Comments on Control/Framework

Craig E. Tull

NERSC - LBNL

ATLAS Software Workshop

CERN - September 1, 1999

HENP Computing Challenges

Experiment	Data	Compute
E895 (AGS)	10 TB/yr	600 SPECint95
BaBar (SLAC)	400 TB/yr	5,000 SPECint95
STAR (RHIC)	266 TB/yr	10,100 SPECint95
PHENIX (RHIC)	700 TB/yr	8,500 SPECint95
D0 Run II (FNAL)	280 TB/yr	4,075 SPECint95
CDF Run II (FNAL)	464 TB/yr	3,650 SPECint95
ATLAS (LHC)	1100 TB/yr	2,000,000 SPECint95

Experiment	Countrie:	Institutes	Collaborators	Time Frame
E895 (AGS)	3	12	49	2000
BaBar (SLAC)	9	85	600	2010
STAR (RHIC)	7	34	400	2010
PHENIX (RHIC)	10	41	400	2010
D0 Run II (FNAL)	11	77	500	2005
CDF Run II (FNAL)	8	41	490	2005
ATLAS (LHC)	34	144	1700	2015

Realities of ATLAS Computing

- Large Data Volume
- Large, Globally Distributed Collaboration
- Long Lived (>15 years) Project
- Large (>2M LOC), Complex Analyses
- Distributed, Heterogeneous Systems
- Reliance on Commercial Software & Standards
- Evolving Computer Industry & Technology
- Object Oriented Programming
- Legacy Software
- Legacy Software Programmers
- Limited Computing Manpower
- Most Computing Manpower are not Professionals

What is Control?

- Lassi’s definition: The control is the part of the infrastructure that makes sure that
  - —The right piece of software
  - —Runs
  - —At the right time
  - —With the right inputs and
  - —The outputs go to the right place

What is a Framework?

- **Gamma, et al., <u>Design Patterns</u>**

"<u>When you use a toolkit, you write the main body of the application and call the code you want to reuse. When you use a framework, you reuse the main body and write the code it calls."</u>

"Not only can you <u>build applications faster</u> as a result, but the <u>applications have similar structures</u>. They are <u>easier to maintain</u>, and they <u>seem more consistent to their users</u>. On the other hand, you lose some creative freedom, since many design decisions have been made for you."

"If applications are hard to design, and toolkits are harder, then frameworks are hardest of all. ... Any substantive change to the framework's design would reduce its benefits considerably, since the framework's main contribution to an application is the architecture it defines. Therefore it's <u>imperative to design the framework to be as flexible and extensible as possible</u>."

According to ATF Glossary

- **Framework:**
  A skeleton of an application into which developers plug in their code, using mechanisms defined by the framework. (See also toolset.) Frameworks have a tendency to impact significantly the architecture of the system. For example, Geant3 is a framework.
- **Toolset:**
  A collection of functionality, implemented as subroutines and functions, or classes. Toolsets tend to have a smaller influence on the architecture of a system than frameworks (see above). For example, HBOOK is a toolset.

What is a Component?

- **Lakos, <u>Large-Scale C++ Software Design</u>**

"<u>A component is not a class and vice versa. Conceptually, a component embodies a subset of the logical design that makes sense to exist as an independent, cohesive unit."</u>

"bundles a <u>manageable amount of cohesive functionality</u> that often spans several logical entities..."

"<u>lifted as a single unit from one system and reused effectively in another system</u> without having to rewrite any code."

- **Garone, <u>Managing Component-Based Development: SELECT Software Tools</u>**
  - *Is both discrete and well defined in terms of its functionality.*
  - *Provides standardized, clear, and usable interfaces to its methods*
  - *Can run in a container, with other components, or standalone (or any combination of these)*

