Antonio Ceseracciu

(for the BaBar computing group)

SLAC

The evolution of the distributed Event Reconstruction Control System in BaBar

CHEP 2004, Interlaken

BaBar Data Reconstruction

- Prompt Reconstruction (PR)
  - performs Calibration and Event Reconstruction from raw data coming from the detector, using ~ 400 CPUs
  - consists of a set of tools i.e. executable programs
  - executable programs contain “physics” code
- PR Control System
  - presented in CHEP’03: *The new BaBar Data Reconstruction Control System*
  - manages production of Calibration and Event data
  - runs the PR tools as black boxes
  - provides a high level user interface to the system

Control System Evolution

- This talk will focus on the Evolution of the Control System.
  - One original design requirement of the BaBar Control System concerned its evolution: *flexibility*.
  - This talk will discuss how designing for flexibility affected the actual evolution of the system.
  - The most effective solutions will be highlighted.

Defining flexibility

- *Flexibility*: the quality of being adaptable, or, evolve
  - a metric for flexibility can be derived by looking at the overhead in run-time performance and development time introduced by increase in complexity
  - a basic metric for sheer complexity in software is the number of lines of code (LOC)
  - e.g.: in an *ideal* flexible system, the effort required for adding a new feature or fixing a bug is constant regardless of the complexity
  - e.g.: common experience tells that it is more difficult to debug a 1,000 lines program than a 10 lines one!

The requirement of flexibility

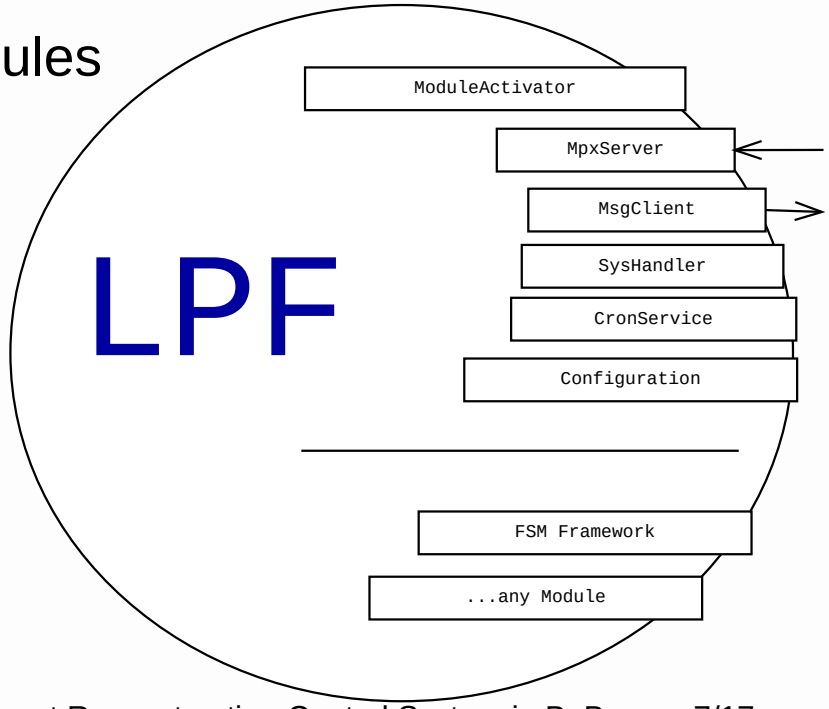
- Flexibility was a main requirement for the Control System:
  - **complexity**: it is a distributed system
  - **evolution in time**: changes in the processing model, and in the environment
  - **evolution in space**: the same system must run in different infrastructures
  - **human interface**: ordinary maintenance must be handled at configuration level, not require interaction with developers

Designing for flexibility

- Design features concerning flexibility:
  - **modularity**: code components rather than applications
  - **protocols**: explicit modeling of interaction between components
  - **configuration system**: assemble components into entities and complete systems
  - **abstractions**: identify and express fundamental patterns
  - **anticipation**: future development paths are part of the initial requirements.
- The next slides discuss the implementation choices for each of these features.

