Software Review Panel

LHCb Answers to Architecture, Data Model and Program Infrastructure

Pere Mato for the LHCb Collaboration
15th March 2000

What requirements, current and future, have you identified for your software architecture or 'framework'?

- A set of architecturally significant scenarios (“use-cases”) were collected for the design of the architecture (Nov 98)
  - http://lhcb.cern.ch/computing/offline/pdf/gaudiscenarios.pdf
- Scenarios provided from different “stakeholders” by means of personal interviews.
  - Users: physicist users, physicist developers, data production managers, framework developers.
  - Managers, libraries, etc.
- Scenarios capture the kind of functions and qualities the system must satisfy.

What requirements, current and future, have you identified for your software architecture or 'framework'?

- We divided the scenarios into four categories according to the type of user:
  - (A) Scenarios which deal with the **use** of applications built within the framework. **Functionality** of the system.
  - (B) Scenarios which deal with the **development** of components built within the framework.
  - (C) Scenarios which deal with **configuration** management.
  - (D) Scenarios which deal with the interaction of the framework with the **environment** and handling of the **change**.
- Architecture review (26th Nov 1998)

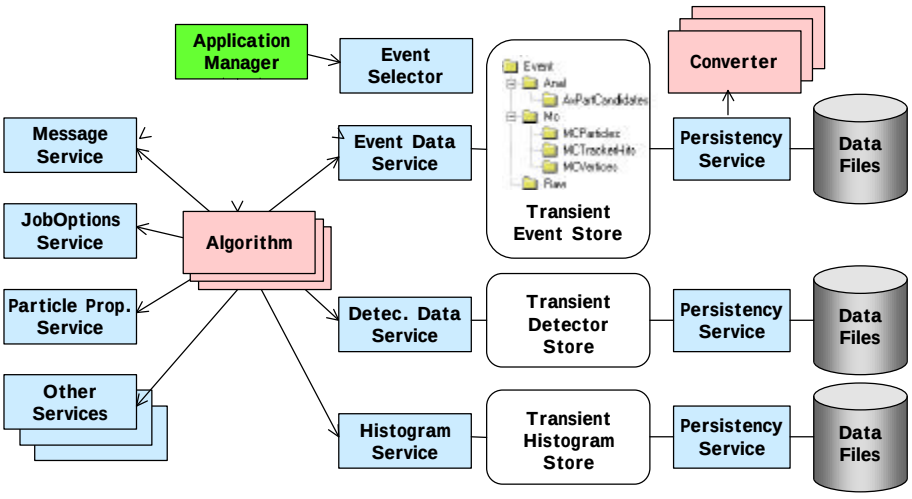
Requirements: How will they evolve?

- We have assumed that we will never know the complete set of requirements.
  - The traditional "water-fall" model does not work.
  - We will discover the requirements during subsequent development iterations.
- We are convinced that the USDP software development process easily copes with the evolution of requirements
  - USDP = use-case driven, architecture-centric, iterative and incremental
- In each development iteration of the framework (release every 3 or 4 months) we refine and collect new requirements
  - Sometimes even contradicting requirements !!
- Our philosophy: *"start very simple and add the complication later if needed"*

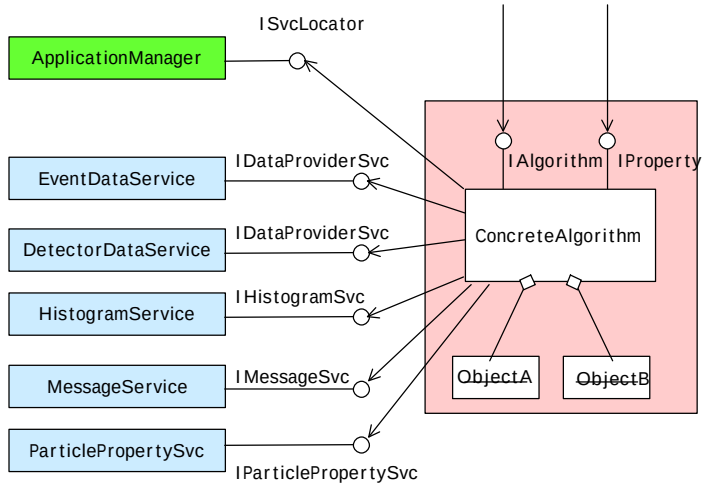
Requirements: Language evolution?

