ALEPH 87-16

DATAQ 87-2

J. Harvey et al.

6.2.1987

January 30 1987

**Use of Software Engineering Techniques in the
Design of the ALEPH Data Acquisition System**

T.Charity, J.Harvey, R.McClatchey

(Presented at the Computing in HEP Conference at Asilomar - 1987)

Use of Software Engineering Techniques in the Design of the ALEPH Data Acquisition System

T.Charity and R.McClatchey
CERN

J.Harvey
Rutherford Appleton Laboratory

presented by T.Charity, EP Division, CERN

Abstract

The SASD methodology is being used to provide a rigorous design framework for various components of the ALEPH data acquisition system. The Entity-Relationship data model is used to describe the layout and configuration of the control and acquisition systems and detector components. State Transition Diagrams are used to specify control applications such as run control and resource management and Data Flow Diagrams assist in decomposing software tasks and defining interfaces between processes. These techniques encourage rigorous software design leading to enhanced functionality and reliability. Improved documentation and communication ensures continuity over the system life-cycle and simplifies project management.

1. Introduction

The ALEPH detector is being prepared for operation at the Large Electron Positron (LEP) collider currently being constructed at CERN and is due to begin data taking in 1989. The complexity of the detector and its associated data acquisition system [1] poses severe problems for the design of the online software needed to operate the equipment. In this paper we outline the approach we have taken in tackling these problems.

Some of the principal features of the data acquisition system may be summarised as follows. The readout system consists of some 700 000 electronic channels producing events of up to 100 kbyte at a rate of 10^7 events per year. A distributed system of microprocessors and VAX computers is used for data acquisition and control of the experiment, communicating via Fastbus and local area networks. The Fastbus system is a segmented, high speed bus for establishing communication paths between the intelligent processing modules used to digitise, collect and process raw signals coming from the detectors. The possible ways in which Fastbus devices can be configured is complicated as are the rules governing the passing of operations between devices. In addition, a second class of microprocessors is used to set and survey the detector operating state

(Slow Control). A system of partitioning has been devised to permit independent data-taking activities to proceed so that several groups can be working on different parts of the detector at the same time. For this system to be operated successfully access to all online resources has to be controlled to ensure that the partitions active at any time do not interfere with one another. A further requirement is that the data formatting and processing performed by the acquisition system must also be compatible with the other software components of the experiment, in particular the event reconstruction and simulation programs.

Project management must ensure that the software effort may be easily divided, and that new members of the design team may be quickly integrated to cope with the changing manpower situation. This implies a commonly agreed method for communicating problems and assessing progress in the different areas of system development. A further requirement of the online environment is the necessity to test new pieces of hardware as soon as they are commissioned and so prototypes of the overall system must be quickly available to provide a framework for these tests. These activities must continue in parallel with the overall design of the desired system, to allow a smooth transition from the initial specification of requirements through to the implementation of a robust and manageable product over the lifetime of the experiment.

The Structured Analysis / Structured Design (SASD) methodology has been adopted by the collaboration to provide a framework for software development in which the problems outlined above become manageable [2]. The Structured Analysis and Structured Design methods [3] have been found to be useful in describing the principal activities required by the system, and the requirements of a control system have been met with additional methods which are applicable in real-time applications [4]. The complexity of the hardware configuration and readout components has been analysed using data-modelling techniques, to describe the underlying structure of the data. Finally the detailed design and packaging of the system has provided a method for allocating the tasks in a distributed environment.

2. Analysis of the Data Flow

The Data Flow Diagram (DFD) in Figure 1 represents the entire software system, from the detection of particles through to generation of reconstructed events and their subsequent physics analysis. The context of the data acquisition system is represented by the process 'Record Data from ALEPH', which takes particles and simulated data as input, and generates event records as output to the online data store.

The diagram illustrates several major advantages resulting from the use of Structured Analysis. All the major software projects within ALEPH (data acquisition, reconstruction, simulation, and analysis) are represented, and their compatibility is ensured by definition of the data flows linking the processes. One central database is used for storage of all information concerning the detector configuration. This provides consistency between different parts of the software, and eases the problems of maintenance of a large number of parameters. Common data structures are defined throughout the system to simplify communication and transfer of data between the

processes.