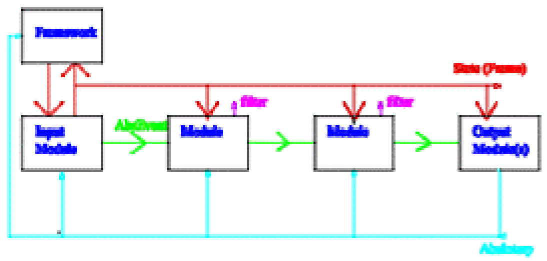
Physics Frameworks

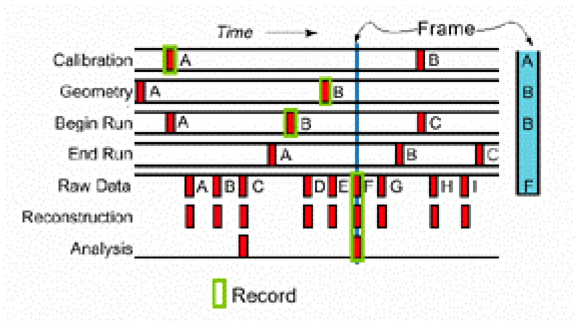
- **Framework-"like" programs have been used in HENP to address software complexity challenge.**
  - **—ac++, arve, carf, cleo fw, d0 fw, gaudi, jas, lulu, openscientist, orca, paw, root, staf, tas, ...**
- **Too often no distinction is made between software which address:**
  - **—control, data I/O, graphics, data analysis, etc.**
- **This arises from the desire/necessity that these "domains" be seamlessly integrated.**
- **However, the resultant lack of compartmentaliza-tion complicates maintenance, upgrades, etc.**

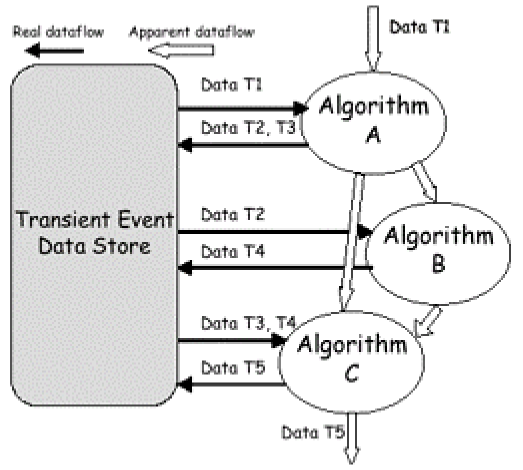
"the size of the component to be changed has a much larger impact on effort than the size of the change itself."- Niessink

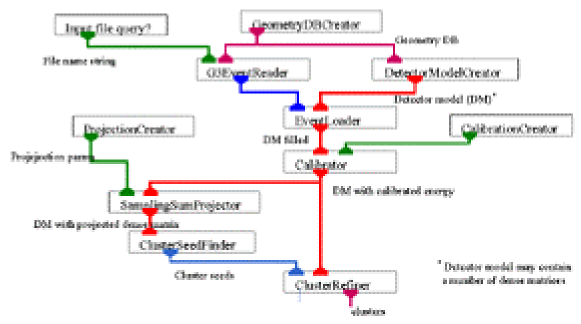
Framework Design Classifications

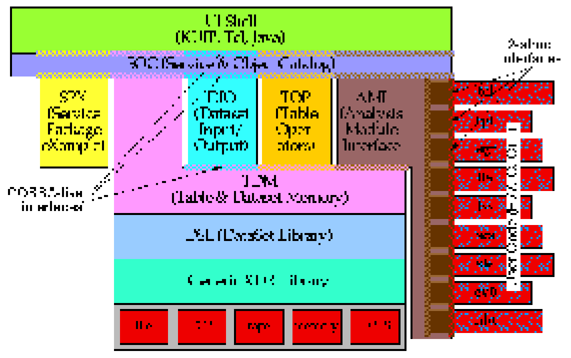
- **Finite State Machine - AC++**
- **Action on Demand - CARF**
- **Stream/Record/Frame - CLEO**
- **Simulated Data Flow - Gaudi**
- Mobile Agents - JAS
- **Object Network - ONCM**
- C++ Interpreter - ROOT
- **Software Bus - StAF**










UI Shell
(KUIP, Tcl, Java)
SOC (Service & Object Catalog)
SPX (Service Package eXample)
DIO (Dataset Input/Output)
TOP (Table Operators)
AMI (Analysis Module Interface)
TDM (Table & Dataset Memory)
DSL (DataSet Library)
Generic STAF library
CORBA-like interfaces

Finite State Machine - AC++

- Data and Algorithms indistinct
- All communication between components are "persistable objects"
  - —STL-like containers of basic types & storable pointers
- Using ROOT I/O
- Framework is ignorant of event model

