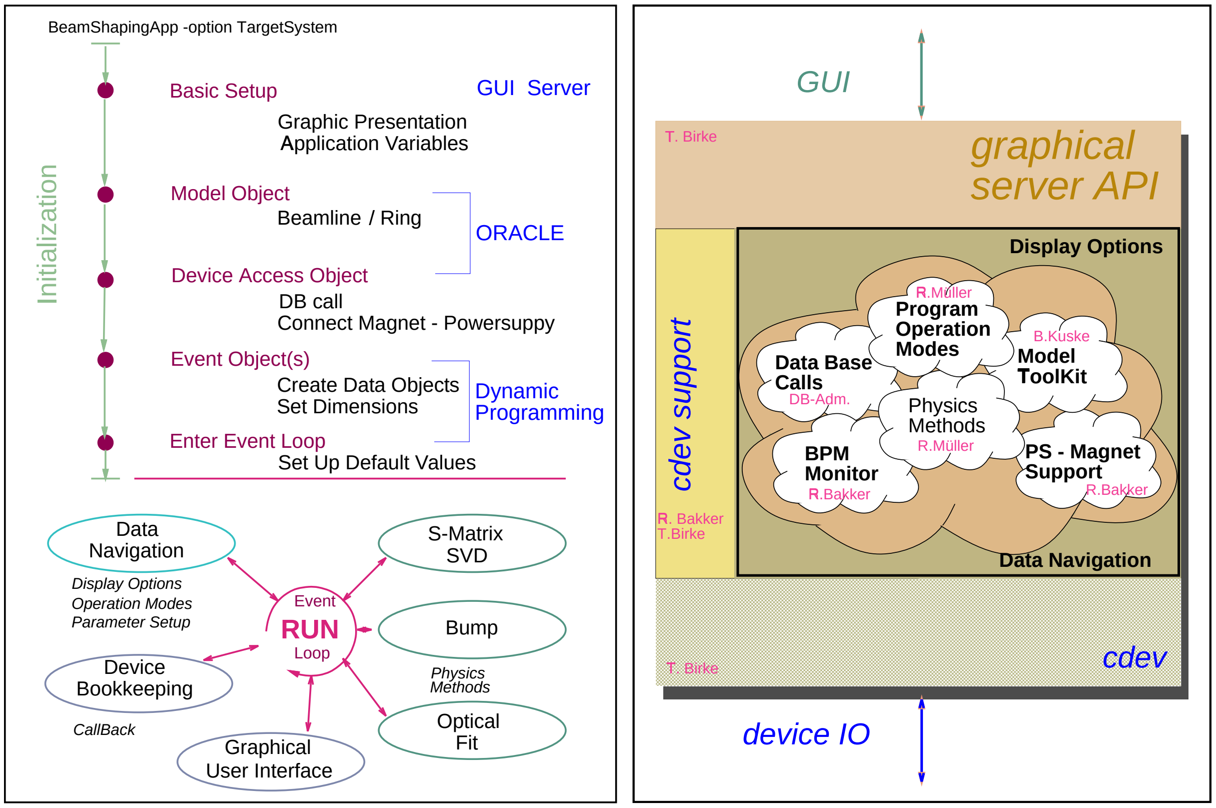


Fig. 4: Sketch of Program Flow (left), Corresponding Work Break Up (right)