Figure 2 illustrates how the data acquisition system may be represented by part of an expansion of the process 'Record Data from ALEPH'. The resulting diagram is typical of a conventional dataflow pipeline, consisting of a series of processes progressively modifying a data stream. The pipeline formats and classifies the incoming data streams, allowing monitoring and selection of events and subevents at different stages of the DFD. An important aspect of the diagram is that the physical implementation and packaging of the processes is undefined at this stage, where we are concerned only with logical operations. The design and implementation of such processes which perform successive transformations on the data uses the technique of 'transform analysis' to derive the program structure from this logical model. This technique results in a highly modular and hierarchical structure where the input and output branches are cleanly interfaced to the central processing modules. Such systems are generally more robust to changes, and are easier to implement and maintain.

3. Control and Interaction

In the first example no time dependency is indicated on the diagram in that it is tacitly assumed that the presence of input data is sufficient to activate the dataflow pipeline. However in a data acquisition environment the state of the whole system can change due to the detection of a particular event in the system's environment and this can be made to affect the way the system subsequently behaves. A simple example of this would be the placing of the system in a disabled state following the detection of loss of high voltage on one of the detectors. It is extremely important for the technical methods to handle these features correctly as they have a major influence on the behaviour of the system.

This situation typically occurs in interactive applications where the operator can change the state of the system by the press of a button. Figure 3 shows an example of a DFD describing that part of the system for providing operator control over data taking activities. As can be seen the events are represented by dotted lines to distinguish them from normal data flows and the control transformation which handles these events ("Invoke DAQ Action ") by a dotted circle. The program structure derived from diagrams of this type follows a well-defined pattern and is illustrated in the structure chart of Figure 4. On this diagram the control transformation appears as a transaction centre which must keep track of which state the system is in and use this information together with the input control flow to determine which action procedure to activate.

The control logic for implementing the transaction centre can be extremely complicated and experience has shown that it is in this area that those errors which are most difficult to detect are introduced into the code. However the State Transition Diagram (STD) provides an ideal technique for specifying this control logic as it provides a method for representing the states of the system, the conditions for transitions between states to be made and the actions to be performed on making the transitions. The STD for our data taking control system is shown in Figure 5. The information stored on the STD can also be represented in terms of a table (Figure 6). In this form it may be used

together with some standard control logic as a basis for automating the implementation of the transaction centre. Another advantage of the tabular representation is that it forces the designer to consider the possibility of each condition occurring for every state of the system.

4. Modelling the Readout System

Certain composite dataflows appearing on the DFDs can be extremely complicated and likewise certain data stores take on more the character of a database. Decomposition of these DFDs then becomes difficult without a thorough understanding of the content of the data being dealt with. To solve this problem the Entity-Relationship model has been used to provide a comprehensive description of the data items and the relationships between them.

The online system can be subdivided into several related subschema for data modelling (Figure 7). These subschema include descriptions of the readout components (Fastbus), the detector control components (Slow Control), the data taking environment (Run and Partition descriptions) and the overall detector description. Figure 8 shows an extract of the Fastbus data model describing the physical layout and addressability of Fastbus modules. The design of the complete Fastbus subschema has required a full description of the Fastbus standard [5]. One of the major advantages we have found of using this approach is that the resulting data model can be analysed in detail to see if it correctly interprets this standard.

Once an accurate model of Fastbus has been constructed it becomes simpler to design the general purpose software required for the initialisation of a Fastbus system. Figure 9 shows a subset of a structure chart used in the initialisation software design. In this figure the interface to the database is shown as a collection of procedures which control access to and ensure consistency of the database contents. These procedures isolate the operations to be performed on the database and thereby hide the details of how the data are stored. In this way the access routines can be designed and implemented independently of the rest of the initialisation software. The data model together with a list of valid operations to be performed on the data provides sufficient information for the design of the access procedures.

As already mentioned the data model describing the online environment has been merged with a complete description of the detector and a model describing the offline data environment to provide a database for use throughout the experiment. This centralised database avoids duplication of data and provides a mechanism for common access from online and offline software. In addition, as this data model has been designed using an Entity-Relationship approach it maps conveniently onto a tabular form. This has allowed us to profit both from the use of a commercial relational database management system, such as Oracle, and from the use of a specially designed package, ADAMO [6], which can be used to map Entity-Relationship structures onto tables and manipulate them from FORTRAN programs. These two packages are complementary in that Oracle provides sophisticated multi-user control, data protection and consistency checking whereas ADAMO provides the rapid access to data (stored in direct access files) required by the online software. The interface between ADAMO and Oracle is described in another paper contributed to

this conference [7].

