assembly language. OCCAM, a real-time programming language specifically designed for transputer architectures should also be mentioned.

ADA is unique because it supports structured programming and some of the latest methodologies such as object-oriented design. This language also contains special constructs for embedded system programming (e.g. parallel task exception handling, intertask communication, rendezvous task synchronization, *etc.*).

External Communications

A real-time system communicates with the outside world via three types of services: input/output (I/O), networking and the man-machine interface.

Input/output services of specialized types must generally be added to extend the scope of the standard I/O support provided by the real-time operating system. These modifications can cope with specific requirements of the application program by, for example, permitting specific device drivers to be imported.

Networking allows communication with other systems along local area networks (LAN) or wide area networks (WAN) and the activation of important remote operations such as booting, diskless use, virtual terminal, file access, *etc.* Open, standardized protocols for communication such as OSI of the International Standards Organization must be adopted to ensure easy evolution and implementation and independence from a supplier.

The *man — machine* interface permits, *via* proper console displays, interaction at a high level with the real-time system. One can therefore readily perform operations like program development which require the down-loading of real-time programs into a front-end processor from a host computer for remote debugging or execution of a process. Graphics support and the display of several windows on a console display employing standards such as X Window may be helpful.

Software Development Environment

Cross-development is commonly used for distributed system software. It consists in creating application code on a computerized system called the *host* system and executing this code on a *target* system located in a different computer. This approach is adopted because the target system has been slimmed down to the essentials to maximize performance when running the application. The host environment, meanwhile, is better equipped with powerful tools and various utilities for preparing the application code.

Within the host environment, it is desirable that:

– productivity be sought as far as possible during system development;

– the host and cross compiler derive from the same basic technology (or *root*) to avoid discrepancies between the program codes they generate. An additional advantage is that the user need only learn one type of man — machine interface;

– a *remote debugger* for source code capable of reaching the target through the network should be available. Combined with multiwindowing information display facilities, this will ease the control of program execution and the isolation and tracing of software errors;

– facilities to *simulate* the target environment be provided to ease the testing of embedded programs, and thus to be independent of target hardware such as timing and I/O devices.

Software Engineering

The main objectives of software engineering are significant improvements in software quality and productivity with reductions in the cost of maintaining and modifying programs. They can be achieved using different methods including structured or object-oriented approaches which are invoked by computer-aided software engineering (CASE) tools. A wide variety of CASE tools, based on workstations or personal computers, are now available to cover the major phases of the software life-cycle including analysis, design, coding and testing as well as information modelling and documentation. Real-time features such as control flow graphs, state transition diagrams, process activation tables and state-event matrices expand CASE tools capabilities to support interactive and concurrent processes.

Standards

A key concern is the minimization of investment over the life of a system while improving its reliability. One possible approach is to make use of acknowledged industry *(de facto)* or international open standards to allow: an industrial approach to software development; time saving and reliability; portability (*i.e.* being independent of the hardware); maintainability; the sharing of experience and products (*e.g.* collaboration between laboratories and industrial plants). An example of an international joint effort to propose standards for real-time software is the Open Real-Time Kernel Interface Definition (ORKID) project uniting industry and research.

Conclusions

Implementations of real-time systems for local intelligence are generally at variance with the technical requirements discussed above. This has shown up in a study of the types of real-time systems that are being used by institutes belonging to the EPCS Group. There clearly exists a wide variety of systems that mix industrial and in-house developments. This very undesirable situation persists because implementations depend on local constraints. It can be improved by the "agora" activities of organizations such as the EPCS Group where common issues of importance can be discussed to trigger solutions along some principal directions.

FURTHER READING

Proc. Europhysics Conf. on Control Systems for Experimental Physics, *CERN Report 90-08*, Ed. B. Kuiper (1990).

Proc. of the Workshop on Real-Time Systems for Microprocessors, Chamonix, France, 1989.

Proc. Int. Conf. on Accelerator and Large Experimental Physics Control Systems, *Nucl. Inst. and Methods* **A293** (1990).

