The *BaBar* Online Computing System

Gregory P. Dubois-Felsmann
Caltech

for the *BaBar* Computing Group

Computing in High Energy Physics
Padova, Italy, 7-11 February 2000

Contributors

G. Abrams (7), J. Bartelt (17), R. Bartoldus (4), R. Berger (17), D.N. Brown (7), E. Chen (1), R. Claus (17), R. Cowan (10), S. Dasu (19), C. Day (7), F. Di Lodovico (3), G.P. Dubois-Felsmann (1), B. Franek (16), N. Geddes (16), T. Glanzman (17), G. Grosdidier (6), P. Grosso (18, now at 17), R. Hamilton (4), A. Hasan (17), M. Huffer (17), R. Kapur (7), A. Khan (8), Yu. Kolomensky (1), A.J. Lankford (2), S. Lewis (7), C. Lionberger (7), S. Luitz (17), S. Metzler (1), M. Morandin (12), C. O'Grady (17), J. Olsen (9), T. Pavel (17), A. Romosan (7), E. I. Rosenberg (5), J.J. Russell (17), F. Safai-Tehrani (14), A. Samuel (1), O. Saxton (17), I. Scott (15), N.B. Sinev (11), E. Varnes (13), M. Weaver (1), S. Yang (1), S.M. Xella (16), G. Zioulas (2, now at 17)

(for the *BaBar* Computing Group)

(1) California Institute of Technology
(2) University of California, Irvine
(3) University of Edinburgh
(4) University of Iowa
(5) Iowa State University
(6) LAL, Orsay
(7) Lawrence Berkeley National Laboratory
(8) University of Manchester
(9) University of Maryland
(10) Massachusetts Institute of Technology
(11) University of Oregon
(12) INFN Sezione di Padova
(13) Princeton University
(14) Univ. di Roma 'La Sapienza', INFN Sezione di Roma
(15) Royal Holloway and Bedford New College
(16) Rutherford Appleton Laboratory
(17) Stanford Linear Accelerator Center
(18) INFN Sezione di Torino
(19) University of Wisconsin

(Core software only; detector subsystems provided subsystem-specific online software, and many valuable contributions to the core software emerged from these efforts. Our warm thanks to theoperations crew as well, for much valuable input and much patience.)

Outline

Goal is to sketch system, share OO experiences

- Not a detailed technical presentation of the system design (but a long paper is forthcoming)

- Overview and basic concepts
  - Other *BaBar* online-related CHEP presentations
- Non-event data
- Event data
- System integration
- Applications
- Lessons Learned: A personal meditation

Overview and basic concepts

- *BaBar*
- Fundamental online system requirements
- Online system status to date
- Overall structure & schematic
- External tools used
- Other *BaBar* online-related CHEP presentations

BaBar

- Designed to measure CP-violating observables and related quantities in *B* meson decays
  - Uses asymmetric e^+ e^- collisions producing the (4S), at the PEP-II facility at SLAC
  - Detector design:
    - Standard 4 detector geometry skewed to match asymmetry
    - Components: high channel count silicon detector, drift chamber, DIRC (Cerenkov p.i.d.), CsI crystal calorimeter, instrumented iron
  - High *B* meson statistics needed:
    - Very high luminosities, hence beam currents, hence backgrounds, hence rates and data volume
    - High-availability, low-deadtime "factory" operation

Fundamental Online System Requirements

- Event data path:
  - Acquire digitized calibration-corrected data from *BaBar* systems - O(300,000) channels:
    - 2000 Hz hardware trigger (Level 1) rate
    - O(13 μs) trigger latency, buffering depth
    - O(33 kB/event) after zero suppression
  - Perform software triggering (Level 3) to reach 100 Hz rate to mass storage and full reconstruction
  - Do this with minimal (<1%) deadtime
  - Provide for fast-feedback (minutes) data quality monitoring
  - Perform full reconstruction with O(8 hour) latency

Fundamental Requirements, cont.