Action on Demand - CARF

- User requests "Object of Interest"
  - —Pull, rather than Push concept
- Data and Algorithms distinct?
- Data Objects identified by Type + Producer
- Event based high-level steering

Stream/Record/Frame - CLEO

- Not event-centric model
- Records have time & duration
- Stream = set of time-ordered records
- Frame is sync of many streams
  - —snapshot of states of streams

Simulated Data Flow - Gaudi

- **Data and Algorithms distinct**
- **Physical design - strong focus**
- **Services defined in terms of abstract interfaces**
- **Global event - Hierarchical event model**
- **Converter objects for transient Û persistent**
- **Data flow not part of algorithm interface**

Object Network - ONCM

- Data and Algorithms distinct
- Dependencies between components determine execution order
- Based on Observer pattern concept
- Data flow part of algorithm interface
- Data objects - Any valid C++ class
- Distributed memory management

Software Bus - StAF

- Data and Algorithms distinct
- Physical design - strong focus
- Dynamic loading of components
- CORBA = Widely Accepted Standard
- Data Objects - RDB tables & containers
  - Hierarchical (UFS-like) event model
- Algorithm Objects - wrapped F77/C/C++
  - Algorithms independent of framework
  - Data flow part of algorithm interface
- Strong reliance on code generation

"Common" Design Prinicples

- Component design
  - —Black box/Black world
- Data distinct from Algorithms
  - —Data HAS a special role in physics analysis
- Physical design considerations
  - —Compile, link, & run time dependencies
- Data flow part of algorithm interface
  - —Interface says what component does
- Observer Model for Algorithms
  - —Automatic actions based on dependency

Other Design Prinicples

- Code generation tools
  - —Makes users' lives simpler
  - —Provides natural tool for migrations
- Dynamic loading
  - —Provides fast prototyping capability
- Code Þ Script Þ GUI
  - —Prog/Script Lang. - Batch
  - —Script Lang. - Complex Interactive
  - —GUI - Ease of use and automation

PC: Code Generation Tools

- Tools like SWIG already exist
- Need to generate: headers, adapters, skeletons,...
- Programming Lang.-neutral Description Lang?
  - —Multi-lang support, graceful retirement

	IDL	ODL/DDL	C++	Java	XMI/MOF
Suitability control	+ interface description	+ superset of IDL	Too complex?	arguments	+
Suitability event	- no persistence, no associations, ...	+ ATLAS is using it	as IDL	No associations	+
Tools: parsers	Sun CFE, ORBs, JavaCC	Objectivity internal	-cint	JavaCC	Argo, Rose?
Tools: API	Interface Repository (ORBS)	Objectivity internal	- ROOT Meta Library	Language	?
Stability	+	?	+	At least as dict language	?
Lifetime	industry standard, tools to evolve	dying?	+	+	?
Overall	simple, suported	Much related to Objectivity	Hardest for generator	Easiest for generator	Shot in the dark

JM: JavaCC IDL 2 SWIG

- Use IDL as the single method for describing interfaces between components
- Parser and AST generation using *jjtree* and *javacc*
- SWIG emitter scans tree 3 times to generate typemaps, classes, & additional methods
- SWIG supports several different interface languages (eg. Tcl, Perl, Python, etc.)
- IDL vs. C++
  - —Productions: IDL=75/C++=111
  - —JavaCC Specifications: IDL=10k, C++=39k
  - —JavaCC LOOKAHEADs: IDL=3, C++=120

