A Spectrum of Four Event Data Model Proposals

19-Jan-1998
Rob Kennedy

- Model 1: Minimal Change, complete "missing" pieces only
- Model 2: Same Banks Model, but add support for discontiguous record, robust iterators, and explore objects in record, etc.
- Model 3: Use IDL or Java description of objects to auto-generate skeleton support of persistent representations
- Model 4: Use C++ class-based description of objects to auto-generate support for persistent representations

- Note: The proposals are not mutally incompatible. We might combine aspects of 2 and 3, with a schedule of what to do first, and when (so others can plan). Is anything really held up by this?

Model 1: Minimal Change

This model is the minimum that we already agreed to accomplish:
- Bank Sets for controlling which Banks are input or output
- Bank Keys as supported fundamental storage types in Banks
- Complete support for datarep conversion with TryRoot files
- Integration of AC++/Trybos with the Data Handling system
- Minimal F77 Interface in Trybos for existing prod/sim code
- Create GeometryConstants, move Banks classes to Banks

This model is the minimal set of tasks to enable complete data analysis. Clearly, not the most convenient approach for developers.

Banks Sets and Integration:	1 week on Trybos, ___ on AC++
Bank Keys as storage type:	1 week on Trybos (not YBOS compat)
Datarep conversion TryRoot:	2 weeks on Trybos
AC++/Trybos w/DH system:	? weeks on Trybos, ___ on AC++
Minimal F77 Interface:	2 weeks on Trybos
GeometryConstants and use:	2 weeks on various packages
Design, doc, admin:	2 weeks on Trybos, ___ on AC++

(At Meeting): Root Branch- Trybos Bank coordination support....

Model 2: Same Banks, More Change

This model includes all in Model 1, adds extensions which should simplify development, as well as some exploration/experimentation:

- ➪ Dynamic and discontiguous Trybos Record, robust iterators
- ➪ Revise how Bank Numbers are assigned to Banks (robust links)
- ➪ Support collections on record given how a single element is stored
- ➪ Try to extend this to permit "persistable" objects on record
- ➪ Fuller F77 Interface to support prod/sim cross-check code

This model emphasizes development which leads to the greatest productivity gains. Users still write Banks classes themselves.

(Subtotal from Model 1):	10 weeks on Trybos, ___ on AC++
Discontiguous Records:	3 weeks on Trybos
Revise Bank Number assign:	1 week on Trybos and user code
Support collections:	4? weeks on Trybos
Persistable object on record:	4? weeks on Trybos
Fuller F77 Interface:	2 weeks on Trybos
Design, doc, admin:	2 weeks on Trybos

Model 3: IDL/Java, Autogenerate

This model includes all of Model 2 except collections support and objects in record, but alters how users describe their objects:

- ➪ Users write IDL/Java to describe the interface of Banks objects
- ➪ Create a generator tool which reads IDL/Java and generates header files and source code stubs for persistent representations of objects, and perhaps methods to convert to/from transient/persistent
- ➪ Users fill in the source code stubs with actual functionality
- ➪ Some aspects of the C++ language could not be supported

This model includes well-understood productivity enhancing development, but also involves a change to how users specify their objects... and spreading the development effort across a larger number of people.

(Subtotal from used Model 2):	17 weeks on Trybos, __on AC++
How users will write IDL/Java:	2? weeks, needs to be defined
Create/polish auto-generator tool:	2-3? weeks
Design, doc, admin:	2-3? weeks

Model 4: C++, Autogenerate

This model is same as Model 3, but C++ used instead of IDL/Java:

- ➪ Users write C++ headers for transient objects
- ➪ Use a generator tool (rootcint) which reads C++headers files and generates header files and source code for persistent representations of objects, as well as methods to convert to/from transient/persistent
- ➪ Users may need to fill in a few source code stubs with actual functionality, where data type support is incomplete
- ➪ Users may need to work around incomplete C++ language support in rootcint, such as lack of support for STL...

This model minimizes the effort expended by developers, but uses the full Root development model and rootcint. These are considered a maintenance burden (incomplete C++), and their use adds risk.

(At Meeting) Look at using D0OM, D0 cint, and a Trybos/Root back-end if this direction is taken.

Summary for Offline Meeting?

- ➪ EDM Schedule has slipped 2 weeks, otherwise mostly on track
- ➪ Surveys of current offline and EDM are all in, except Cal/Jet
- ➪ Preliminary EDM proposal in 2 weeks, with *use-cases*. (The current discussion focusses on requirements, technical aspects).
- ➪ Final EDM recommendation in Mid-Feb 1999
- ➪ Decision on EDM by S/w Mgmt by beginning March 1999
- ➪ Complete implementation on high priorities by end March 1999
- ➪ Transition occurs as implementation available, mid-April 1999

- ➪ No aspect of EDM prevents development in other areas of offline
- ➪ A number of aspects of the EDM can improve productivity and robustness in offline development, and should be made available as soon as reasonably possible
- ➪ Current development model to be supported while EDM enhancements and policies are defined and implemented

- ➪ IMHO, we should pursue Model 2 and/or Model 3, with the unselected model acting as an alternate.