- Monitoring and control:
  - Monitor all slow control data (voltages, gas flows, etc.)
    - Provide operator alarms
    - Interlock data-taking with desired conditions
  - Provide high-efficiency interface with accelerator operations
    - Switching between “injection permitted” and running states
  - Support 24x7 operation by non-experts
  - Allow detector components to be operated in parallel
    - “Partitioning” - for calibration and diagnostics

Overall status and experience

- Online system coded almost entirely in C++
  - Very few contributors with any significant previous OO experience
    - *BaBar* provided consultant-taught training in OO design
  - Almost entirely ex nihilo, except for external packages used (*q.v.*)
  - Starting in 1996-1997, depending on which component
- Prototypes working in October 1998
  - Cosmic ray commissioning runs
- Full scale physics data acquisition on schedule in April 1999
  - All online system components available
  - 1000-1200 Hz hardware trigger accept rate, limited by interim network-based event builder, was just compatible with beam conditions
  - >90% running efficiency achieved within three months
- Many upgrades since then
  - Principal focus on improving interfaces, automation, reliability; now...
  - Deploying VME event builder to reach 2000 Hz+ w/o deadtime; needed!

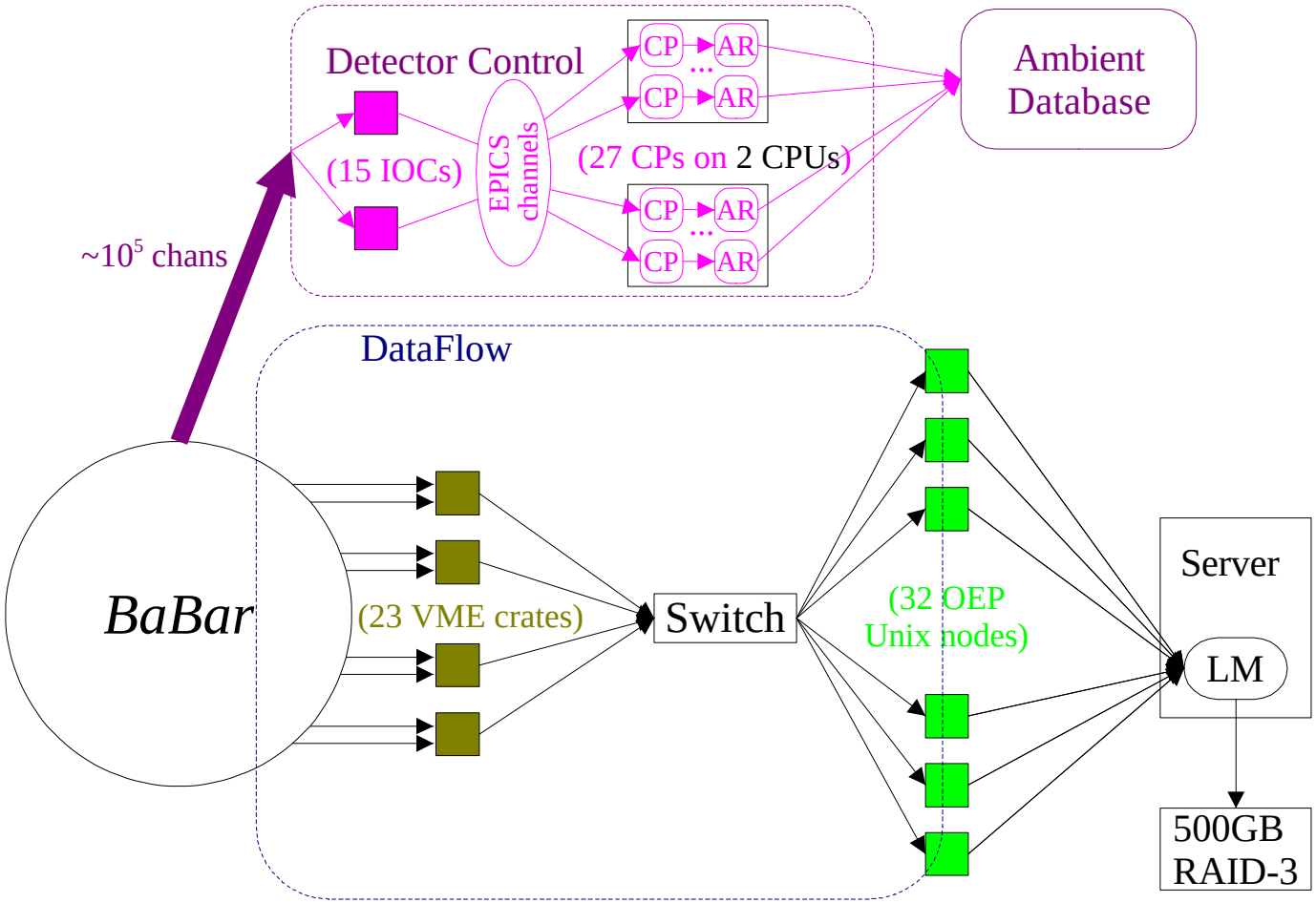
Overall Structure

- Event path components:
  - DataFlow: control of front-end electronics, data acquisition, “feature extraction”, and event building, feeding...
  - Online Event Processing (OEP): near-real-time processing of complete, built events, including software triggering and data quality monitoring, writing accepted events to...
  - Logging Manager: collects data from OEP, writes mass storage, redistributes data to...
  - Prompt Reconstruction: single-pass full physics-quality reconstruction and output to OO database event store

Overall Structure II

- Non-event data components
  - Detector Control: control and monitoring of detector systems and environment
  - Databases for configuration, calibration, and enviromental data
  - Run Control: ties major functional components together, provides primary operator interface

Functional Schematic



External Tools

Software packages from outside *BaBar*
(with significant use in one or more online system components)

- VxWorks
- EPICS
- Objectivity/DB
- CMLOG (Thomas Jefferson Lab. error logging system)
- TAO (D. Schmidt's freeware CORBA implementation)
  - and ACE (C++ wrappers for Unix IPC and network interfaces)
- DIM (Delphi fault-tolerant publish-subscribe messaging, C. Gaspar)
- Rogue Wave Tools.h++
- CLHEP (extensively *BaBar*-modified)
- JAS (Java Analysis Studio, T. Johnson)

Other *BaBar* online-related presentations

A180 - The BaBar Prompt Reconstruction Manager
A288 - The BaBar Prompt Reconstruction System
B112 - The BaBar Online Databases
B138 - Event Logging and Distribution for the BaBar Online System
B157 - DIM, a Portable and Efficient Package for ... Interprocess Communication
B189 - A Data Acquisition Monitoring System for BaBar
