GAUDI - A Software Architecture and Framework for Building HEP data processing Applications

Developed in the context of the LHCb Experiment

Pere Mato, CERN
August 1999
Outline

- Introduction
  - The LHCb experiment
  - Computing organization strategy
  - Software development strategy

- GAUDI architecture
  - Goals and scope
  - Design choices
  - Selected topics

- Implementation
  - Technology choices, Tools, ...
  - Schedule so far
LHCb: Physics Goals

- LHCb is a dedicated experiment at LHC collider for precision measurements of CP-violation and rare decays
  - CP violation currently observed in kaon decays is consistent with Standard Model, but cannot exclude that CP violation is partly or even entirely due to new physics.
  - Cosmology (baryon genesis) suggests that an additional source of CP violation other than the Standard Model is needed.

- LHC is an ideal place to produce lots of $B_d$ and $B_s$
- All interesting decay channels have $10^{-5}$ visible branching fractions.
LHCb: The Detector
LHCb: The Detector

- Single-arm spectrometer with forward angular coverage from ~10 mrad to ~300(250) mrad.
  - Vertex detector
    » Si r-φ strip detector, single-sided 150 µm
  - Tracking system
    » Outer: drift chamber honeycomb. Inner: MSGC with GEM or MCSC
  - RICH system
    » RICH1: Aerogel + C₄F₁₀. RICH2: CF₄
  - Calorimeter system
  - Muon system
    » Multi-gap RPC and CPC
LHCb in numbers

- Collaboration: ~45 Institutes, ~350 participants
- Cost of the experiment: 86 MCHF
- Electronics: ~$10^6$ readout channels
- Trigger System: 4 Levels. 40 MHz → 1 MHz → 40 kHz → 5 kHz → 200 Hz
- Data Acquisition: 100 kB/event. 2-4 GB/s → 20 MB/s. 1.5 $10^6$ MIPs
- Status of the Experiment:
  - Technical proposal submitted in February 1998
  - Approved in September 1998
  - R&D phase for ~2 years
LHCb Computing: Goals

- Need to focus on quality but at the same time be efficient in use of resources

- Identify all the roles and their responsibilities
  - system architect, project leaders, librarian, framework developers, client developers, ...

- Good communication to arrive at common aims and understanding
  - need to know at all times what everybody is doing (regular meetings)
  - Procedures for taking decisions must be agreed and followed

- Common language supported through documentation and training
  - Web
  - Handbooks (user, engineering, management)
  - Formal training
Organizing software development activities

**Manage**
Plan, initiate, track, coordinate
Set priorities and schedules, resolve conflicts

**Build**
- Develop models, Evaluate toolkits
- Architect components and systems
- Choose integration standard
- Engineer reusable components

**Support**
- Support development processes
- Manage and maintain components
- Certify, classify, distribute
- Document, give feedback

**Assemble**
- Design application
- Find and specialise components
- Develop missing components
- Integrate components

Requirements
Existing software

August, 1999  GAUDI
Project Organization

Support
- Facilities
  - CPU farms
  - Desktop
  - Storage
  - Network
  - System Man.

Arch. Review
- M
- A
- E
- ...

Technical Review
- E
- M
- A
- ...

Steering Group
- C
- M
- C
- ...

Frameworks
- Architecture, Components, Integration technology, Libraries and toolkits

Analysis
- M

Reconstruction
- M

Simulation
- M

Controls
- M

DAQ
- M

Control Room
- M

Assemble

Build

Support
- Software
  - SDE
  - Process
  - Quality
  - Librarian
  - Training
  - Webmaster

Vendors
- IT-PDP

Vendors
- IT-ASD

August, 1999

Coordinator
- C

Architect
- A

Project Manager
- M

Project Engineer
- E
Strategy for development of new software

- Start with a small design team 6-8 people
  - architect, librarian, domain specialists with design/programming experience
- Collect URs and scenarios, use to validate design
- Establish the basic criteria for overall design
- Make technology choices for implementations of first prototypes
- Incremental approach to development. Releases every ~4 months.
- Development cycle is user-driven. Priorities, feedback, etc.
- Strategic decisions taken following thorough review (~1/year)
- Releases accompanied by complete documentation
- Expand development team to cover new domains
LHCb Offline software road map

Integration and Commissioning

Detailed Implementation

Working Prototype, ‘retire’ SICB

Exploitation

Incremental releases

Release Number

Major Technology Choices

Review

GAUDI Architecture

Antoni Gaudí
Barcelona (1852-1926)
Modernist style architect
GAUDI: Scope

- We want to develop an Architecture and Framework to be used in ALL the LHCb event data processing applications in all stages:
  - High level trigger, simulation, reconstruction, analysis.
  - Control applications are not included (slow control, run control)
- Physicists will develop applications by customizing the framework by subclassing, composing and configuring.
- Components will be developed in general using other specialized frameworks (GUI, object persistency, simulation, …)
GAUDI: Expected Benefits

- Allow physicists to focus on solving the physics problem.
  - Must be easy to use
  - Non physics-related functionality implemented by the framework
- Common vocabulary, better specifications of what needs to be done.
- Ensure low coupling between concurrent developments.
- Guarantee a later smooth integration of developments.
- Facilitate software re-use
Software Structure

A series of data processing applications built on top of the frameworks and implementing the required physics algorithms.