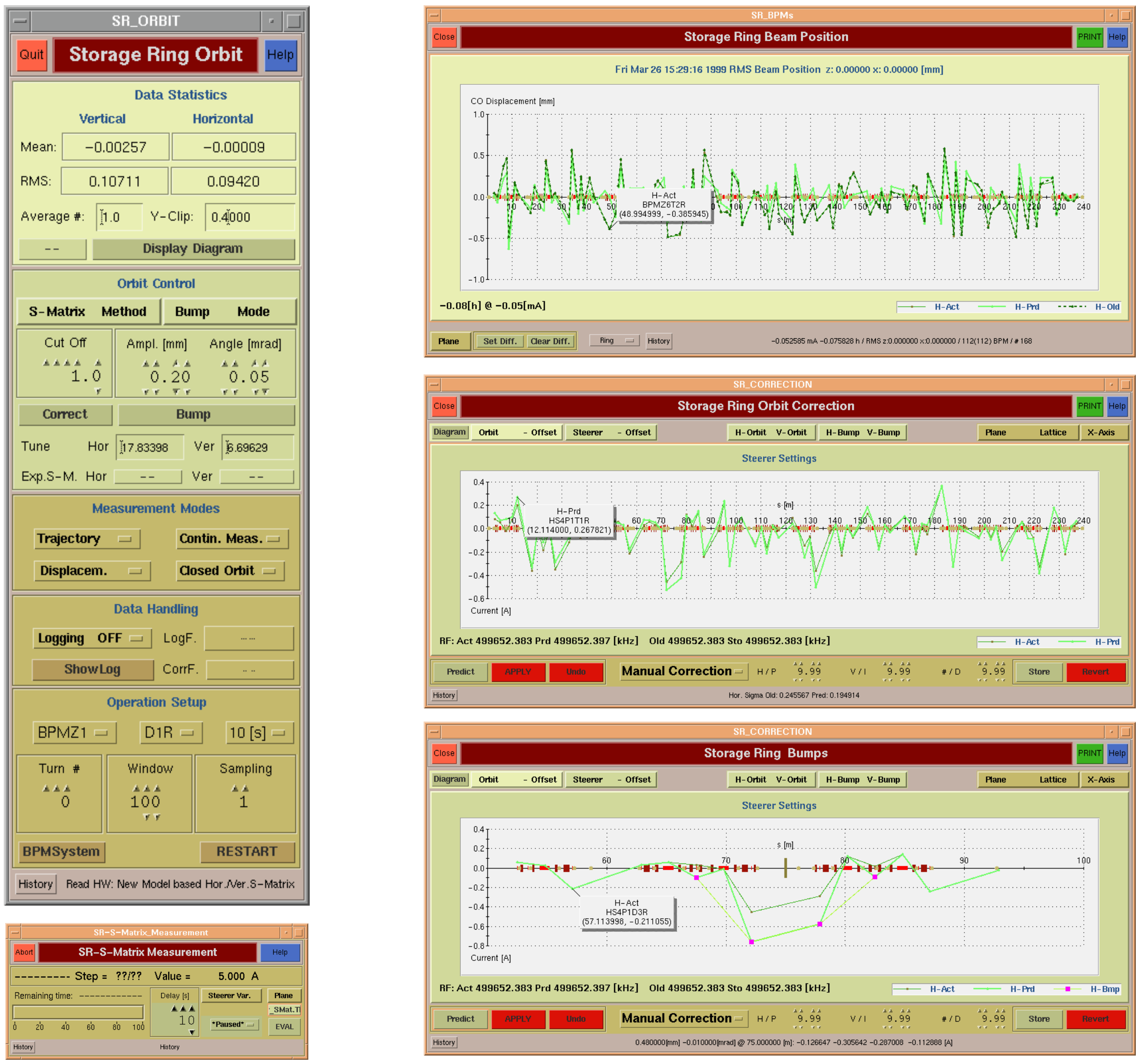
power supply set point synchronization, model update and fitting of optical parameters are taken care of by such a support library. Orbit measurements, BPM data validity checks and statistical analysis are handled by another utility library.

2.2 Program Flow

Only during start-up the programs branch according to the command line arguments: the application triggers the GUI server to read the appropriate base form and marks (if needed) the accelerator section to be modeled. Further on the program auto-configures (see fig. 4):

- Basic set up: from the variables and presentation elements found in the associated graphic description file the GUI server configures screen and data objects, allocation of the corresponding data structures within the application is initiated.
- Model/Device access objects: replies from the database provide the informations needed to set up the model and the magnet - power-supply correlations.
- Data buffers and event objects are dynamically created according to detected dimensions, correlations and associated callback functions.

Physics methods adapt to the particular system. Device bookkeeping and screen update is generally based on callbacks. As residual non-generic program elements specific data navigation features, like display options, meaningful system or program operation modes etc., remain and reflect the specific system differences.


