A Complete Data Structure for the High Level Software of ELETTRA

Mark Plesko
Sincrotrone Trieste
Padriciano 99, I-34012 TRIESTE
e-mail: plesko@elettra.trieste.it

Abstract:

ELETTRA is a third generation light source at an electron beam energy of 1.5 to 2 GeV. For commissioning and operation a multitude of high level applications had to be prepared, preferably not depending on any details of the control system or parameters which might change during time. The unique data structure which is used in all applications, the underlying file structure which contains all necessary data unseen by the application programmers and the set of utility routines which transparently access the control system are presented in this paper. The data structure contains all data which describe the transfer line and storage ring from the machine physics point of view and in addition contains all necessary references to the control system in order to access single and multiple controlled points. In this way it greatly simplifies the development of applications. The content of the structure is not specific to ELETTRA, since it is generated dynamically at run-time from several text-only editable data files. Due to this generality, it can be easily ported to any accelerator complex.

I. INTRODUCTION

The third generation light source ELETTRA [1] consists of a linear accelerator, a storage ring and a 100m long transfer line between them. Both the transfer line and the storage ring are attached to a control system, which has been developed in-house [2].

Modern accelerator control systems like the one of ELETTRA allow digital access to practically all points of the accelerator. Most of the machine physics related tasks that were previously done manually and analysed off-line can now be automated via software which has direct access to the controlled points through the control system. The software which performs these tasks is generally called high level software (HLS). The list of typical HLS applications includes orbit correction, modelling, measurement of Twiss parameters and machine monitoring. The HLS applications written for ELETTRA are described in a separate contribution [3].

For the HLS of ELETTRA a special machine physics oriented data structure was developed for the following reasons:

- the HLS was authored by machine physicists who have little knowledge of the control system details;
- most of the HLS operates on machine physics variables which have to be readily available to the programmer;
- much of the constant data necessary for the HLS is not provided by the control system;
- the individual HLS programs should not contain any explicit reference to control variables;
- the actual control system parameters and procedure calls must be hidden from the HLS programmer in a transparent way;
- the HLS should be maximally flexible in case of hardware changes, even to the point of being easily portable to other accelerators.

The following section describes the main aspects of the data structure in detail.

II. THE HIGH LEVEL DATA STRUCTURE

The data structure was developed with the following goals in mind:

- allow management of the constant data from one single source in order to quickly respond to hardware changes;
- keep control system related data separate from machine physics related data;
- design the structure as general as possible, ready for later requests and for use on other accelerators;
- provide all relevant machine physics parameters readily in the data structure or on a function call;
- create the data structure dynamically whenever the HLS program starts making it thus completely independent of the hardware platform;
- provide general modelling tools and the necessary data for each HLS application;
- channel all actions to/from the hardware through one routine which can be updated to any control system changes without any modification of the HLS;
- force HLS programmers to a standardised approach thus simplifying maintenance of the HLS.

All the requirements led to the design of the three components of the data structure: the **fixed data files**, the **run-time data structure** and the **utility routines**. The data files contain the calibration data and other fixed parameters of the accelerator elements. The run-time data structure keeps these fixed data and other data that is read in from the control system and converted to machine physics parameters, such that it is easily accessed by the HLS program. The utility routines are divided into three layers (see figure 1). The lowest layer is communicating with the control system and is not directly accessed by the HLS. The middle layer provides generic routines to access single components or groups of them and store obtained data in the run-time data structure. The uppermost layer operates only on machine physics variables and provides modelling tools. The HLS application is written on top of these layers.