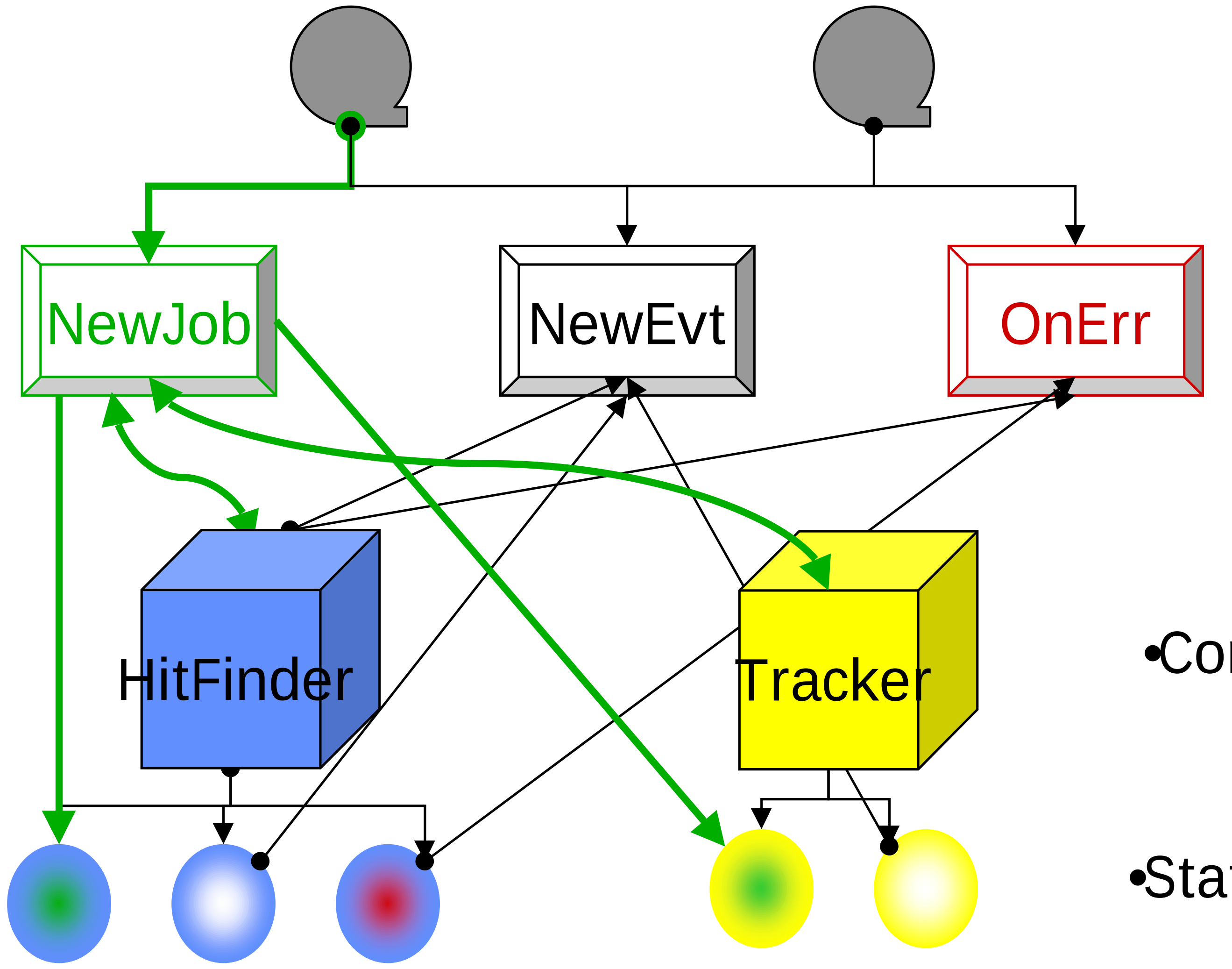
PC: Control State Object Net

- Is data-flow the way we think when we analyze the data?
  - —No! We pull data at random (well…) from the modules that reconstructed them, after they are done for that event (run, job,…)

- How easy will be to predict (and repeat!) the execution path of a 1000 objects network?

- I don’t think we can reasonably interact with a self-triggering network of say 1000 components without knowing its global state.