Databases In Physics Control Systems

D.P. Gurd

TRIUMF, Vancouver, Canada

One or more databases lie at the heart of modern computer-based control systems used in large-scale experiments. While ensuring that all the computers and subsystems reference consistent data, they also enforce a straightforward discipline for making changes and provide a reliable source of system documentation. I shall describe these databases and discuss the critical issues and compromises facing their designers.

David Gurd recently joined the Controls Group, SSC Laboratory, TX, USA as its Leader having spent 20 years at TRIUMF in Canada. He received his Ph.D. in nuclear physics from the University of Alberta, Canada.

Integrated Technical Database

A complete project description in the form of an integrated technical database should underlie all phases of an accelerator or other large experimental physics project. This database is used for design, simulation, cost estimating, procurement, construction, and eventually for operation and management. The approach was used for CERN's Large Electron Positron Accelerator (LEP) and will be adopted at the Superconducting Super Collider (SSC) in Texas, USA and at the KAON accelerator planned for Vancouver, Canada.

Specialized databases including run-time databases required for accelerator control are extracted from the integrated

technical database (Fig. 1). The designs of the experiment and the control system are therefore intimately connected so modelling and simulation parameters required during the design phase are the same as those used for operation and tuning; changes in the first show up automatically in the other.

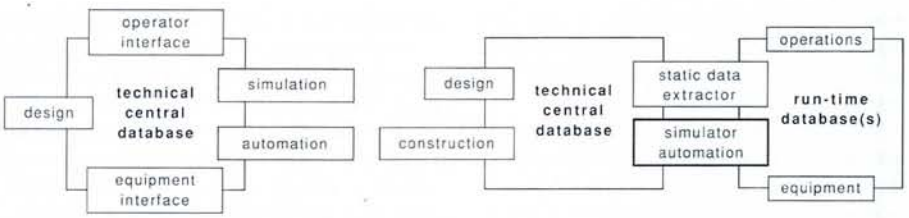

Fig. 1 — Illustration of database implementations. (a – left) The run-time database (distributed or not) is the vital link between the application level (operation) and equipment control. It is individually designed for each control system implementation, as are the corresponding access routines. Some implementations exploit a central, commercially available DBMS, based on standard query languages (SQL), from which they derive the static starting copy of the run-time database. Simulation and automation routines can access both databases.

(b – right) The configuration of the ideally distributed database that could be implemented if an adequate management and access system were available.

Descriptive Database

The descriptive database of accelerator devices, which is generally referred to as the control system database, should originate as an extracted subset of the integrated technical database. It is generally structured around channels, each channel being an analogue or digital signal, so the size of the database is related to the number of channels (Fermilab's Tevatron accelerator has about 50000 channels, while the SSC envisages as many as 140000).

Hardware

The descriptive database of accelerator devices contains all the information describing experimental components (device name and mnemonic; bus or network addresses of controlling modules; parameters for routines required for control; data acquisition; data conversion). In the case of accelerators, this information has necessarily been organized in control systems long before formal database theory was applied. Examples are the "equipment modules" of the control system of CERN's PS accelerator.

Some of the descriptive database entries define properties of equipment items (*e.g.* magnets, beam monitor instruments, radio frequency cavities); others describe devices used to operate this equipment (*e.g.* power supplies and ion gauge vacuum controllers) as well as those controllers which are generally considered to be a part of the control system itself (*e.g.* analogue-to-digital converters, function generators).

Soft channels

In addition to the description of individual channels in terms of hardware, the descriptive database links devices to form higher level entities, or soft channels (*e.g.* a "quadrupole triplet" in an accelerator). At a higher level still, the database contains machine physics information not required for routine control functions, but used for simulation or modelling, or for high-level control functionality such as "move the focus 3 cm down the accelerator beam pipe". It must therefore include detailed data for individual components (*e.g.* the physical location of a quadrupole, its magnetic length, fringe field effects, imperfections, skewness). Information linking many individual components (*e.g.* that required to make chromaticity or closed-orbit corrections to the beam) may also be included.