Mean: Vertical −0.00257 Horizontal −0.00009

RMS: Vertical 0.10711 Horizontal 0.09420

Average #: 1.0 Y−Clip: 0.4000

Tune Hor 17.83398 Ver 6.69629

Fri Mar 26 15:29:16 1999 RMS Beam Position z: 0.00000 x: 0.00000 [mm]

−0.08[h] @ −0.05[mA]

RF: Act 499652.383 Prd 499652.397 [kHz] Old 499652.383 Sto 499652.383 [kHz]

Hor. Sigma Old: 0.245567 Pred: 0.194914

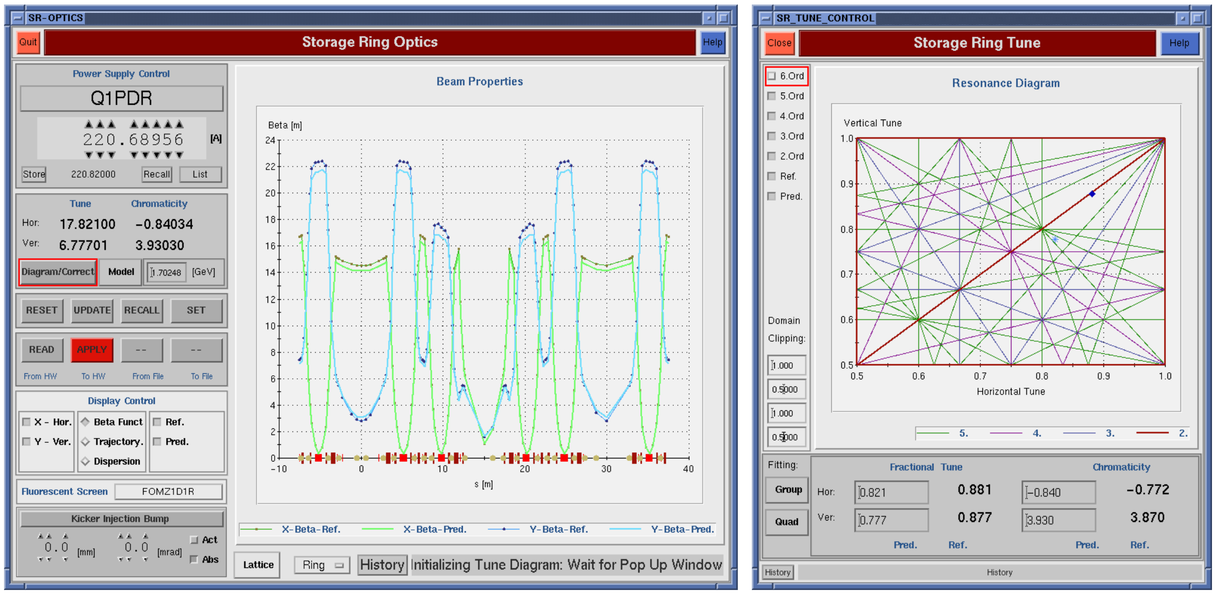
Fig. 5: The ‘Orbit’ GUI: Main Controller Windows (left), BPM Data, Orbit Correction and Bump Screens (right, from top)

3. APPLICATIONS

3.1 Orbit Control

The ‘Orbit’ display and control program provides a wide variety of functionalities covering several areas of control activity in the different accelerator sections synchrotron, storage ring and transfer lines. The following basic program modes have already been described in more detail [5]:

- support of an ‘All-in-One’ GUI (Fig. 5)
- data presentation, statistical evaluation
- control of the measurement system and manipulation of data validity
- manual or automatic orbit correction with different methods under varying conditions
- selection of closed orbit bumps from preconfigured lists (insertion device, achromat) or by free configuration (click on coordinate, correction element)
- data correlation, response matrix measurement:
  device stepping, data logging, postprocessor launching.


SR-OPTICS — Storage Ring Optics

Power Supply Control: Q1PDR 220.68956 [A]; Store 220.82000

Tune / Chromaticity — Hor: 17.82100 −0.84034; Ver: 6.77701 3.93030

Model 1.70248 [GeV]

Fluorescent Screen: FOMZ1D1R

Kicker Injection Bump: 0.0 [mm] 0.0 [mrad]

Initializing Tune Diagram: Wait for Pop Up Window

SR_TUNE_CONTROL — Storage Ring Tune

Domain Clipping: 1.000, 0.5000, 1.000, 0.5000

Fitting:	Fractional Tune Pred.	Fractional Tune Ref.	Chromaticity Pred.	Chromaticity Ref.
Hor:	0.821	0.881	−0.840	−0.772
Ver:	0.777	0.877	3.930	3.870

Fig. 6: The ‘Optics: Main Controller Window (left), Tune and Chromaticity Correction (right)

Due to feedback of evaluated measurements into the current/magnetic field conversion factors and the BPM calibrations the program works even in the model based mode very reliable and accurate.