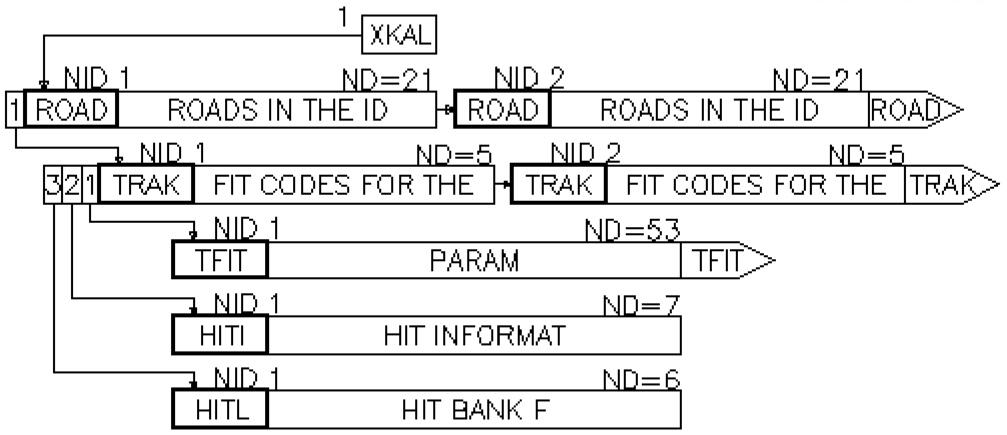
PC: The Control States Network

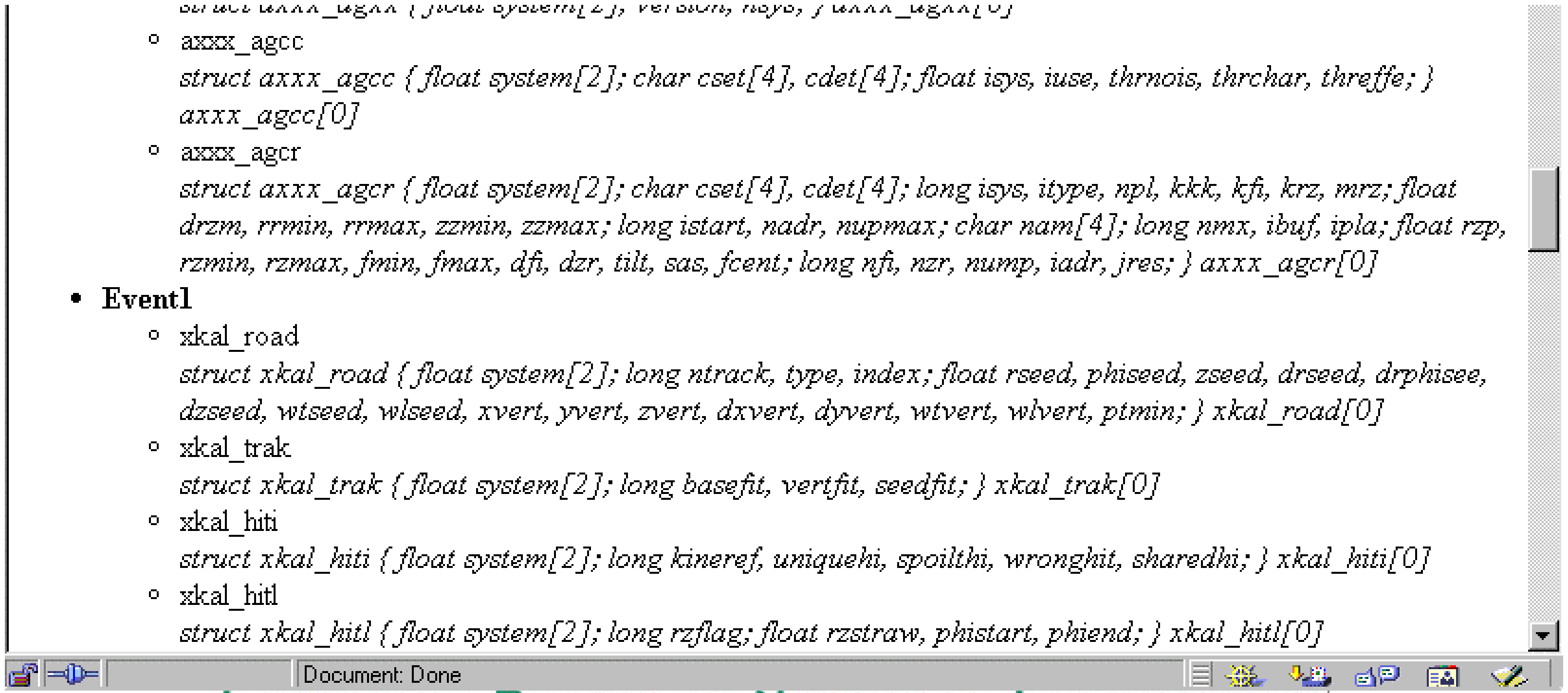


- Sources
- States
- Components
- State Methods

ATLAS TDR Software in StAF

- **XKALMAN Reconstruction**
- **ZEBRA banks to StAF Tables & Datasets**





- axxx_agcc

  struct axxx_agcc { float system[2]; char cset[4], cdet[4]; float isys, iuse, thrnois, thrchar, threffe; } axxx_agcc[0]
- axxx_agcr

