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].

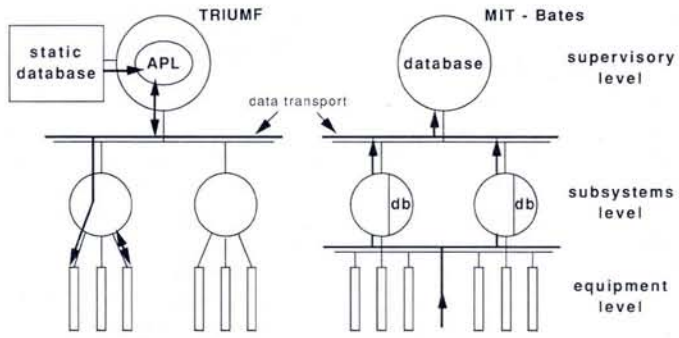

Fig. 2 — Illustrations of the data flows implemented at the TRIUMF accelerator complex (left) and at the MIT-Bates Pulse Stretcher Ring (right). At TRIUMF, live data are memorized in the modules/processors at the equipment level: application programs (APL) use data from the static database and live data 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

merge at the integration stage, when both environments must be available simultaneously and we have to deal with multi-user host/target testing as well as debugging of a real-time system. Unpleasant surprises, *e.g.* incompatible timing constraints, can be avoided by applying simulation early on in the analysis and design stages of development using integrated CASE tools.

Analysis

A popular structured development technique [1, 2] for ECA's involves the generation of an *essential model* which is a full description of what the control system must do without taking account of implementation constraints (failures, finite operation speed, *etc.*). Activities are then mapped to various combinations of hardware (processors, devices, *etc.*) and software (tasks, interrupt routines, *etc.*) components in the subsequent *implementation model*. Choices have to be made depending on technical and economic requirements. The final model gives the detailed design of the software components so it specifies how the system works.

Analysis for the essential model starts by defining the boundary of the system, the objects in its environment and the events it must respond to. Graphical representations using CASE tools help make the model understandable to the specialists who are involved (system analysts, users, engineers, *etc.*). Division of the system into processes, each describing an essential data or control transformation, and the definition of the response to external events result in numerous description files, diagrams and specifications. The analysis itself is therefore under control because these outputs can be comfortably browsed through, sorted, cross-referenced, inter-related and reported upon using CASE tools.

The tools also ensure the quality of the overall analysis by providing facilities for checking syntax, completeness and consistency. All specifications and system requirements are stored in a central project repository with clear-cut identification for ease of retrieval.

Simulation

An ECA's response to external events is visualized by placing a token on the appropriate element of a data flow diagram. Token simulation using the activation of tokens in various combinations can be performed to look for deadlocks, starvation, closed loops, *etc.* The effects of events are calculated if the system is fully specified. Simulation, more significantly, checks for predictabiliy, *i.e.* that the system's end state depends only on the state at the beginning and on the event — a vital property of a correct model and one that can be established as early as the analysis and specification stage using integrated CASE.

Architecture

The essential model must be organized in such a way that any asynchronous processing is identified and separated out. Choosing a solution between, at one extreme, a single general purpose processor and, at the other, a special device for each process will be guided by implementation constraints (existing or prescribed components, location of data acquisition, fault tolerance, interfacing activities, *etc.*).

For a system composed of special devices and general purpose processors, the highest level of architecture in the implementation model is the *processor model*. Integrated CASE tools contain graphical editors that support the manipulation of the processor model and the identification and partitioning of sub-systems which can be modelled separately.

The resources provided for scheduling, communication, synchronization, *etc.* in a multi-task operating system determine the program code and workspace requirements. Implementation constraints are once again important so it is useful to formulate a task model of a general purpose processor's software architecture.

Simulation can be used to analyse the performance of the architecture, albeit in a more quantitative manner than for the analysis stage. Apart from the overall behaviour of the system being simulated, the internal execution times are crucial. Simulations are performed with different times for each processor and modelled device to identify bottlenecks. The effect of changing the allocation of processing by moving software tasks to special purpose hardware is studied in an iterative fashion. Design errors (*e.g.* deadlock, mutual exclusion) are identified and the architecture is tuned by reassigning time-consuming tasks or by specifying a faster processor or another real-time kernel (see page 34).

Once software has been defined, tools for its detailed design are exploited. Techniques include structure charts, program definition languages and object-oriented programing. Code generation, for instance, is then possible using structure charts.

Implementation and Testing

Software implementation for ECA typically means cross-development on a host computer and testing on a remote target. The host provides a powerful development environment with editing, debugging and compiling facilities incorporating advanced CASE tools enabling programmers to quickly compile, download and test programs in the target. The target therefore needs programs to allow downloading from the host and the generation of system status information during implementation and testing. These programs provide a "remote support task" in the target and they should run with the same stripped down, high-speed kernel as the final production environment.

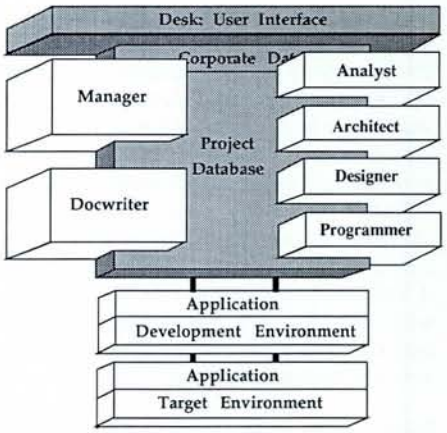

Fig. 1 — The architecture of a modern integrated project support environment that provides a set of integrated computer tools for analysing, designing, testing and implementing real-time control systems to operate in the target environment. Workbenches providing specific engineering and support tools to the development are integrated for display and processing and share a project database.

Integrated Project Support

Integrated project support environments provided by second generation CASE tools typically have the structure (shown in Fig. 1) characterized by a set of tools (called *workbenches*) for systems engineering and project support. Integration of all workbenches is on at least three levels (presentation, data and process) using graphical man — machine interfaces (X Window, OSF/Motif, *etc.*) common to all workbenches, data integration based on a central repository, and processing in a multi-user hardware environment comprising a network of workstations, minicomputers and the like.

Conclusions

Sets of integrated CASE tools with dedicated functionalities now exist to support a rigorous development methodology for a real-time control system based on embedded computer applications, where hardware and software must be precisely matched. The system can be simulated at early stages in its development to detect specification and design errors. Graphical design editors combined with code generation improve productivity as well as the consistency of system documentation; version and configuration management tools enhance quality control.

REFERENCES

[1] Yourdon E., Modern Structured Analysis (Yourdon Press) 1989.

[2] Ward P.T. and Mellor S.J., Structured Development for Real-Time Systems (Yourdon Press) 1986.