

OPTICON Network 3.6

Future Astronomical Software Environments



Preben Grosbøl
DMD/ESO

SAMPO Kick-off meeting
January 13-14, 2005

Outline

- OPTICON Network 3.6 on software
- Vision and Scope
- User Scenarios
- High-level Requirements
- Architectural Concept



Data Analysis in Astronomy

→ Issues for astronomers doing data analysis:

✓ many different systems exist but:

- they are largely **incompatible** with respect to scripting language

- it is **complicated** to share data between them

- interesting application from other systems cannot be easily used

✓ they provide only **limited interfaces** to Web services and archive

✓ it is difficult to fully exploit available computer resources

→ Solutions may involve:

✓ creating **common environment** for data analysis

✓ making important **legacy applications** available

✓ providing **easy development** platform for new tasks

✓ increasing the ability to **collaborate** and of **sharing** of software

✓ supporting **easy access to resources** e.g. archives, VO, and GRID

OPTICON Network 3.6 - Objectives

- Network on Future Astronomical Software Environments:
 - ✓ Funded through OPTICON EU FP6 proposal
 - ✓ Official start 2004-01-01 for 4 years
 - ✓ Main objectives are to:
 - establish **high-level requirements** for an astronomical software environment
 - identify areas where a **common environment** is desirable and feasible
 - recommend **architecture** and **high-level design** for these areas
 - draft **interface specifications** for the environment
 - ✓ OPTICON funds only network activities
 - discussions on software environments
 - face-to-face meetings etc.
 - but no software developments

Schedule and Members

→ General schedule for activities:

- ✓ define general **scope** - 2004
- ✓ draft **high-level requirements** - 2004/2005
- ✓ propose **architectural concept** and **high-level design** - 2005
- ✓ outline **standards** and **interface specifications** - 2006/2007

→ Network participants:

- ✓ 20 members from Europe
- ✓ 6 members from North America

→ Methodology

- ✓ monthly phone meetings
- ✓ TWiki site
 - **<<http://archive.eso.org/opticon/twiki/bin/view/Main>>**
- ✓ e-mail lists
 - **face@eso.org** for discussions of requirements and architecture
- ✓ face-to-face meetings (e.g. 2004-12-02/03, ESO)

Vision Statement

- A future environment must offer:
 - ✓ important **legacy applications** from current systems
 - ✓ simple **development** of new tasks
 - ✓ ability to collaborate on and **share software**
 - ✓ **easy access** to archives, VO and GRID
 - ✓ **efficient desktop** workplace for astronomers
 - ✓ **trivial installation** on standard desktops
- Success of any such environment will depend on:
 - ✓ availability of **new state-of-the-art applications**
 - ✓ **reliability** of the full environment
 - ✓ **stability** of interfaces to applications
 - ✓ **flexible** design isolating astronomical tasks from IT world
 - ✓ user **support** including documentation and maintenance

Scope of Network

- Main scope of the network discussions is:
 - ✓ outline **user scenarios** for environment
 - ✓ discuss **reasons** for creating a new environment
 - ✓ learn from good and bad **experiences** with current systems
 - ✓ define high-level **astronomical requirements**
 - ✓ consider **architectural concept** and technical requirements
 - ✓ specify **standard interfaces** between parts of the system
 - ✓ list mandatory and optional **services** provided by environment
 - ✓ outline **development/deployment** strategy
 - ✓ identify **relevant standards** for adding new packages

Free, Open Source

→ Free vs. licensed software:

- ✓ full base version must be available without license fees
 - ensure wide availability even on personal systems
- ✓ extension and enhancement with commercial products possible
 - optimization for special usage e.g. embedded systems