- The current implementation of the architecture (GAUDI) is based on C++.
- We did foresee the *scenario* of a possible change of programming language during the design of the architecture.
- GAUDI was designed with “Java in mind”
  - Use constructs which are possible to implement with Java (avoid multiple inheritance, interfaces, avoid templates, …)
- We are currently doing an evaluation of Java
  - Gather information for an eventual decision for migration to Java
  - Translation of the GAUDI framework into Java

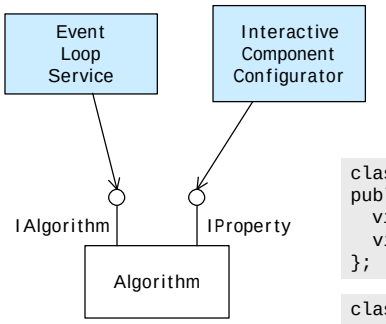
GAUDI Architecture



GAUDI: Interface Model



Interface Model (2)



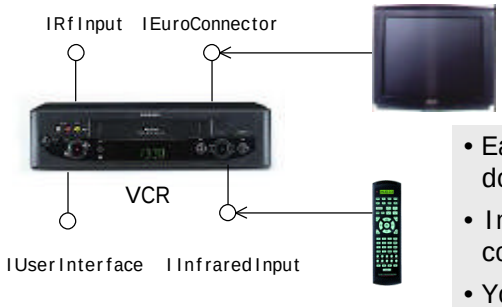

```
class IAlgorithm : virtual public IInterface {
 public:
  virtual StatusCode initialize() = 0;
  virtual StatusCode execute() = 0;
  virtual StatusCode finalize() = 0;

  virtual const std::string& name() const = 0;
  virtual StatusCode sysInitialize () = 0;
  virtual StatusCode sysFinalize () = 0;
};
```

```
class IProperty : virtual public IInterface  {
public:
  virtual StatusCode setProperty (const Property& p) = 0;
  virtual StatusCode getProperty (Property *p) const = 0;
};
```

```
class Algorithm : virtual public IAlgorithm,
                  virtual public IProperty {
public:
 ...
}
```

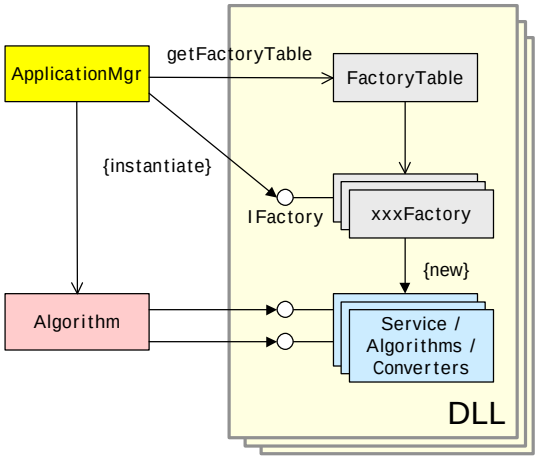
VCR Interface model



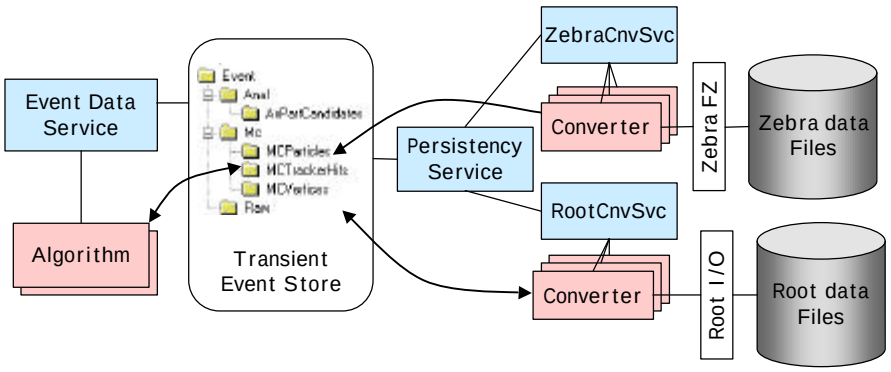
- Each interface is specialized in a domain.
- Interfaces are independent of concrete implementations.
- You can mix devices from several constructors.
- Application built by composing.
- Standardizing on the interfaces gives us a big leverage.