B358 - Automated Data Quality Monitoring in the BaBar online and offline systems
B360 - BaBar Online Detector Control
B375 - The BaBar Event Builder
C103 - Operational Experience with the BaBar Database
C106 - An Overview of the BaBar Conditions Database
C110 - Improving Performance of Object Oriented Databases, BaBar...
D161 - Managing ... commands/processes within a distributed environment through C++ and CORBA in BaBar Prompt Reconstruction
D192 - Traffic analysis and performance ... in the [BaBar DAQ] networks
D290 - Production Experience with CORBA in the BaBar experiment
E2.115 - Sun['s] AutoClient and management of computer farms at BaBar
F1.250 - Java Analysis Studio

Non-event data components

- Detector Control
- Ambient and Conditions Databases
- Configuration Database

Detector Control

- Controls and monitors...
  - detector operating parameters (power supplies, gas systems)
  - environmental conditions (temperature, humidity)
- Ensures safe and reliable operations; interlocks...
  - ... data-taking with maintenance of configured operating point,
  - ... PEP-II injection with safe detector conditions
    - Informs accelerator operator of *BaBar* backgrounds and deadtime
- Implementation based on EPICS
  - 15 Motorola MVME177 IOCs handling O(100,000) channels
    - Update rates vary from a few per hour to a few per second
  - User interfaces built from *BaBar* adaptations of EPICS Display Manager (DM), Alarm Handler (ALH)
  - *BaBar*-specific C++ interface layer ("component proxy")
    - 27 "logical components" visible to Run Control, receive configuration commands and return summary "runnable" flag;
    - write data to Ambient Database Archivers...

Ambient/Conditions Database

- Databases of *History*
  - Both are time-indexed Objectivity databases
    - have multiple *containers*: time interval series for a single type of data
  - Ambient: history of Detector Control information
    - C.P.'s feed "Archiver" processes through shared memory buffer cache
      - Data are flushed once/hour to DB; aggregate write rate: 2 MB/hour
    - Access tools:
      - C++-based CORBA servers for cache and DB
      - Java client app for visualization (migrating now to JAS)
      - N-tuple extractor for detailed offline analyses
  - Conditions: revisable history of detector behavior
    - Key online application: calibrations
    - Available for use in reconstruction and analysis
    - Data for any time interval may be superseded, with retrievable history

Configuration Database

- Database of *Intention*
  - *BaBar* state machine's Configure transition is available to all online system components; need to associate with config. data
    - Transition carries a "configuration key"
    - Configuration database allows any component to use the key and its identity to retrieve appropriate object(s)
    - Strongly-typed interface based on Objectivity; templated C++ API
    - Hierarchical organization - "config. tree" - reflects detector systems
    - Motif GUI allows graphical build-up and editing of configurations
    - Aliases are maintained for the production configurations for various "run types" (Physics, Cosmics, ...)