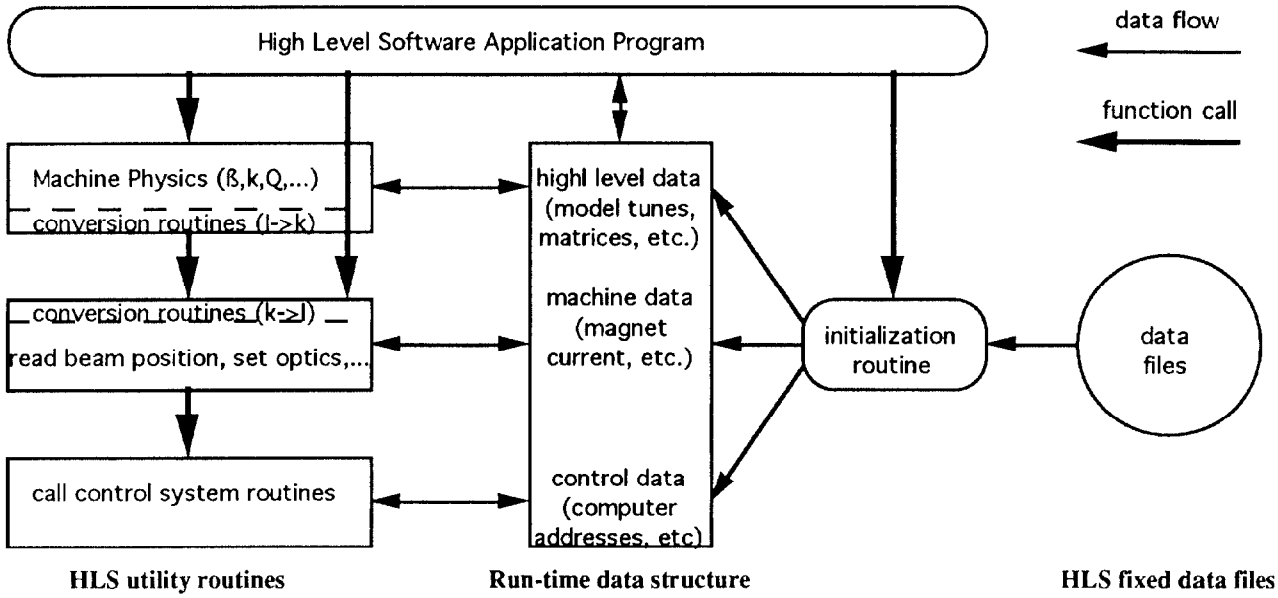

Figure 1: The components of the HLS data structure in relation to a HLS application.

A. The Structure of the High Level Data Files

The high level software (HLS) files contain the constant parameters that describe the ELETTRA transfer line and storage ring from the machine physics point of view. The data in the files is read by an initialisation routine, which must be called by the application program. The routine creates the run-time data structure providing access to all the data from within application programs.

The files for the HLS are the *Calibration File*, the *Parameter File*, the *Access File* and the *Structure File*. The Calibration File contains data obtained from the calibration of elements, mainly magnetic elements (bend, quad, sextupole, ID, corrector, etc.) but may contain calibration data for any type of equipment. The Parameter File contains fixed parameters that describe non-changing properties of elements, like length, lower and upper limits of the changing properties, etc. The Access File contains information about the control system addresses, like name of equipment, control computer host and port of control routines and references to relevant records in the Parameter and Calibration File. Finally, the Structure File represents the logical structure of the transfer line or the storage ring, with their elements listed according to their position in the structure. For each element, there is one record containing the name and the type of the element and references to the relevant Access and Calibration File records.

The Calibration, the Parameter and the Access File contain data of all elements at ELETTRA and are therefore unique. The structure file appears twice - there is one for the transfer line and one for the storage ring.

The way the files are related and read from the application programs is shown in figure 2. The Calibration, the Parameter and the Access File are direct access files. The individual records are selected via keywords. Usually, the Structure File is read - via the initialisation routine - sequentially, one record after the other, in this case actually line by line, as each line contains information about one element and thus corresponds to a record. The Structure File records contain references to records of the Access File and to records of the Calibration File. The Access File itself may contain further references to the Parameter File and Calibration File records. In special cases (RF, tune measurement, etc.), where the equipment is not related to a specific point in the structure and hence does not have a record in the Structure File, the application program reads the Access File directly.

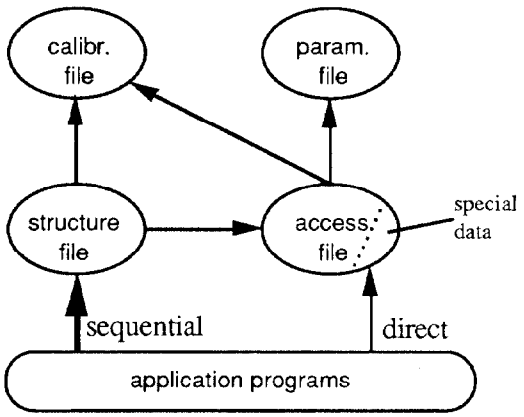

Figure 2: The relation among the data files and the application programs.