DBMS

The descriptive database must provide many different "views" of the accelerator control system. Important at this level are managerial features such as data integrity, convenient on-line back-up, adaptability and auto-documentation. These requirements can be met by large commercial database management systems (DBMS) so the generally adopted strategy is to have a DBMS running on a central control system "database computer".

The DBMS manages and maintains all descriptive data from which the efficient and specialized run-time databases are generated and distributed. These run-time databases are, in some cases, the only permitted means of communicating between high-level system processes and low-level tasks interfaced directly to devices. Correct design of the DBMS is therefore important and this has been recognized by the controls community which is now attempting to standardize the minimum requirements and specifications.

Live Databases

The descriptive database must be accessed in real time in response to operator actions, control system algorithms and operating conditions. The large DBMS generally cannot satisfy this requirement so specialized and efficient, live *static* databases, often in the form of simply structured, easily accessed ("flat") tables, are extracted and distributed throughout the system for *real-time* access.

The control system run-time database describing the current state of the machine combines the static run-time database and a live run-time database. It specifies the digital and analogue statuses of hardware devices, of soft channels derived from basic physical measurements, and of overall features of the process (*e.g.* beam characteristics at various locations around the accelerator ring and at different times in the machine cycle).

The control system run-time database is constantly changing with the operating conditions. It describes, on a microscopic level, the state of the machine at a specific time within a cycle (a "snapshot") and, on a macroscopic level, the prevailing timing structure and programme. The operator interacts with this database, which is the origin of the information displayed on consoles and used by all applications.

Fast access is the dominant requirement for the dynamic database so data must be in a predictable format that can be used by any of the standard data access tools available to application programs and to operators. The manner and degree of data distribution is also a critical design issue.

Other Databases

Real-time information

In addition to the information distilled from the integrated technical database, the control system must access in real time many other types of data from easily accessible databases. These data include operation limits for security systems, messages for alarm and error conditions, and text and possibly colours associated with specific digital states (*e.g.* green for normal operation). Information describing console devices such as knobs, screens, buttons, touch panels and their controllers and addresses is also needed.

Reference information

Another database, which may form part of the integrated technical database, contains reference information such as serial numbers for equipment, module type, cabling information, physical location, installation and modification dates, fault reports (with reports of remedial action), *etc.* This information can be used for the production of manuals and other documentation, and for system maintenance.

Machine parameterization

Falling into a completely different category are databases that contain partial or complete parameterizations of the experimental complex or its subsystems. They contain information required to reproduce process conditions and tuning or opera-

ting modes (*e.g.* the set points of a magnet, the positions of diagnostic probes, timing functions). Measurements of process parameters used with closed-loop algorithms to recreate settings may also form part of this database (*e.g.* the profiles and emittances of an accelerator beam).

These databases grow whenever an operator saves the machine conditions so they can become very large. High-capacity storage media may be required together with accurate file management procedures to facilitate retrieving a particular set of operating conditions.

Logs

The parameterization database is in fact one form of data log. Many other logs may be stored in the control system, including separate logs for information, error messages, operator interventions, network use, *etc.* None are of value without database management tools for ready access, sorting, correlating and reporting.

Distributing Databases

Features of the data flow — its origin, users, controllers, transfer rates, frequency, *etc.* — determine data distribution. There are clearly conflicting requirements on the various types of databases: some are accessed infrequently while others must be distributed among several geographically separated sub-systems and yet provide fast access for real-time control. Whatever the system design and the topography of its communications network, the distributions of the static and live databases will be set by network loading and the required response times.

The desirability of retaining some information in more than one place, must be weighed against the importance of keeping data consistent throughout the system. The run-time descriptive databases are therefore generally downloaded into processors located as close as possible to the equipment they describe. The exception is physics information used for high-level calculations (*e.g.* closed-orbit corrections and beamline transport adjustments for an accelerator) which should probably not be widely distributed but brought close to the large number-crunching computers charged with modelling tasks.