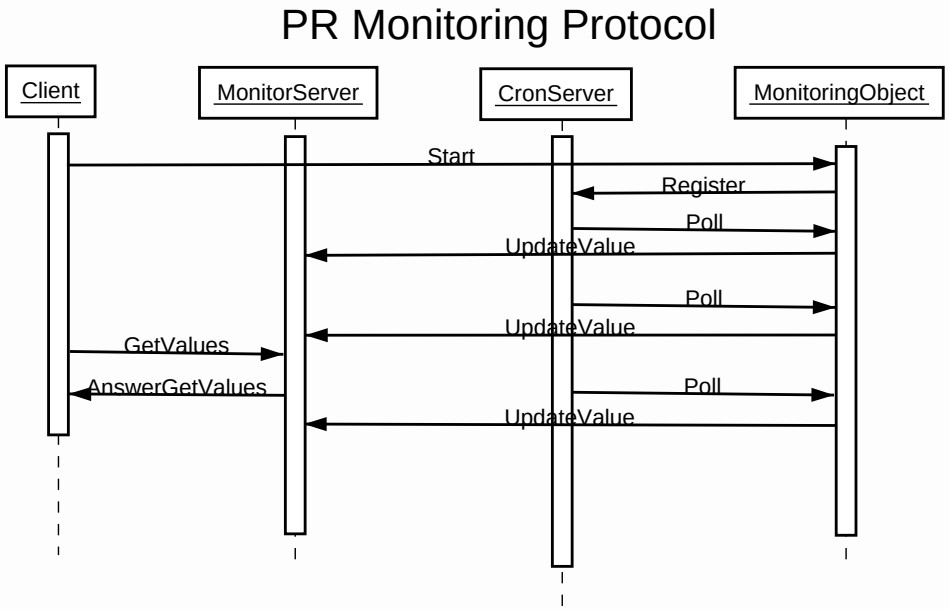
Modularity

- Light Processing Framework (LPF)
  - Everything in the Control System runs inside an LPF
  - In other words, everything is coded as LPF modules
  - Light: delegates any task to core modules
- LPF provides:
  - Message passing engine
  - Cooperative multitasking
  - Static module loading



Protocols

- In a modular system, interaction between modules can be more complex than the internal logic of the modules themselves
- Explicit definition of protocols and interfaces is a good way to capture this complexity
- e.g. the diagram below shows the protocol for a subsystem.



Configuration System

- In a dynamic modular system, the configuration engine is responsible for assembling the system together:
  - assemble components into entities, or agents
  - assemble agents into a distributed system
- The configuration information includes all the needed information for activating (booting) the system
- The system is described as a hierarchical tree of services
- XML language proved fit to this task, being inherently hierarchical
- This all adds to the common “good programming” practices: nothing is hardcoded; orthogonal configuration spaces for different modules

Abstractions

- Identifying patterns and providing abstract models for them is fundamental in software design.
  - Good abstractions allow us to express evolutionary pressure into extensions, or components, and new policies to combine them, *not* into modifications to the existing code and infrastructure.
- The main high level abstraction in the CS is the Finite State Machine (FSM) processing model
  - processing is split in sequences of steps, or *states*.
  - the FSM defi nition, coded in a custom language, connects each state's jump labels to another state, thus defi ning the processing model.
  - the FSM is also the main interaction point for the user interface.
  - the FSM description is the policy, the states' implementation the mechanism.

Evolution: maintenance

- Ordinary maintenance
  - Policy changes, e.g. adding or removing a check: typically implemented by changing the FSM description.
  - Adding new user commands: complete separation of user interface code from the application code, interfacing via regular message passing.
  - Supporting changes in the infrastructure: adding dedicated modules.
- Major maintenance
  - Paradigm shift in the processing model: forking the FSM description to remove or add states, and coding new states.
  - Deploying the system in a different infrastructure: adapting the configuration.

