Summary of the Workshop on

A Software Bus Common to Accelerator and Large Experimental Physics Control Systems

held at CERN, 15 and 16 May 1995.

Fabien PERRIOLLAT, CERN, Geneva

Software sharing, more specifically sharing of our type of controls software, recently dubbed SOSH, has been under discussion for a few years and the topic is receiving growing attention at ICALEPCS and at dedicated workshops.

At ICALEPCS'91 in Tsukuba, Peter Lucas chaired a panel discussion on Standards and World-Wide Sharing of Software (ref: proceedings p. 597), there was a conference session on Integration of Industrial Systems to Open Control Systems, which is one important way of sharing, and there were contributed papers on the topic. At ICALEPCS'93 in Berlin, there was the first major reporting on the EPICS Collaboration's success story (ref: proceedings p. 486), Peter Lucas again chaired a panel discussion on Should We Buy Not Build a Control System? (ref: proceedings p.484), Berend Kuiper chaired another panel session on Software Sharing (ref: proceedings p. 501) and there were again contributed papers on the subject.

On the Saturday following that Berlin ICALEPCS'93, a one day dedicated workshop was organized on software sharing. At that workshop, candidate functionality for software sharing and prerequisites for software sharing were identified and gathered in respective draft lists. In a concluding Joint Statement (ref: ICALEPCS'93 proc. p. 513) the importance and basic feasibility were emphasized and the possible key role of a software bus was pointed out. Three recommendations were then formulated: (1) to make documentation and where possible code more generally accessible over the World Wide Web; (2) to push for a working party with the objective of investigating the possibility of identifying and jointly adopting a software bus; (3) to improve and detail the mentioned lists.

The workshop at CERN, held in May 1995, on which I am reporting now, was organized in response to those recommendations. Besides monitoring progress on points (1) and (3), the main objectives were exploring common ground and possibly making a step towards the working party as indicated in point (2). Point (1) had made good progress....several labs already have parts of their controls documentation and software available on the Web. Nothing had happened on point (3), since it remains a bit vague until more progress on point (2) is made. Thus point (2), **the search for a common software bus**, was the dominating topic, as illustrated by the abbreviated workshop program:

Welcome - Tutorial (on software bus principles), Q&A - Charting Labs' interest and involvement in this approach (Presentations by Laboratories FNAL, CEBAF, DESY, VISTA, DAPHNE, KEK, ESRF, BESSY, CERN, ELETTRA, ZEUTHEN, CERN) - Analysis (objectives) - Analysis (functionality) - Ways, stages, hurdles - Conclusions: quit or next step?

As stated earlier, the general notion of the software bus - in one form or another - is alive and well in labs and industries and may be illustrated here by (1) the "Plug-and-Play" model which we all see publicized daily in the PC world and elsewhere and (2) the recently coined, more flexible concept of "Middleware", which in our control systems is most of the software between the user part (operator interface plus applications) and the equipment access. It seems also that, compared with a number of years ago, we in the HEP labs have become a bit more ego-less and that more exchange is taking place while new technologies are emerging. No doubt the most striking example in terms of coagulating power is the EPICS collaboration; but we should immediately add (alphabetically) ACNET, CDEV, CICERO, FactoryLink, LabView, Vsystem, etc. The labs' presentations during this workshop all revolved around very similar issues. The keywords heard repeatedly were Share, Collaboration, Functionality, API, GUI, Formal Definition, Data Model, Client/Server, Object, Device, Network Hidden, Standard, Technology, OOT, CORBA, OLE. So it is clear that the same things are alive in most labs. But it also is clear that much work must be done - both conceptually and organizationally - if one wants to converge to a common software bus.

Obviously the main substance of that May 95 workshop was formed by the two analysis sessions. I shall now shortly touch on a number of issues which were debated and shall liberally mix in my own points of view.

First there was the question **what is a software bus?** One view was that it is an API (application program interface). That API has two faces: firstly the external one, which is seen by all application programs and provides the necessary services for those programs which have a direct control functionality for the accelerator, e.g. changing a magnet current, displaying a closed orbit, etc., and secondly the one which the API shares with the internal components of the control system, e.g. the equipment access, etc. The workshop dwelled more on the external interface.

Second, there is the **I/O model** and there were several views on that. It may be a synchronous one, in which any transaction must be completed before the next one. The other possibility is an asynchronous I/O, which is very large in scope. One aspect is that there must be some synchronization capability, and one approach is a deferred I/O facility in which one sets up the I/O in question and executes that at an appropriate moment, in a way which is closer to the synchronous mode. Also there is the option to add a call-back on completion of an asynchronous I/O (as inspired by the X-Motif example), and of course one needs a timeout.

In our types of control we also need **event driven I/O**, typically related to three types of events. First there are machine and experiment events and second there are time events (e.g. from the machine timing system), both types coming from the hardware and requiring real-time processing. Third there are events created by the software itself and which need to have the same capabilities as the previous ones.