To illustrate some of the differences, consider simplified models of three relatively small accelerator control systems (Fig. 2). At TRIUMF, live data remain in CAMAC registers close to their sources and are never collected or converted from raw values, except at the specific request of a high-level application program. The run-time static database is located centrally and is available to all applications. The MIT — Bates Pulse Stretcher Ring exploits the opposite approach. Data are routinely broadcast by low-level sources and accepted only by interested higher level processors that need look no further than their own memory for live data to run an application program [1].

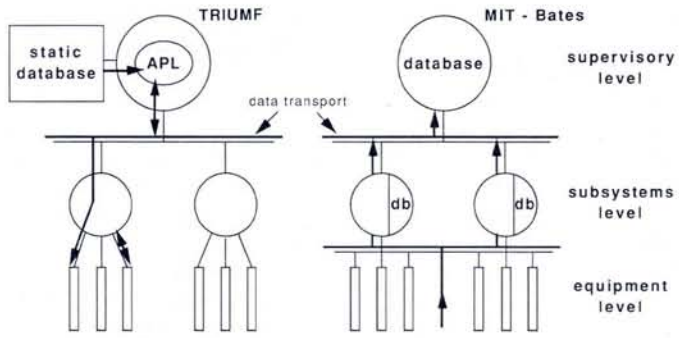

Fig. 2 — Illustrations of the data flows implemented at the TRIUMF accelerator complex (left) and at the MIT-Bates Pulse Stretcher Ring (right). At TRIUMF, live data are memorized in the modules/processors at the equipment level: application programs (APL) use data from the static database and live data from the front-end memory. Dataflow is initiated by the application programs. At MIT-Bates, on the other hand, front-end microprocessors broadcast all their live data to the network(s). They are picked up by the appropriate higher level processors and stored in local memory (their specific run-time databases) where data can be read when needed.

The LAMPF/PSR control system exemplifies a third approach: all live data is gathered in a central repository [2]. Applications talk only to the repository, which is filled asynchronously from lower level data sources. All operations pass through the live database that acts as a buffer between the applications and intelligent equipment controllers.

Future Trends

Databases and their distribution must be carefully designed to maximize the efficiency of database interactions. Recent experience, notably in designing LEP and two of its large detectors, has shown the value of modern software methodologies, especially the *entity-relationship model* as applied to database design. The model connects database entities according to relationships between specific data items (on-to-one, one-to-many, many-to-many). There also exist some computer-assisted software engineering (CASE) tools for automating database design based on entity-relationship drawings which visualize the items and their relationships by *e.g.* bubbles and arrows, repectively. The language and techniques of *object-oriented programming* are proving particularly relevant in designing databases for a control system, a process which simply involves organizing equipment-type objects. Finally, *artificial intelligence tools* can help one design the correct database.

REFERENCES

[1] Flanz J. *et al.*, Proc. Particle Accelerator Conference, Chicago (1989) 34.

[2] Clout P. *et al.*, *Nucl. Inst. and Methods* **A247** (1986) 116.

Integrated Project Support

H.E. Keus

Westmount Technology, Delft, The Netherlands

Large experiments in physics require the integration of complex and precisely matched hardware and software components into real-time control systems.

The critical nature of these projects demands extensive systems engineering, configuration management and quality control by multidisciplinary teams working in both the development and target environments. Requirements for integrated project support are starting to be satisfied by integrated computer-assisted software engineering (CASE) tools. Uses include projects with embedded computer applications (ECA) where one integrates unconventional hardware components controlled by dedicated software.

H.E. Keus has a Ph.D in nuclear physics from the University of Leiden. After 6 years of research on information processing with the Dutch post and telecommunications services, he joined, in 1987, Westmount Technology bv, POB 5063, NL-2600 GB Delft where he is responsible for strategic developments.

Developing An ECA

The first stage in the development of an ECA usually involves an analysis of the entire real-time control system to end up with a set of defined subsystems containing hardware and software components. These two components are then developed independently with support by computer-aided engineering (CAE) and CASE, respectively. Hardware engineering involves the target (the actual system performing the real-time task) while software development proceeds in a host — target environment. The two parallel streams