3.2 Optics

Since tune and chromaticity measurements are still offline the ‘Optics’ program is in a less mature stage. It serves more as an online model describing the optical properties of the actual machine set points and their dependencies on parameter variations (See fig. 6 and [5]). Fitting procedures adjusting the tune with a selectable quadrupole group are first steps toward an automated compensation of tune shift and beta beating caused by gap variation of not fully balanced insertion devices.

4. SUMMARY, EXPERIENCES

In summary the consistency of a generic and completely event driven system has appreciable advantages. The reference repository and the C++ modelling toolkit have shown their benefits and established the environment for powerful support libraries. The generic applications evolved ‘down-stream’ with the installation procedure of the accelerator segments. They worked very reliable at start-up of the storage ring and turned out to be essential for the commissioning procedure.

5. LESSONS LEARNED

The generality achieved with the applications developed at BESSY and the examination of portable solutions successfully applied elsewhere give an idea of a possible framework for fully generic, sharable ABS software.

5.1 Required Modules

A precursor of generic ABS applications would be a common repository of easy integrable ‘component-ware’. The core of the typical beam steering control issues, like

- Orbit recentering and manipulation
- Dispersion control

- Coupling compensation
- Emittance conservation
- Lifetime improvement

comprise comparable software fragments with only slightly differing methods and implementation constraints. The example of a reliable, precise and flexible orbit control program gives insight on the ingredients of the major modules:

- Solver/Calculation Module:
  A number of different mathematical methods are well established and known to find the correction suited ‘best’ for a specific situation (SVD, harmonic correction, MICADO, etc.). The achievable ‘improvement’ depends strongly on the appropriate weighing of the different figure of merits (RMS deviation at electronic/photon BPM, ‘fixed’ source point, etc.) as well as on the specific problem break up strategy (global/local correction subsystems, model based/experimental response matrix etc.).
  Data conditioning strategies have an equally strong effect on the performance of a correction procedure. BPM data require a plausibility analysis, weighing, detection of improper operation of the measurement system. Limits on the adressable setpoints of the involved correctors (status, max values, digital resolution) add the necessity of boundary condition handling to the calculation of the ideal correction.
  In this context implementation issues (API, programming language) as well as dependencies on external software packages are more technical portability aspects.
- Data Aquisition, Apply Procedures:
  This program segments strongly depend on the specifics of the measurement system and the different control system protocols. This is reflected in the principles of sending corrections (synchronization) and by the implementation variants (control system communication paradigm). But there are still a number of common procedures, e.g. the orbit measurement data have to undergo validity checks, fault detection analysis etc. Ramp utilities have to handle proper step size, react on errors and exceptions. Drift compensations (‘slow feedback’ procedures) require adjustable regulator algorithms (PID).

5.2 Minimal Common Ground: Interfaces, Architecture

There are numerous ways to glue different software modules together to make up applications with a comparable spectrum of functionalities. For generic ABS software a minimal granularity seems to be necessary (Fig. 7): Various solver and correction calculation facilities as well as an integrated model are the obvious core of any ABS application. The GUI and the control system I/O should be separated by appropriate interfaces, preferable API’s. CDEV [6] as middleware between application and control system has already proven to enhance program portability. For the GUI java is a promising candidate already under study at various places. In addition the data taking and evaluating engine as well as the set point applying ramp unit should be well encapsulated to allow for specifics of BPM and corrector systems. Finally localization of program configuration is a must for any generic solution.

5.3 Remaining Problem Source: Diversity

On the way to operational application programs a number of different implementation variants are feasible that are well suited to impair the usefulness of an ABS program for another site:

- Model Integration: sucessfully applied solutions comprise toolkit libraries, model servers, shell scripts using standard tools (e.g. MAD) in a filter/pipe mode or sets of precalculated output data.
- Configuration Management: if a RDBMS is properly set up, online queries are feasible. Otherwise (more or less) standardized file/data set formats are necessary.