Factories & Dynamic Loading

- **Plug-and-Play**
- **Factory pattern** to avoid using concrete implementation.
- Run-time discovery of components.
- Only pure abstract classes (**interfaces**) are accessible.

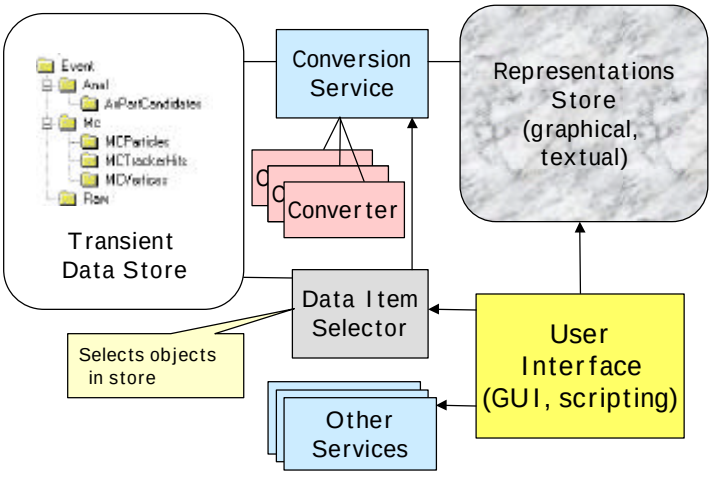


Persistency



- Various technologies available in the same program: Objy, Root, Zebra,…
- **Converters** transform objects from one representation to another.

User Interaction / Visualization



Do you believe that your requirements for framework, software build and release process and data persistency are distinct and different from the other experiments?

- Framework
  - NO
- Software build and release process
  - NO
- Data persistency
  - NO. Even less demanding than the other experiments.

- The requirements are the same (very similar), the implementation framework could also be the same**.**
  - Event and Detector data model and Algorithms are LHCb specific
- The GAUDI framework is not LHCb specific.

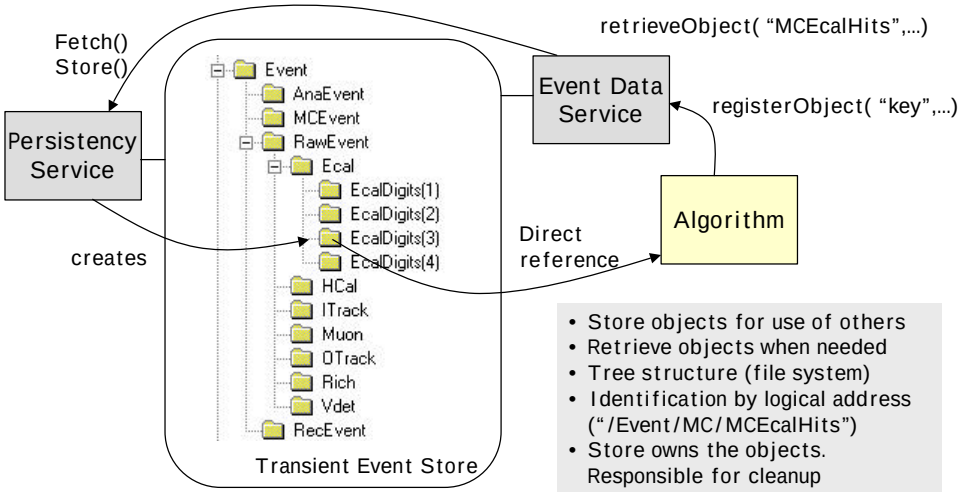
Do you believe that your requirements for framework, software build and release process and data persistency are distinct and different from the other experiments?