→ Open vs. restricted source code

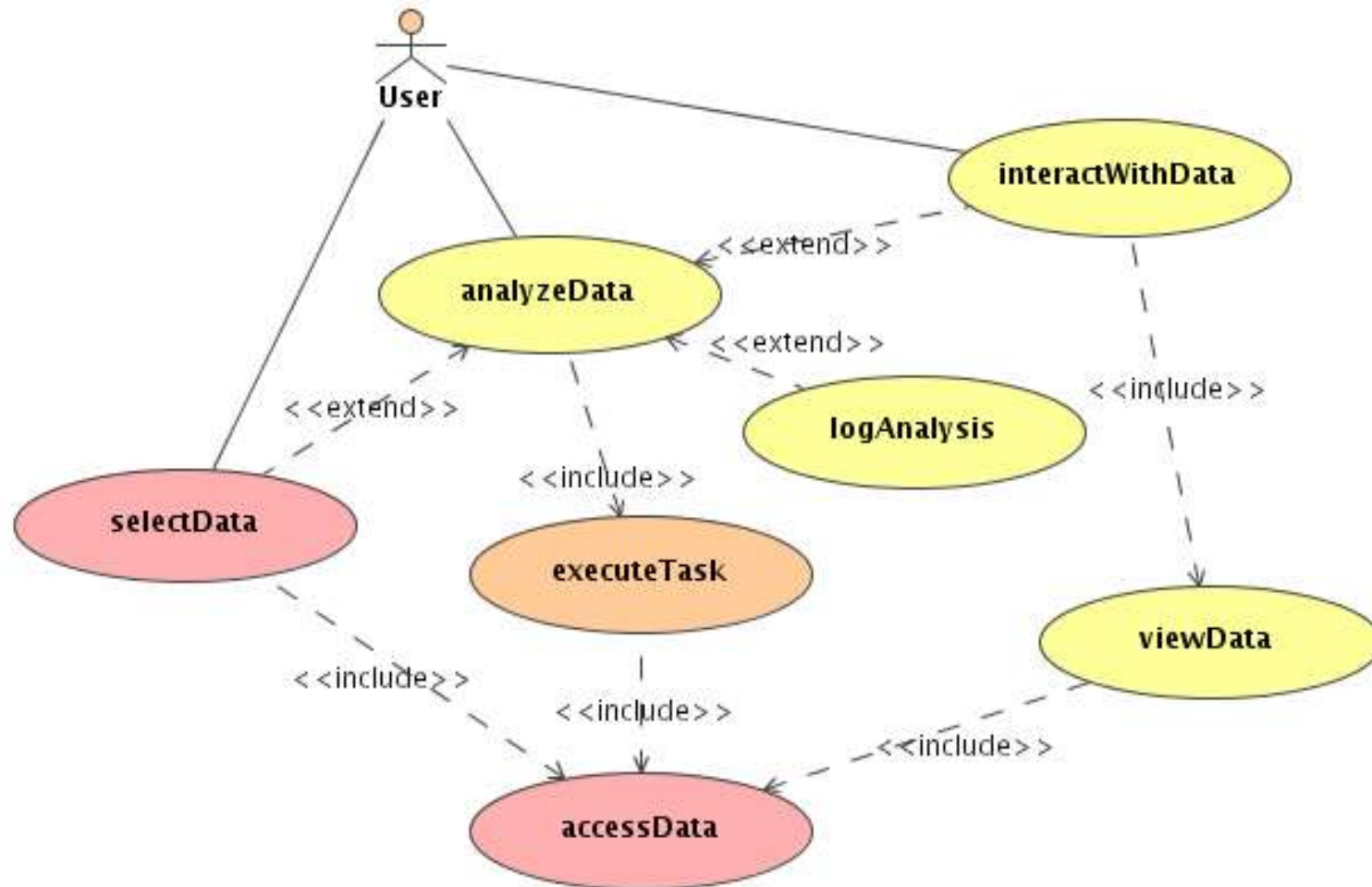
- ✓ all scientific code must be open source
 - essential for verification of results
 - pure mathematical libraries may be an exception if verifiable
- ✓ all interface specification must be open
 - allow multiple implementations of environment
 - configuration control by independent body

→ Recommendation to follow GNU Public License policy

User Scenarios

- Single user **analyzing** her own data on a desktop
 - ✓ standard **processing** and **analyzing** of astronomical data
 - ✓ work on **stand-alone** system e.g. laptop
- Student **developing** a new application
 - ✓ **development** cycle of new tasks
 - ✓ **distribution** including documentation and packaging
- Astronomer studying **large, complex, multi-wavelength** data
 - ✓ access to large, **distributed data** sets (e.g. VO)
 - ✓ usage of **distributed computer** resources (e.g. GRID)
- **Team** collaborating on a common data set
 - ✓ **distributed team** of people
 - ✓ access to **common** data
- **Integrator** of a specific system
 - ✓ integration and testing of **complete** system for specific usage

Use Cases



Typical top-level UML Use Case for astronomer analyzing large data set

Requirements - Environment

→ Environment itself:

- ✓ **easy to install** on typical systems including desktops
- ✓ standard system **freely** available and **open** source
- ✓ simple to add **new application** code for typical astronomer
 - support for languages like **FORTRAN**, **C**, **C++**, **Java** and **Python**
- ✓ **standards** should be defined for:
 - **documentation** and help
 - **error** handling and **logging**
 - **test** and validation of both system and scientific tasks
 - distribution of **patches** and updates
 - interface to **commercial** packages
- ✓ use **existing standards** whenever possible
- ✓ **revision control** of all code, documentation etc.
- ✓ support for **internationalization** of user interaction