Event data

- DataFlow
- Online Event Processing (OEP)
- Logging

- Prompt Reconstruction (covered in detail elsewhere)

DataFlow

- Hardware platform:
  - 24 9U VME crates containing ~160 Readout Modules (ROMs)
    - Custom combination of MVME2300 with i960 I/O processor and *BaBar*-specific front-end electronics interfaces
    - Comm'n with front ends is via "GLINK" gigabit optical fiber (2-3 / ROM)
  - Custom partition control and trigger distribution "Fast Control" system
    - Up to 12 arbitrarily selected sets of crates can be operated independently
  - 100Base-T fully switched network connecting all ROMs, gigabit links to fast servers
- Tasks:
  - Configuration and readout of front-end electronics
  - Data transport, buffering, and hierarchical event building from ROMs through to OEP (*q.v.*) Unix nodes, with minimal deadtime at 2000 Hz
  - "Feature extraction" (zero-supression, calibration correction...) of data as required by detector subsystems
  - Enforcement and monitoring of back pressure (deadtime)
  - Masking and prescaling of Level 1 (hardware) triggers

DataFlow II

- Implementation:
  - Under VxWorks RTOS on ROMs, Solaris on Unix
  - System almost entirely coded in C++
    - Object-oriented design throughout
    - Most *user* APIs based on abstractions with pure interfaces, while critical internals typically coded statically or inlined
    - Assessment: 1) judicious use of language features did not compromise net performance (sufficiently rich user interface would typically require comparable trade-offs); but 2) core and user development, and especially subsequent modification, was greatly eased by OO methodology.
  - Performance achieved:
    - Deadtime threshold at ~1200 Hz with interim purely network-based event build; ~2500 Hz on test bench with final VME e.b. now being deployed (both with typical 35 kB events).
  - Notable new feature: fine-grained performance, deadtime monitoring:
    - Able to account for all forms of deadtime in the system (2.6 μs/event plus any back pressure from slow consumers), classified by source and trigger type and on precise, frequent time boundaries.

DataFlow III

- Detector subsystem interface:
  - Subsystems contribute front-end configuration, “feature extraction”, and calibration code by implementing abstract finite-state-machine action interfaces defined and invoked by DataFlow.
  - Code is loaded using VxWorks shareable library support; cold boot of entire system from E-450 server takes 5-10 seconds.
  - Loading of calibration constants and run parameters from conditions and confign DBs involves Unix-side middleware that reads from Objectivity and builds “tagged containers” (*q.v.*) for download to ROMs.
- Data transport and access (in ROM code and beyond):
  - Basic DataFlow transport is for vectors of uint32_t labeled with geographical and logical addresses as event hierarchy is built;
  - A type-safe layer (“pauper’s OO database”) has been built that insulates users from this anonmyous data:
    - *Tagged container* (TC) base class adds persistent type labels to data
    - Templated “smart pointer”-like accessors verify labels
    - Type ⇔ label correspondence is created at link time, under release control; tools available to monitor stability of labels

Online Event Processing (OEP)

- Processes fully built events on Unix farm (32 Sun Ultra-5s):
  - Executes of the Level 3 trigger algorithms, allowing appends to events;
  - Performs rapid-feedback data quality assurance (“Fast Monitoring”);
  - Supplies data to online event displays;
  - Logs selected events, with optional prescaling by Level 3 type; and
  - Supports online calibrations.
- Implementation:
  - OO extension of *BaBar* offline Framework and ProxyDict data model
    - Modules easily moved between OEP, offline applications (notably, trigger and monitoring code are typically developed first in pure offline apps)
  - Essence of OEP is thus a DataFlow/TC - to - offline adapter
    - Access to TC data is via templated abstract iterators, decoupling users from details of adaptation and event hierarchy
    - Both guaranteed event receipt, w/ back pressure, and sampling supported
    - DataFlow “push” adapted to offline “pull” data source model
    - Configuration via TCL code, as for offline, but found w/ key in config. DB
  - Toolkit (Distributed Histogrammer) provided for summing monitoring data and time histories across nodes, w/ CORBA API, Java clients