- Why are we not sharing a common framework?
  - The list of conceptual requirements is probably the same but they are weighted differently. Different starting assumptions.
  - At the time you need a framework, the candidate does not exists or you are unable to find.
  - If embarked in a new design, you will end up with a different solution.

What do you mean by your data model? How is your data model defined?

- In GAUDI
  - We separate “data” from “algorithms”
  - We separate between “persistent data” and “transient data”.
  - Algorithms are “producers” and “consumers” of named data into transient data stores (tree like structure).
- Event Data Model
  - “Structure of the transient event data made available to *Algorithms*”
- How is the data model defined?
  - Named objects inherits from *DataObject* and are typically collections of small objects of a C++ class defined in a header file. In principle any class can be a contained object. Relationships implemented using *smart references* (load on demand)

Event Data Store



How does this interact with the data persistency mechanism? Language choices? Analysis tools?

- GAUDI separates “transient” and “persistency” data representations
  - Physics code independent of the persistency technology
  - Different optimization criteria.
  - Transient representation as a bridge between various persistent representations.
  - Possibility to change persistency solution with small impact.
- The Data Model is completely independent from the persistency technology we are using or going to use in the future.
- The current Data Model definition is tighten to C++. Would be better to define the data model using a programming language independent object definition language.
  - Code generation to various languages
  - Mixing languages (Java, C++)

How does this interact with the data persistency mechanism? Language choices? Analysis tools?

- Analysis tools
  - Existing analysis tools are bounded to a specific object persistency solution.
  - This is a problem.
  - The current solution is to produce statistical data (histograms, n-tuples) on the form the analysis tool require.
  - It would be nice to be able to plug other object persistency solutions. The experiment data model could be made available to the analysis tool.

What infrastructure will be used to assure that all conditions, parameters and code which were used to create a data object are codified and known?

- The current infrastructure is based on the **configuration management system** and the **bookkeeping** database
- Configuration Management
  - The version of the code and some input data files (particle decays, detector description, etc.) is controlled using a set of configuration management tools: **CVS** to manage code repository and **CMT** to manage the build and release of versions of the software.
- Bookkeeping database
  - Contains the list of all data sets available. Each data set is qualified with a set of parameters to allow sophisticated selections. Based on ORACLE.
- Plan to enhance the system with a package similar to the Run Control Parameter (RCP) of Fermilab.

What mechanisms are you using to assure that the core infrastructure components of the software have broad experiment input, validation and testing?

- Participation in defining the requirements
  - Collected use-cases or scenarios by personal interviews.
- Weekly computing meetings between subdetectors and core team
  - Framework & other projects status reports. Presentations from subdetectors.
- Incremental releases
  - Early usage. Feedback from users.
- Software weeks (coinciding with releases of framework)
  - Presentations and Tutorials.
  - Planning with subdetectors what goes into the next release.
- Subdetector software reviews
  - Feedback of framework. How it is used. New requirements.

What requirements do various software milestones, and other experiment milestones, place on functionality and timescale for delivery of core infrastructure components? Will these be met?

- The experiment milestones (mainly subdetector TDR's) influences requirements, priorities and timescale of infrastructure components.
  - Availability of people for developing the new software
  - Need to produce results for detector studies
- For the TDR's to be submitted this year has been decided to use the old FORTRAN algorithms "wrapped" into the new GAUDI framework
  - Requirements: wrapping FORTRAN should possible, production quality, conversion of data from/to FORTRAN to/from C++.
  - Detector description duplication (possible inconsistencies)

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
  - expand GAUDI team
- May 30 '99 - GAUDI second release
  - second software week …
  - expand GAUDI team (GEANT4 simulation toolkit)
- Nov 23 '99 - GAUDI third release
  - Functionally complete version (basic services, access to SICb data, detector description framework, histograms, n-tuples, examples, …)