Requirements - System

→ Scripting and execution:

- ✓ execution of individual task from **command line**
- ✓ standard for **passing parameters** between tasks
- ✓ scripting with both **interactive** and **batch** modes
- ✓ **scalability** with respect to data and computation
- ✓ **full logging** of tasks executed including
 - host, version, parameters, results and possible errors/warnings
 - user comments and notes (e.g. worksheets)
 - repeat of tasks previously recorded
- ✓ **location** of data and computation **transparent** to user
 - user may decide where things are done
- ✓ access to Web, VO and GRID **services**
- ✓ support of **GUI's** for task execution and workflow definition

Requirements - Data

→ Data structures and access:

- ✓ scalar and array data of **standard types** supported
- ✓ definition of **groups** or **collections** of data
- ✓ support for **units**, **errors** and **flags** including **propagation**
- ✓ standard **coordinate**, **time** and **unit** systems available
- ✓ typical **mathematical operations** available for relevant data
- ✓ support for **database/table** functionality
- ✓ **sharing**, **locking** and **read-only** data supported
- ✓ **standard data formats** for in-/output e.g. FITS and VO
- ✓ manipulation of **large data** sets possible
- ✓ change **history** associated to data

Requirements - Other

→ Visualization:

- ✓ graphical representation of **data** including their **errors**
- ✓ comparison of **different data** for same object/field
- ✓ overlay of maps defined in **different coordinate** systems

→ Support for modeling and testing:

- ✓ **classification** of data
- ✓ **statistical** tests
- ✓ **robust** estimators
- ✓ **comparison** between data and model

Architecture - Perspectives

→ User perspective

✓ astronomer

- uses system, as-is, for processing and analysis of data
- accesses to wide range of data sources e.g. archives, VO

✓ developer

- writes new application code
- uses system interfaces, libraries and services

✓ integrator

- builds end-to-end system for specific purpose

→ Reference Use Cases

✓ desktop data processing and analyzing

✓ VO data access and analysis

✓ pipeline processing

Architecture - Layered Concept

→ Four Layered Component-Framework Architecture

✓ Presentation layer

- interface to 'outside' world e.g. users, browsers, ...

✓ Application layer

- top-level scripting applications using components

✓ Execution Framework

- distributed virtual machine e.g. through GRID

- package manager administers component-containers

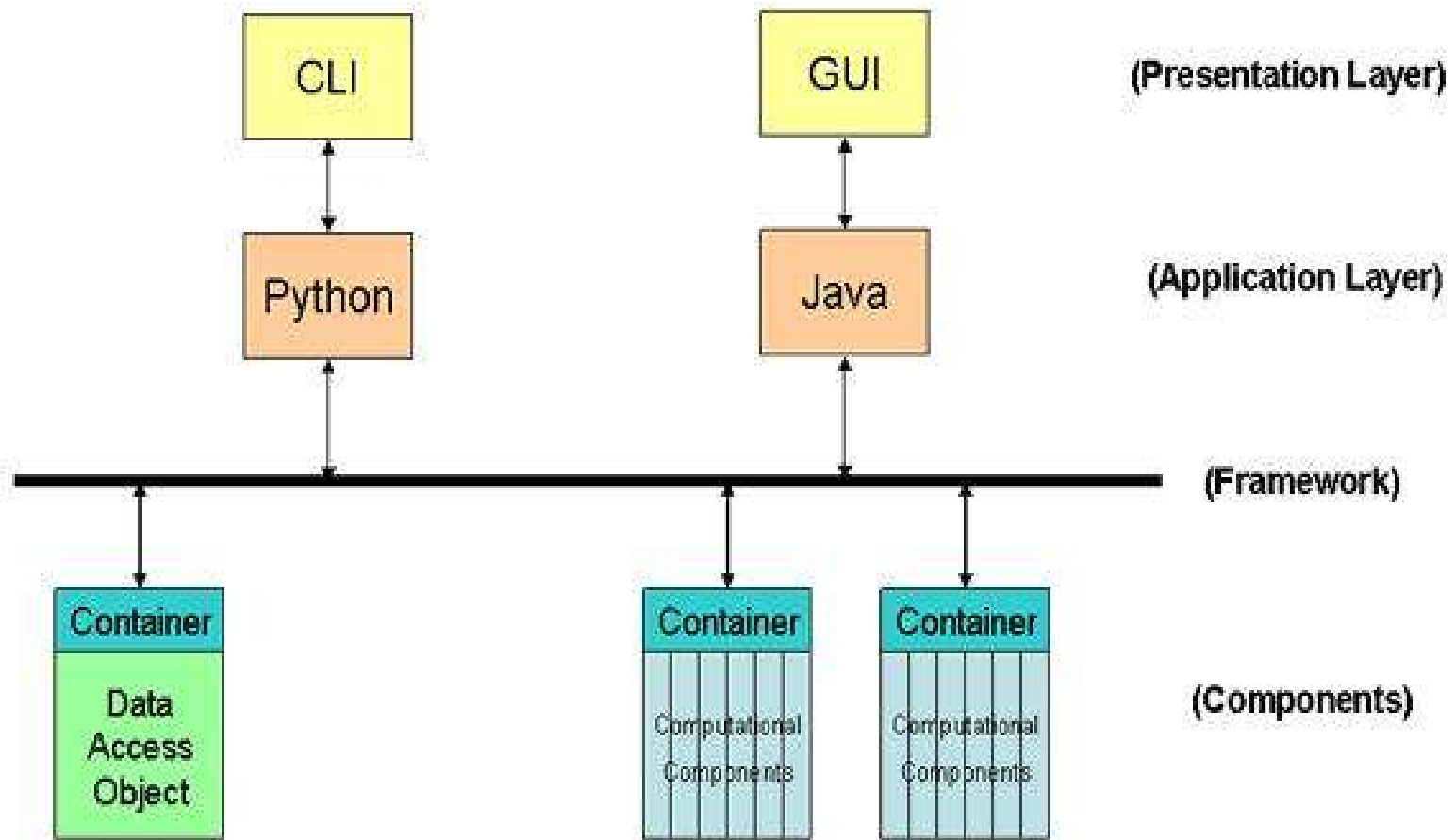
- software bus offers e.g. messaging and parameter passing