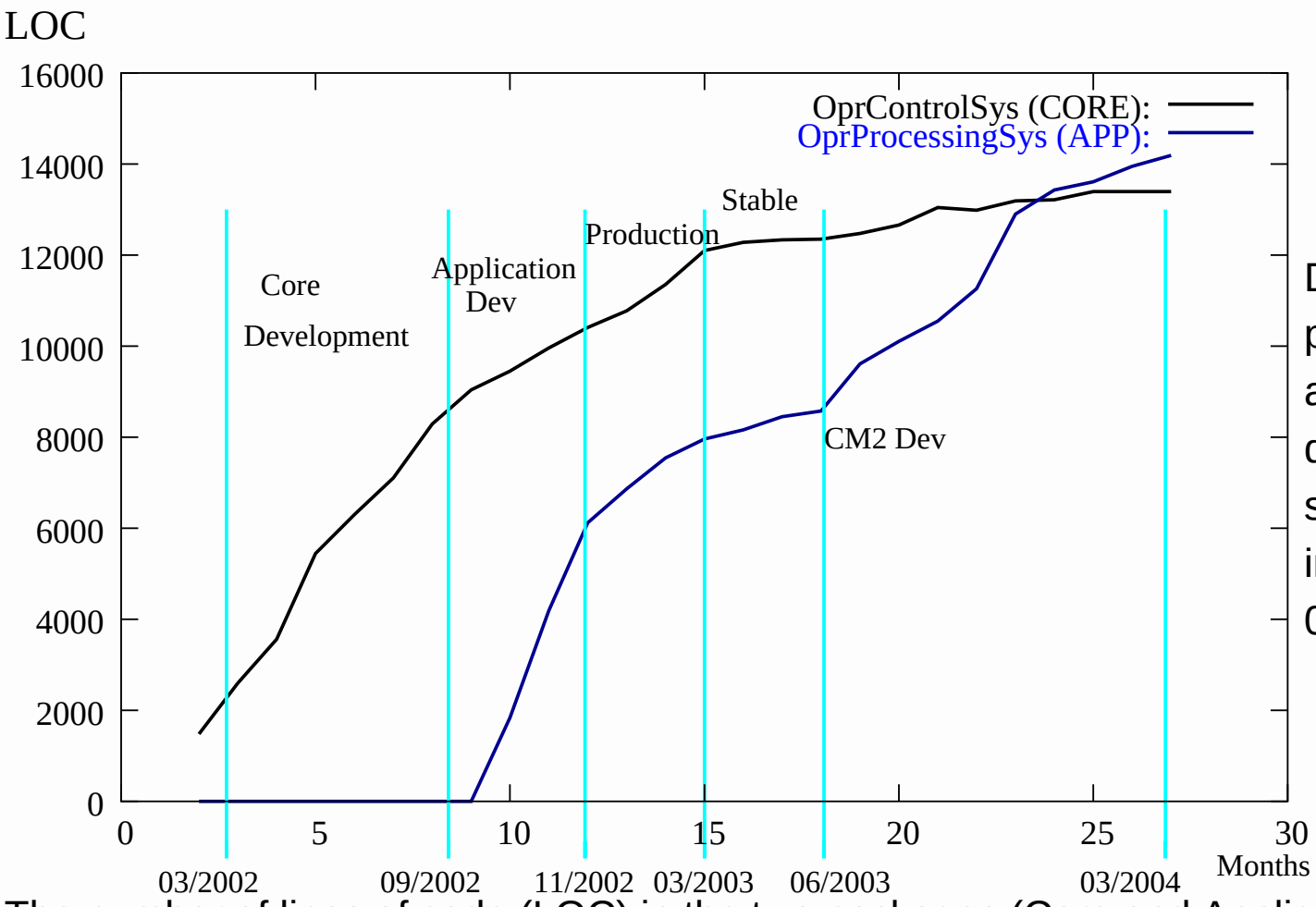
Evolution: CM2 requirements

- The Computing Model 2 upgrade:
  - During the year 2003, BaBar deployed a new computing model, called CM2.
  - See presentation “The new BaBar Analysis Computing Model”
- New requirements for the data production system:
  - While efficient for physics analysis purposes and in reducing the latency to make the data available, CM2 imposed additional requirements to the control system
  - Responsibility for data files management passed from the DBMS to the Control System, increasing its complexity
  - An additional time consuming post processing step is required to assemble the final data collections

Evolution: upgrade

- The Control System under evolutionary pressure
  - Forking the FSMs definition to create a new processing model and accomodate the new states needed
  - Addition of a PostProcessing stage
    - Independent system, described by a dedicated FSM
    - Filesystem interface: it is coupled to the main processing system by a simple filesystem interface.
    - Scaling: any number of PostProcessing FSMs can be statically coupled to a processing farm
- The main upgrade was implemented quickly, and allowed basic production on time with the availability of the event store tools.
- Later development effort was concerned mostly with improved checking and recovery of error conditions.

Evolution: codebase



During the project, the allocated FTE decreased steadily from initial 3.0 to a final 0.5.

The number of lines of code (LOC) in the two packages (Core and Application) of the code repository, as a basic metric for complexity. The plot reflects the different development phases.

Considerations: shortcomings

- Some design choices would have probably been different if designed now.
  - Tcp/Ip interface: the CS uses a custom protocol and implementation. An alternate approach could be a XML based interface, with SOAP message passing. This would also make it easy to enable a GRID interface to parts of the system.
  - Programming language: the system was programmed in OO perl. While the dynamic typed nature of perl has been essentially useful, some discipline and conventions were needed to keep the code base maintainable. A language like python would be a sensible alternative.
  - Overdesign: parts of the system were designed with too generic purposes, and then not used or not fully exploited. This happens naturally whenever the requirements must be discovered or refi ned during the development, though.

Summary and comments

- The experience with the BaBar reconstruction Control System displays how a careful design can grant qualities to a software system.
  - In this talk, we often pictured the system as self-evolving, to analyze its behavior. That is of course not the case.
  - Any stage of a system's maintenance require thoughtful decisions, to maintain consistency with the design.
  - Higher level design patterns offer a guidance to those decisions. They won't make the system immune to later bad design or bad coding.
- The solutions discussed in this talk contributed substantially to implement a flexible and maintainable system.

Who

- Main developers
  - Antonio Ceseracciu
  - Martino Piemontese
  - Francesco Safai Tehrani
- Co-developers and testers
  - Peter Elmer
  - Doug Johnson
  - Teela Marie Pulliam
  - Fulvio Galeazzi
- Helpers and Users
  - Sridhara Dasu
  - Adil Hasan
  - Doug Johnson
  - Olga Igonkina
  - Christian Flacco
  - ...to name a few. Thanks!