B. The Run-Time Data Structure

The high level software (HLS) run-time data structure is a global data structure implemented in the language C, which contains all important parameters that describe the ELETTRA transfer line and storage ring from the machine physics point of view. Its hierarchical structure provides a logical and

efficient access to all the data from within application programs.

The run-time data structure is based on a complex tree structure with many branches and back-references. It consists of a multitude of data records declared in C as *structs* and references to and among them declared as *pointers*. There are several groups of pointers which allow different methods of access to the data records, depending on the requirements.

Each element in the line (transfer line or storage ring) has a corresponding `general_record`, which contains general data about this element, as well as a corresponding *specific* record, which contains data depending on the type of the element. The specific records are `drift_record` for drifts, `magnet_record` for bends, quads and sextupoles, etc. The data which is relevant to several elements, such as current of a power supply which drives several magnets, is stored in the `family_record`. The records contain all fixed data that describes an element and the variable data that has been acquired from the equipment through the control system.

To each element in the structure there is a corresponding 3x3 horizontal and vertical transfer matrix and a `twiss_record`, which contains the Twiss functions beta and alpha, the phase and the dispersion and its derivative in both planes at the position of the element. The list of records includes also the `access_record` containing control system related data from the access file, the `parameter_record` which is read from the parameter file and others.

Access to records is provided via pointers. One array of pointers is pointing to each element's `general_record`, ordered according to the positions of the elements in the structure file. Another access to elements is possible through the *specific* pointers, which point to *specific* records, declared for each type. Records themselves contain various pointers, allowing access of data through several levels of the data hierarchy. Several records may have pointers to a record with common data. Pointer links are bi-directional where this is useful for the programmers.

C. The High Level Utility Routines

Actions which are common to many programs have been gathered in the set of utility routines. They can be divided into two groups: machine physics calculations and actions on the machine. The routines which perform actions on the machine do not communicate with the control system directly in order to keep the explicit references to the control system at a minimum. Instead they register the individual actions via the run-time data structure and call a special routine which handles all control system specific tasks.

Thus there are three layers of utility routines (see figure 1), two of which are accessible by the application programmer: the *control system layer*, the *machine action layer* and the *machine physics layer*.

The control system layer has been added only to allow transparent access to the control system via one well defined point. The advantage is that individual programmers do not even have to care of the explicit implementation of the control system. The control system had been developed in parallel and all changes could be easily added to the routines of the control system layer without changing any line of HLS code. As a side product, it was easy to replace the control system layer with a machine simulation program in order to test each application before releasing it for use on the real machine [3].

The machine action layer contains the most routines. It includes routines which read/set one single power supply current, switch on magnets in a pre-defined way, routines which act on all power supplies of a given type, routines which close or open the scraper, read the current and calculate the lifetime, read the current state of the machine into the run-time data structure, save the run-time data-structure to a machine file, etc. The input and output of these routines do not include any explicit reference to the actual control system.

A set of conversion routines converts the machine parameters like magnet currents, insertion device gaps, etc. to machine physics parameters like quadrupole strengths, ID strengths and vice versa. The machine physics parameters are being acted upon by machine physics routines, most notably the routine `update_twiss`, which calculates the transfer matrices and the linear model of the machine in one step. For a HLS program to use the model of the actual state, all it takes is a call to the routine `input_all` which sequentially reads all magnets, converts the currents into strengths and finally calls `update_twiss`.

III. CONCLUSIONS

The high level software (HLS) data structure contains all necessary items for machine physicists to quickly prepare HLS applications. The usefulness of the data structure became evident by the success of the HLS which was developed in a short time and proved to be very stable. The run-time data structure and the utility routines are generally usable on any accelerator. Thus the HLS written for ELETTRA can easily be ported to any other machine.

IV. ACKNOWLEDGEMENTS

The author thanks the following persons for their contributions: C.J. Bocchetta wrote the basic version of the control system communication routine, F. Iazzourene provided the data for the data files, R. Nagaoka wrote the basic version of update_twiss and L. Tosi wrote the conversion and linear optics routines for insertion devices. Fruitful discussions were held also with E. Karantzoulis, M. Lonza, R. Richter, C. Scafuri and A. Wrulich.

V. REFERENCES

[1] ELETTRA - Conceptual Design Report, Sincrotrone Trieste, April 1989.

[2] D.Bulfone, Status and Prospects of the ELETTRA Control System, Proc. ICALEPS 93, Berlin 93.

[3] M.Plesko et. al, The High Level Software of ELETTRA, these Proceedings.