✓ Component-Container

- containers provide standard interface to Framework layer

- components implement of computational tasks e.g. in C, FORTRAN

Architecture - Component Framework

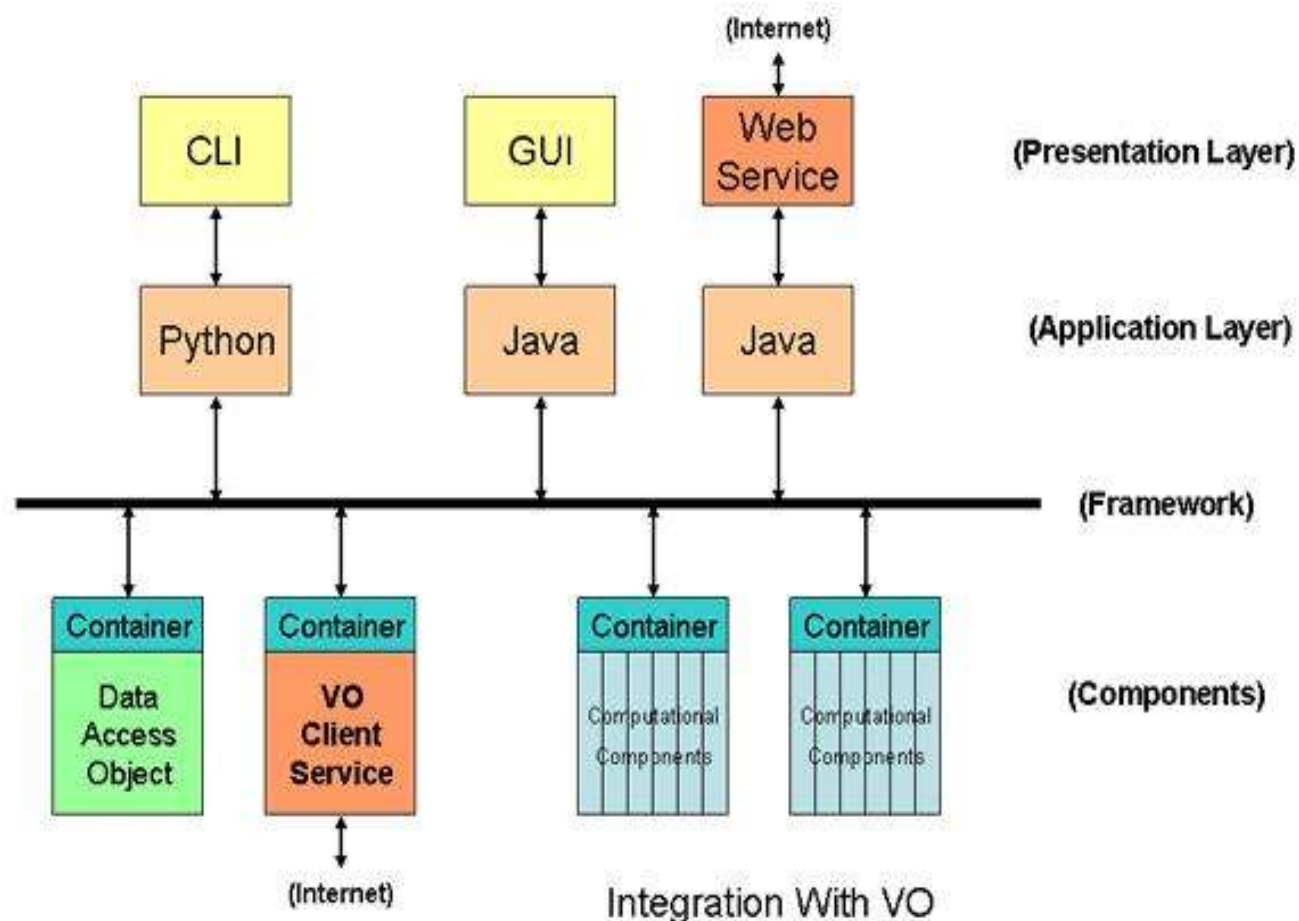


Component-Framework Architecture

Architecture - Interface to VO

→ Interface to VO world

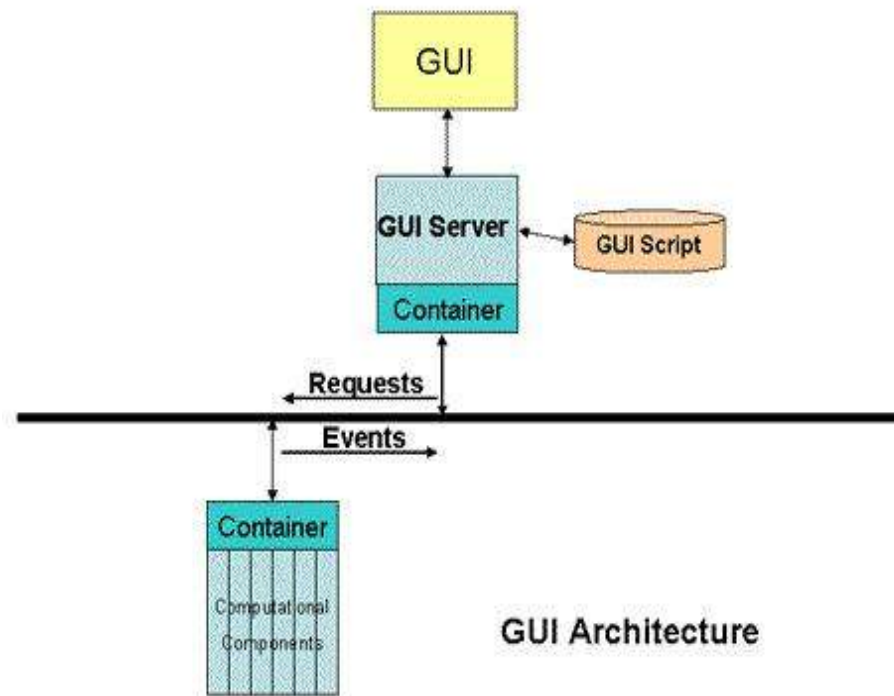
- ✓ **usage** of VO Services through a component
- ✓ **implementation** of Web Services for VO



Architecture - Integration of GUI's

→ Graphical User Interfaces

- ✓ normal GUI's implemented at **presentation** layer
- ✓ for **highly interactive** GUI's either through:
 - **GUI Server** using framework messaging for communication
 - **sharing container** with the computational component



What's next?

- Plans for next 18 month period
 - ✓ Consolidate high-level requirements
 - add one level of more detailed requirements
 - ✓ Agree on architectural concept
 - general comments on concept by end January 2005
 - ✓ Start high-level design
 - identify major parts or subsystems
 - list required interface specifications
 - outline minimum container with associated interface
 - detail model for parameter passing
 - ✓ List important technologies to be prototyped
 - ✓ Coordinate groups which intend to prototype parts of system
 - propose suitable demonstrations for such prototypes
 - ✓ Face-to-face meetings (tentative)
 - June 2005 ESO, October 2005 Madrid (before ADASS)