Logging

- Logging Manager writes events accepted by Level 3, collected from all OEP nodes, to “intermediate store”
  - ~500 GB RAID-3 disk buffer on E-450 server
    - Provides decoupling between logging and reconstruction rates
    - Allows running during outages in central SLAC HPSS archiving system
  - Logged data is in original tagged container format
  - Requirement: 35 kB/ev, 100 ev/s; achieved: >500 ev/s
  - Control and monitoring via CORBA interface
  - Bulk data channel is vanilla TCP (required for performance)
- Files are copied to HPSS archive
  - Tapes are written directly to improve throughput
- Second “reverse” Logging Manager farms out events to Prompt Reconstruction on 100-200 nodes
  - Output written to Objectivity event store
  - Covered in several other CHEP 2000 presentations
- Archived raw data in HPSS available for limited diagnostic use

System Integration

- Run Control
- Error logging
- Release management

Run Control

- Principal operator interface
  - Establishment and control of detector partitions
  - Selection and distribution of configuration keys
  - Control of system state (start, stop runs, ...)
- Coordinates online components
  - Provides synchronization and automation as appropriate
- Controls recording to Web/Oracle electronic logbook
- Based on FSM model of all components
  - Implemented in RAL SMI++ object-oriented FSM modeling system;
    - layered above Delphi DIM fault-tolerant publish-subscribe messaging
  - External components represented by “proxy” processes
  - Internal “abstract components” may be constructed from these
  - Interpreted language (SML) allows programming of behavior
- Configurable Motif GUI

Error Handling

- Errors affecting immediate operation are reflected to Run Control as error states in proxies, typically causing run pause
- All errors reported to CMLOG (TJ Lab) archive server
  - CMLOG clients allow viewing live messages and browsing archive, configurable to show specific types of errors
  - Most messages reported through implementation of CMLOG logging client as C++ ostream
    - Code wishing to report errors uses only a generic

      ```
      ostream& ErrMsg( int severity )
      ```

      global function interface. Choice of implementation of ErrMsg, via Singleton pattern, is made on a per-application basis as part of image initialization (choices at present are cout/cerr or CMLOG or both)
  - DataFlow VxWorks clients report errors through DF-specific channel to controlling Unix process, which then reports to CMLOG

Release Management

- Elaboration of *BaBar* offline release system
  - Single CVS repository shared with offline
  - Most online software built using standard *BaBar* SoftRelTools
    - Common releases with offline; occasional dedicated releases (1-2 per month) to fine-tune production online system
    - Objectivity database schema established by these releases
      - Critical that running code match version of schema actually installed in production database
      - Couples to releases used for Prompt Reconstruction, analysis
  - DataFlow software has special build system
    - Handles VxWorks and Unix builds symmetrically
    - "Online release" layers this over core SRT release containing code common to both worlds
  - Production system allows for deployment of patches between releases
    - Full snapshot of code versions taken at start of every run
- Dependency issues require constant vigilance
  - Frequently dominate latency for deployment of new or corrected code
  - Heavy use of shareable libraries on Unix would help greatly, but...
    - Working with shareables in C++ has proved difficult (late start)

Applications

- Level 3 Trigger
- Fast Monitoring
- Calibration

Level 3 Trigger

- Constructed from standard *BaBar* Framework modules
  - Works both online, in OEP, and offline on playback data and in simulation production
    - Permitted original development of Level 3 algorithms and the online system itself to take place entirely in parallel
    - Greatly facilitates ongoing refinements
- Must operate in O(10 ms) per Level 1 Accept
    - Only online-specific concession to this is that the algorithms use raw tagged container data directly, without performing the conversion to the standard offline "digi" format
      - Digi-to-TC conversion available for use in simulations
    - Highly optimized drift chamber tracking and calorimeter clustering algorithms compute inputs to selection algorithms
    - Main idea is an OR of pure tracking and pure calometric selection, coupled with a *Bhabha* veto algorithm, but...
      - Details are beyond the scope of this presentation
- Fully configurable using configuration key, database
    - Which computations to do, what selections to perform, what cuts to use, which event categories to prescale and by how much, etc.