Of course everyone agreed on the need for **good monitoring** capability of a number of parameters in the system, and there was the notion of cache...but I am not at ease with this concept and prefer not to dwell on it.

When speaking about **implementation** there are broadly speaking two approaches, the closed one and the open one. For the **closed approach** everything must be well defined from the beginning, so specification must be complete and nothing should be added later on. There is then a closed set of messages catering for transactions in both directions, which are exclusive read or write, meaning that the direction of the transaction is unique at any one time. This is contrasted with the **open approach** in which there is an open set mainly implemented as RPC (remote procedure call). Here the services which are exported by the server and which can be used by the client are not defined a priori. That is done in various ways at compile time or at runtime.

Then there was the issue of **object identification**. Again, there may be two approaches, the flat structure and the hierarchical one. In the **flat structure** all elements in the system have the same capability - every object can access anything. In the **hierarchical model** there is a set of classes of objects and you do not necessarily have a complete view of everything within the system. It was doubted whether one should have such a facility at the bus level and the feeling was that this type of structuring was more appropriate on the applications level, keeping it out of the middle ware. A related but not much discussed issue was the **name server** facility which I personally think should be built from commercial software.

Some other important I/O issues were not much discussed. **Performance** is of course one of them. Obviously the performance, after adding something to a system, must be at least as good as before. But performance is the sum of **speed and functionality** and there is a **trade-off**. Closely related to performance in distributed systems is the important issue of **connection management**: should a connection be established permanently or must it be volatile. The discussion here was inconclusive. But there was a consensus that a multicasting facility is needed, and that does not seem to be simple.

There was a long and useful discussion about the **data model**. The first point is that a data model within the OO paradigm is not a natural approach. Normally the data are hidden inside the object and are not directly accessible from the outside. But in a highly distributed system there is a need to define exactly which data are exchanged across the network between various computers. Two approaches to this problem were proposed. A static model, where everything is well defined at compile time, which is efficient but inflexible. The other one is the dynamic model where the data are encapsulated in an object with the full description of the structure and the semantics, or even without the encapsulation in an object, but always with the full description.

Another field of attention for the data model was the **error data** case. It was agreed that the error information for any transaction or exchange has to be provided by specific data, the error code, and it seems to be appropriate to provide some kind of specific data model for this purpose. For the time being not too much was elaborated on this subject but it is certain that some standard must be provided to guarantee better systematic error handling in application programs. Thus this remains a field for further reflection.

As for **technology**, we agreed that in principle any proposal for **a software bus should be independent of technology**. That is easily said but not so easy to implement. At the time of that May workshop at CERN, CORBA was not yet available on all platforms of interest to us and there were some doubts about its performance. The situation is not the same today...but the real time performance is still not so clear. On the other hand there is a clear need for a good old API in C, in addition to one in C++, in order to preserve our large software investments. Finally it was emphasized that any implementation must be "thread safe", that is completely re-entrant for reliable multiprocessing.

Other important issues, only touched upon, were the **global model of the machine(s**) and possible standard conventions around that, e.g. the co-ordinate system for MAD. And then there was the issue of **configuration management** which has an interesting approach in the CICERO project.

This May 1995 workshop at CERN ended in somewhat of a surprise, but a very happy one. Given that in accelerator work the overwhelming proportion of the effort (>80%) goes into developing console application code, we all converged to the conviction that **a common API (application program interface) was the most urgent issue**, if our goal be to share accelerator applications software. This problem had already been addressed at CEBAF by providing the CDEV software package which is a framework which can encapsulate an arbitrary existing control system into a standard form -- a sort of virtual control system interface with both C and C++ bindings. The package had been presented by Chip Watson earlier in the workshop. It appeared that, with modest additional work on that package at CEBAF, it would be ready for evaluation in other laboratories. Chip Watson then committed himself to invest the required effort in the coming few months and make CDEV available to all interested (he has done so in the meantime). Thereafter, when prompted, the large majority of persons present committed themselves to evaluate the potential of CDEV in the environment of their control systems and report on their findings at the next SOSH workshop which would be grouped with ICALEPCS'95.

I now finish with some personal comments. I feel that the workshop gave us a very positive impulse and that one should continue the endeavor in the form of an open group. It is very important that there seem to be some immediate outcome. In fact such a club can efficiently help spreading results of experiments with new technology. I should also emphasize that the work of such a club is not in competition with, but is complementary to, collaborations such as EPICS or other groups meeting on particular commercial products. For parties who must produce a system soon, the answers will obviously not come from this SOSH club. They should turn to EPICS or Vsystem or LabView. But if you want to open your mind and consider the medium and long term future of our control problems, then it may be a good idea to join the SOSH club or the proposed special interest group of the OMG or why not both...