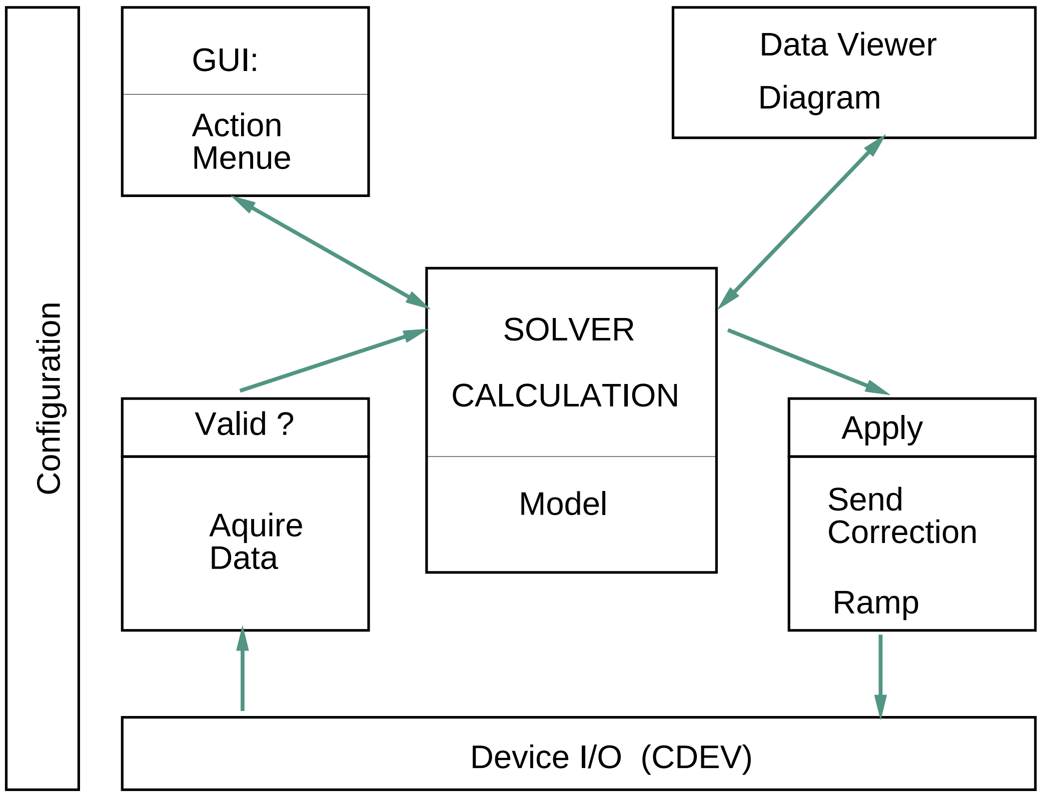


Fig. 7: Sketch of a possible Architecture for Sharable ABS software

- Architecture: agreement on a number of established, canonical toolkit modules is certainly less flexible than well defined APIs or pipe chains (SDDS) allowing for a variety of underlying modules.
- Flow control: extreme flavours are ‘sequential’ (allowing for command file configuration, interpreted interfaces) and ‘data driven’ (asynchronous, based on callbacks, event oriented)
- GUI technology: a common understanding of a minimal and sufficiently powerful set of user interaction and data viewing elements is hard to achieve.

References

[1] T. Birke, R. Lange, R. Müller, Proceedings of the 1995 ICALEPCS, Chicago, 1995, p.648

[2] XRT/graph is a trademark of KL Group Inc., Toronto, Ontario

[3] H. Nishimura, Nucl. Instr. Meth., A 352 (1994), p.379

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

[5] R. Bakker, T. Birke, B. Kuske, R. Lange, R. Müller, Experiences with Commissioning Software Tools at BESSY II, Proceedings of the 1999 PAC, New York, 1999 (WEP49)

[6] J. Chen, W. Akers, G. Heyes, D. Wu, C. Watson, Proceedings of the 1995 ICALEPCS, Chicago, 1995, p.97