5. Packaging, Detailed Design and Implementation

The techniques described in the previous section may be used to provide an overall design for the components of the system, often in the form of structure charts or state-transition diagrams. In the online environment it is necessary to test the readout components at an early stage in order to satisfy the requirements of hardware commissioning; consequently a 'bottom-up' approach to implementation of the design is preferable, whereby the low-level modules of the design are coded and controlled by a 'test driver'. This style of implementation is similar to the traditional bottom-up approach to system implementation, but in this case the implementation takes place within the context of an agreed framework. Test programs developed at this stage may then be easily integrated into the complete system without compromising the design goals of the project.

The overall design must be packaged into physical program modules which in the ALEPH online system involves distributing the component tasks to the various computers and microprocessors present in the readout system. Packaging after the analysis phase simplifies the integration of the distributed tasks since the communication between the processes is defined by the dataflows linking them, and by their underlying data structures.

Following implementation of a prototype system it may become clear that the design must be modified to cope with the real-time constraints imposed by the system. These typically include program efficiency and interrupt response latency. Various performance analysis tools may be used to identify regions in the program which require optimisation. The fact that the design approach results in a balanced hierarchical structure of program modules makes it easier to repackage the components in such a way that the constraints are satisfied without compromising the initial design goals. Furthermore the detailed implementation of individual software modules should not need to be substantially changed thus avoiding large-scale rewriting of code.

6. Conclusions

Software Engineering techniques have been used by the ALEPH data acquisition software group for approximately two years. The methods have been found to be extremely helpful in providing a clear understanding of the required functionality of the system and as a means of communicating ideas and suggestions both within the group and to the collaboration as a whole. Designs have been produced for various components of the project, and implementation is well advanced in some areas. However the lack of convenient automated tools for aiding software development has meant that SASD has been used more as a collection of useful techniques for producing reliable and robust software rather than as a rigorous methodology for proceeding from a requirements specification to a validated set of program modules.

7. Acknowledgements

We wish to thank many of our colleagues for their various contributions, for introducing these techniques, for participating in the design and for reviewing the status of the software project.

8. References

1. I.Videau, The Data Acquisition System for ALEPH,
IEEE Trans. on Nucl. Sci, Vol 4 NS-32, No 4, August 1985
2. G.Kellner, Development of Software for ALEPH using Structured Techniques,
presented at this conference
3. M.Page-Jones, The practical guide to Structured Systems Design,
New York: YOURDON Press (1980)
4. P.T.Ward and S.J.Mellor, Structured Development for Real-Time Systems, Vols 1, 2 and 3,
New York: YOURDON Press (1985)
5. IEEE Standard Fastbus Modular High Speed Data Acquisition and Control System,
ANSI / IEEE Std 960-1986
6. R.Brazzioli, S.M.Fisher, P.Palazzi, W.R.Zhao, The Architecture of the ADAMO System,
ADAMO note 4, Nov. 1986
7. Z.Qian et. al. , Use of the ADAMO Data Management System within ALEPH,
presented at this conference by P.Palazzi

Figure 2 Data Flow Diagram showing processes for collecting and monitoring event data

Figure 3 Data Flow Diagram of an interactive application illustrating use of control flows

Figure 4 Structure Chart illustrating structure of the program modules for the interactive application

Figure 5 State Transition Diagram For Specifying the Transaction Centre In the Interactive Application

State Condition	1 Idle	2 Setting Up	3 Datataking Active	4 Datataking Inactive	5 Resetting After Run	6 Finish
Set Conditions	Update RD 1					Action New State
Start	Initialise 2					Action New State
Setup OK		Enable Trigger 3				Action New State
Setup $\overline{\text{OK}}$		Report Error 1				Action New State
Pause			Disable Trigger 4			Action New State
Continue				Enable Trigger 3		Action New State
Stop/Event Limit			Reset 5	Reset 5		Action New State
Reset OK					Disable Trigger 1	Action New State
Reset $\overline{\text{OK}}$					Report Error 4	Action New State
Come Offline	Null 6					Action New State

Figure 6 Specifying the Transaction Centre Using the State Transition Table

Figure 7 : Entity-Relationship diagram for the context of the Online data model

Figure 8 : Entity-Relationship diagram for the Layout subschema of the Fastbus data model

Figure 9 : The packaging of data access routines for the Fastbus Initialisation software (access routines shown in dashed box)