Fast Monitoring

- First level of data quality assurance
  - Provides rapid feedback to operators to assess basic status of detector subsystems
- Performed in OEP
  - Presently trigger monitoring is done in parallel on all 32 OEP nodes
  - Other detector subsystem monitoring is done on a single machine attached to a sampling event stream
    - Will be expanded to 32 nodes in the near future for increased statistics, enabling adding more sophisticated tests
  - Results are collected using the Distributed Histogramming toolkit, for
    - Display using Java Analysis Studio (with *BaBar* online customizations), in HTML/XML pages provided by subsystems and regularly reviewed by the Data Quality Monitor shift takers, and
    - Automated comparisons to reference histograms and distributions, with alarms produced in CMLOG when comparisons are out of tolerance (presently being commissioned)

Calibration

- May be performed at any stage along the event data path:
  - Against generated references, in DataFlow ROMs or in OEP
  - Using known processes in real data in OEP or in Prompt Reconstruction
- A single software framework is applicable at all stages:
  - Provides basic abstractions of
    - a channel of readout,
    - the characterization of its response,
    - the collection of data to determine the response, and
    - the validation and storage of this determination.
  - Provides additional tools for structuring and controlling calibrations performed in DataFlow
  - General goal is to move reference calibrations as far upstream as possible
    - Such calibrations may be performed in parallel for the various detector subsystems, using partitioning, with
    - Data accumulation and even calibration constant fitting done entirely within DataFlow, only transporting results to OEP for storage; thus
    - Full system calibrations can now routinely be done in 1-2 minutes, initiated by shift crew and fully automated through Run Control

Lessons Learned

(A personal meditation)

- We wrote the entire online system nearly from scratch in about 2.5 years, using OO techniques almost exclusively (something like 90% C++ and 5% Java).
  - Significant external inputs were used, of course.
- Almost all contributors were inexperienced in C++ and OO methodologies
  - Commercial training helpful, especially in creating a common vocabulary
- We found it possible to use these techniques to write highly performant code that has achieved leading-edge throughput for a HEP online system.
- We found that the commonly discussed advantages of object-oriented techniques were indeed present. The application of abstractions and of the hiding of implementations, in particular, has already on many occasions allowed us to make changes and migrations that would have been difficult using traditional methods. In general, the more carefully we followed the OO paradigm the better the results we achieved in this respect.
- It has been possible to use abstractions without unacceptable performance penalties even in the Level 1 trigger event loop in data acquisition.

- In order to attain the full benefits of these techniques, certain aspects of designs must be thought through and specified carefully at a very early stage, particularly the design of high-level interfaces and low-level representational objects (e.g., time stamps) that link across subsystems.
  - This requires a lot of foresight, creativity, and suppression of short-termism.
  - Where we did not do this well enough we have paid a measureable price in every instance.
- Allocating time to develop a full-scale ***integrated*** prototype online system would have been of enormous value.

- We have found that code organization and dependency management is a major challenge in the practical use of C++ in a project of this scale.
  - Much more so than one might conclude from well-known texts and the commercial training we received.
  - We started with a code organization oriented around major functional blocks and individuals' responsibilities.
  - This proved to be highly naïve and has required a continuous stream of code reorganization to manage dependencies. We (I) believe we are close to or beyond the manageability scaling limit of our structures now.
  - Future projects would benefit from substantive advance planning here, and especially from more systematic separation of interface from implementation at the software package level and the use of the idea of “abstract packages”.
  - We have found producing a robust shareable library implementation of our code that actually allows plugging in backwardly compatible upgrades to be very difficult in practice. Making this really practical in a C++ - based system would require planning for this from the very outset in coding rules and release management tools.

- We have obtained only limited benefits from using object-oriented databases.
  - *NB: this statement is restricted to the non-event-store databases used in the online system.*
  - We have not used the unique features of OODBs in a significant way, and we (I) believe that a traditional relational database model would have served our needs well with less development, and especially with less tuning effort.
  - Some benefits were seen:
    - The availability of inheritance relationships has been of some use to us.
    - The ability to use templates in and with database schema classes has permitted some widely used database code to be written very cleanly.
- However,
  - The need for a single common schema for the aggregate of all persistent data, a characteristic of Objectivity, has significantly complicated our dependency management problems, and frequently leads to synchronization delays in the deployment of new code or data types.
  - Concurrency issues in the Objectivity implementation (AMS and lockserver) constantly require fine-tuning to get even modest performance.