GAUDI - The Software Architecture and Framework for building LHCb data processing applications

Marco Cattaneo, CERN
February 2000
Outline

- Introduction
- Design choices
- GAUDI Architecture overview
- Status
- Conclusions
Applications implementing the physics algorithms.

One main framework: **GAUDI**.

Various specialised frameworks: visualisation, persistency, interactivity, simulation (Geant4), etc.

Basic libraries: STL, CLHEP, etc. *(Vocabulary)*

**Frameworks Toolkits**

**Foundation Libraries**

Triggers

Reconstruction

Simulation

Analysis
Develop an Architecture and a Framework to be used at all stages of LHCb data analysis

- Trigger levels 2 and 3, simulation, reconstruction, analysis

Avoid fragmentation and duplication of computing effort

- Single development team across online and offline domains
  - Identify common components, re-use
- Give users (physicists) a framework within which to develop applications
  - Rapid transition away from FORTRAN to minimise legacy code
  - A single framework used by all members of the collaboration

Transparent use of third-party components wherever possible or necessary

- GUI, persistency, simulation....
Software development strategy

- Start with small design team of 6-8 people
  - architect, librarian, domain specialists with design/programming experience
- Collect User Requirements and use-cases
- Establish basic criteria for the overall design
- Make technology choices for implementation of initial prototypes
- Incremental approach to development.
  - Release every ~4 months.
  - Releases accompanied by complete documentation
  - Development cycle driven by the users: priorities, feedback, etc.
- Strategic decisions after thorough design review (~1/year)
Principal design choices

- **Separation between “data” and “algorithms”**
  - Data objects primarily carry data, have only basic methods
    - e.g. Tracking hits
  - Algorithm objects primarily manipulate data
    - e.g. Track fitter

- **Three basic categories of data:**
  - “event data” (obtained from particle collisions, real or simulated)
  - “detector data” (structure, geometry, calibration, alignment, ....)
  - “statistical data” (histograms, ....)

- **Separation between “transient” and “persistent” data.**
  - Isolate user code from persistency technology.
  - Different optimisation criteria.
  - Transient as a bridge between independent representations.
Principal design choices (2)

- **Data store-centred (“blackboard”) architectural style.**
  - Algorithms as producers and consumers of data objects
  - Minimal coupling between algorithms, allows independent development.

- **“User code” encapsulated in a few specific places:**
  - “Algorithms”: physics code
  - “Converters”: convert data objects between representations

- **Well defined component “interfaces”, as “generic” as possible.**
  - Stable, shield clients from the (changing) implementation
  - In C++, pure abstract class
- Each component **implements** one or more interfaces
- Each component **uses** one or more interfaces of other components
- An **Algorithm** uses many **Services**
An Algorithm knows only which data (type and name) it uses as input and produces as output.

The only coupling between algorithms is via the data.

The execution order of the sub-algorithms is the responsibility of the parent algorithm.
Various services are provided to algorithms

Examples:

- Job Options service (configuration “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
(See Markus Frank's talk, C153)

- Stores objects that can be used by other objects (services, algorithms).
- Retrieve objects if necessary.
- Tree structure (file system).
- Identification by logical address ("/Event/RawEvent/Ecal")
- Owns the objects and is responsible for their clean-up.

Persistency Service

Event Data Service

Algorithm

Transient Event Store

RetrieveObject("EcalDigits(3)",...)

registerObject("key",...)

createObject()
Persistency
(See Markus Frank’s talk, C153)

- Various technologies available in the same program: Objy, Root, Zebra,…
- **Converters** transform objects from one representation to another.
Detector Description
(See Radovan Chytracek’s talk, A155)

Data Processing Application

Editors

Persistent Detector Description (DDDB)

Projection view: version & event time

Update persistency

Editors

Transient detector store

Other representations

Conversion services

Algorithms

Detector Data Producers

Marco Cattaneo, 7th February 2000
LHCb - GAUDI
Sub-division into packages is an architectural problem.

Important consequences for:
- compilation time
- link dependencies
- configuration management
- executable size
- ...

Dependencies between packages must be approved by the architect

Avoid circular dependencies
### Implementation
(see Florence Ranjard’s talk, F151)

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

- **Tools and Libraries:**

<table>
<thead>
<tr>
<th>Category</th>
<th>Tools/Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design tools</td>
<td>Visual Thought, Rational Rose</td>
</tr>
<tr>
<td>Coding rules</td>
<td>Interim LHCb coding conventions</td>
</tr>
<tr>
<td>Code Management</td>
<td>CVS</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>CMT</td>
</tr>
<tr>
<td>Problem tracking</td>
<td>Plan to use Remedy</td>
</tr>
<tr>
<td>Compilers/Debuggers</td>
<td>Visual C++, GNU EGCS, ddd</td>
</tr>
<tr>
<td>Libraries</td>
<td>STL, CLHEP, NAG C, HTL, RIO</td>
</tr>
<tr>
<td>Documentation</td>
<td>FrameMaker, Visual Source Safe</td>
</tr>
<tr>
<td>Source code documentation</td>
<td>Object Outline</td>
</tr>
</tbody>
</table>
Work in progress:

- Integration of GEANT4
- Visualisation, event display
- Algorithms and tools for data analysis
- Java evaluation
- Collaboration with AIDA (see Andreas Pfeiffer's presentation, F82)
  - Definition of interfaces
- Ongoing discussions with other experiments
- Deployment for physics applications:
  - migration of reconstruction
  - test beam analysis
  - tracking, RICH pattern recognition
  - ECAL geometry
  - etc.
Conclusions

- We believe it is fundamental to define an architecture
  - And to provide a framework which implements the architecture
  - Ensures adaptability, maintainability and resilience against change.
  - GAUDI is the LHCb architecture and framework

- Physicists have started to enjoy the pain!
  - Many new development activities entirely within Gaudi
  - Ongoing migration of existing code to Gaudi framework

- We welcome advice, criticism, collaboration

Software Project Organisation

Steering Group
- Coordinator (C)
- Architect (A)
- Project Manager (M)
- Project Engineer (E)

Architectural Review
- M Architecture
- A Environment
- E Engineering

Technical Review
- E Engineering
- M Management
- A Architecture

Assemble
- M Assembly
- DAQ Data Acquisition
- M Controls

Build
- M Build
- Reconstruction
- Simulation
- Analysis

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

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

Vendors
- IT-SD

Vendors
- IT-API
Project history

- Sep '98 - architect appointed, design team (6 people) constituted
- Nov 25 '98 - external architecture review
  - objectives, architecture design document, URD, scenarios
- Feb 8 '99 - first GAUDI release
  - first software week, presentations, tutorials
  - plan second release (together with users)
  - expand GAUDI team
- May 30 '99 - second GAUDI release
  - second software week, plan third release with users, expand team.
- Nov 23 '99 - third GAUDI release and software week
  - plan deployment for production applications
- Spring '00 - second external review
Migration Strategy

Objective: all applications exclusively in OO

Transition phase
- Incorporate existing reconstruction and analysis programs in GAUDI (wrap FORTRAN)
  - Split existing program into independent algorithms
  - Develop an OO event model, write converters to populate it from the FORTRAN banks
- Incorporate new OO algorithms developed exclusively in GAUDI
  - e.g. Tracking pattern recognition
  - Write converters to make the results available to the FORTRAN world
    - Many converters in both directions, COMMON blocks etc.

Hybrid phase
- C++ and FORTRAN coexist in a single reconstruction program
  - Two detector descriptions, two cards files, doubled memory use, which output format?
- Gradually replace FORTRAN