A series of Frameworks and Toolkits. One main framework: **GAUDI**, various specialized frameworks: visualization, persistency, interactivity, simulation (Geant4), etc.

A series of basic libraries widely used: STL, CLHEP, etc. (Vocabulary)

Frameworks

Toolkits

Analysis

Simulation

Reconstruction

High level triggers

Foundation Libraries
Major design criteria

- Separation between “data” and “algorithms”
- Three basic types of data:
  - event data (data obtained from the particle collisions)
  - detector data (structure, geometry, calibration, alignment, environmental parameters, ..)
  - statistical data: (histograms, …)
- Separation between “persistent data” and “transient data”.
  - Isolation of user’s code.
  - Different/incompatible optimization criteria.
  - Transient as a bridge between various representations.
Major design criteria (2)

- Data store centered architectural style.
  - “Algorithms” as data producers and consumers.

- *User code* encapsulated in few specific places:
  - “Algorithms”: Physics code
  - “Converters”: Converting data objects into other representations

- All components with well defined “interfaces” and as “generic” as possible.

- Design principles
  - Low coupling, inheritance, static storage, ...
Object diagram
Interfaces

- Each component implements a number of interfaces
- Each component uses a number of interfaces from other components
- An Algorithm uses many Services
Algorithms

- Each Algorithm only knows what data (type and name) is expecting as input and creating as output.
- The only coupling is through the data.
- Scheduling of sub-algorithms is the responsibility of the parent algorithm.
Services

- Services are provided to *Algorithms*
- Examples:
  - Job Options service (card files)
  - Message reporting service
  - Event/Detector/Histogram data service
  - Event Selector
  - Persistency and Conversion services
  - User Interface (GUI)
  - Particle property service
  - ...
Event Data Store

Persistency Service

Transient Event Store

event Data Service

Algorithm

- Store objects for use of others
- Retrieve objects when needed
- Tree structure (file system)
- Identification by logical address ("/Event/MC/MCEcalHits")
- Store owns the objects. Responsible for cleanup

retrieveObject("MCEcalHits", ...)

registerObject("key", ...)

Direct reference

Fetch()
Store()
Event Persistency

Several technologies available simultaneously: Objy, Root, Zebra,...

**Converters** to transform objects from one representation to another

**Generic links** between objects in the persistent world
Detector Description

Data Processing Application

Algorithms

Conversion services

Transient detector store

DetElem

Geom

Calib

Slow

Other representations

Projection view: version & event time

Persistent Detector Description (DDDB)

DetElem

Geom

Calib

Slow

Editors

Detector Data Producers

Update persistency
Data Visualization

Transient Event/Detector Store

Converter

Data Item Selector

• Selects objects in store

Conversion Service

Representations Store (graphical, textual)

User Interface

• Selects objects in store
What are the knobs at our disposal?

– JobOptions. Simple usage. It allows the end-user to overwrite any property of any algorithm or service.

– Algorithm/Service properties database. A more sophisticated way to modify the properties of the algorithms and services.

– Detector database edition to create new versions or releases.

– Write specific code. Configure your application by setting properties at runtime.

– User interface component. Graphical (a la Visual Basic), command line (scripting language), etc.
• **Physical design** (packaging) is an architectural issue.
  • Big consequences on:
    • compilation time
    • link dependencies
    • configuration management
    • executable size
    • ...
  • Package interdependencies require approval of architect.
  • Avoid cyclic dependencies
Implementation

◆ Platforms:
  – **WNT, Linux, IBM AIX, HP-UX**

◆ Tools and Libraries:

<table>
<thead>
<tr>
<th>Design tools</th>
<th>Visual Thought, Rational Rose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding rules</td>
<td>Interim LHCb coding conventions, SPIDER</td>
</tr>
<tr>
<td>Code Management</td>
<td>Visual Source Safe, CVS</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>CMT</td>
</tr>
<tr>
<td>Problem tracking</td>
<td>Planned to use Remedy</td>
</tr>
<tr>
<td>Compilers/Debuggers</td>
<td>Visual C++, GNU EGCS</td>
</tr>
<tr>
<td>Libraries</td>
<td>STL, CLHEP, NAG C, HTL, RIO</td>
</tr>
<tr>
<td>Documentation</td>
<td>FrameMaker</td>
</tr>
<tr>
<td>Source code documentation</td>
<td>Object Outline, DOC++</td>
</tr>
</tbody>
</table>
Schedule so far

- Sept 98 - architect appointed, design team 6 people assembled
- Nov 25 ’98 - 1 day architecture review
  - goals, architecture design document, URD, scenarios
- Feb 8 ‘99 - GAUDI first release
  - first software week with presentations and tutorial sessions
  - plan second release
  - expand GAUDI team
- May 30 ‘99 - GAUDI second release
  - second software week …
  - plan third release
  - expand GAUDI team (GEANT4 simulation toolkit)
- Nov ‘99 - next GAUDI release and software week planned
Conclusions

- Almost completed the first year of the journey towards O-O
  - Architecture being defined (interfaces, functional components)
  - Two releases of the framework with basic functionality to test the ideas.

- Currently working on:
  - Integration of GEANT4
  - Data visualization, event display
  - Detector description
  - Algorithms and tools for data analysis
  - Java evaluation

- We would like to get advice from experienced people
  - Organization, physical design, development tools, libraries, foundation libraries, etc.
  - Strategic design decisions, existing frameworks and solutions, etc.