  struct axxx_agcr { float system[2]; char cset[4], cdet[4]; long isys, itype, npl, kkk, kfi, krz, mrz; float drzm, rrmin, rrmax, zzmin, zzmax; long istart, nadr, nupmax; char nam[4]; long nmx, ibuf, ipla; float rzp, rzmin, rzmax, fmin, fmax, dfi, dzr, tilt, sas, fcent; long nfi, nzr, nump, iadr, jres; } axxx_agcr[0]

- **Event1**
  - xkal_road

    struct xkal_road { float system[2]; long ntrack, type, index; float rseed, phiseed, zseed, drseed, drphisee, dzseed, wtseed, wlseed, xvert, yvert, zvert, dxvert, dyvert, wtvert, wlvert, ptmin; } xkal_road[0]
  - xkal_trak

    struct xkal_trak { float system[2]; long basefit, vertfit, seedfit; } xkal_trak[0]
  - xkal_hiti

    struct xkal_hiti { float system[2]; long kineref, uniquehi, spoilthi, wronghit, sharedhi; } xkal_hiti[0]
  - xkal_hitl

    struct xkal_hitl { float system[2]; long rzflag; float rzstraw, phistart, phiend; } xkal_hitl[0]

Document: Done

Year 1 Design & Proto.

- Requirements Doc.s - ATF
- Use Case Scenarios/Behaviors - Amako, Tull, etc.
- "Market Surveys"
  - —Framework/Control Architectures ATF, LBL
    - Object Network, Software Bus, Simulated Data Flow, Finite State Machine, etc.
  - —Software/Hardware Technologies
    - eg. CORBA 3, IA64, Java Beans, XMI/MOF, ...
    - "Think" 5-20 years out
    - Implement 1-3 years out
  - —Other Software Projects
    - eg. ACE, Clipper, Nova, PP Data Grid, SWIG, ...
  - —Leverage FTEs & Expertise

CD: Common HEP Behavior Patterns

- Reconstruction
  - —Every event touched, all data in an event touched, considerable data generated.
- Data mining
  - —Bulk data processing to extract all events meeting particular, restrictive criteria. Every event in input collection touched, but not all data in an event touched. Little added data. Usually done by iterated refinement.
- Data prospecting
  - —Highly interactive processing of test samples. Intensely volatile no two physicists doing the same thing, no physicist doing the same thing twice.

Time Lines and Pressures

- **Control Framework development <u>MUST</u> be heavily front loaded.**
  - **—To begin analyzing physics data in 2005...**
  - **—...we need an MDC in 2003/2004, which implies...**
  - **—...physicists have developed, integrated, and debugged their analysis code, which requires...**
  - **—...the framework in which their code runs has been designed, developed, debugged, and documented.**
- **However, analysis codes (ie. Physics TDR) exists today that people want to use today.**

Usable Prototype in FY2000

- Physics TDR Software runnable & usable
  - —Comparison w/ TDR
  - —Not all code will be C++-ified at same time
- SW Tools for integration (some, proto.s)
  - —Ease migration for users
- Functional user interface (non-dist'd)
  - —Scripting lang. only - no GUI
- Natural interface to Anal. Tool(s)
  - —Plural seems right - PAW, ROOT, HTL, JAS
- Working interface to ATLAS DB(s)
  - —and ZEBRA? (converted?)
- Usable for OO SW Devel.

Year 1 Technical Decisions

- There are MANY, including:
  - —Primary Design Classification
    - Object Net, Software Bus, etc. - Hybrid/Combination
  - —Physics TDR Software
    - F77 Wrapping, COMMON block elimination, ZEBRA
  - —Core Programming Language
    - Certainly C++ now; Role for Java?
  - —Dictonary Language
    - eg. IDL, C++, ODL/DDL, Java, XMI/MOF, SWIG, ...
  - —Communication Protocol
    - eg. CORBA, Java RMI/Java Beans, DCOM, RYO
  - —Storage Technology Neutral DB Interface
    - Transient/Persistent Mapping