Fostering the adoption of UNICORE Portal

Krzysztof Benedyczak, Piotr Bała, Marcelina Borcz, Valentina Huber, Rafał Kluszczyński, Mariya Petrova, Bernd Schuller, Piotr Piernik

ICM, Warsaw University
FZJ

Outline

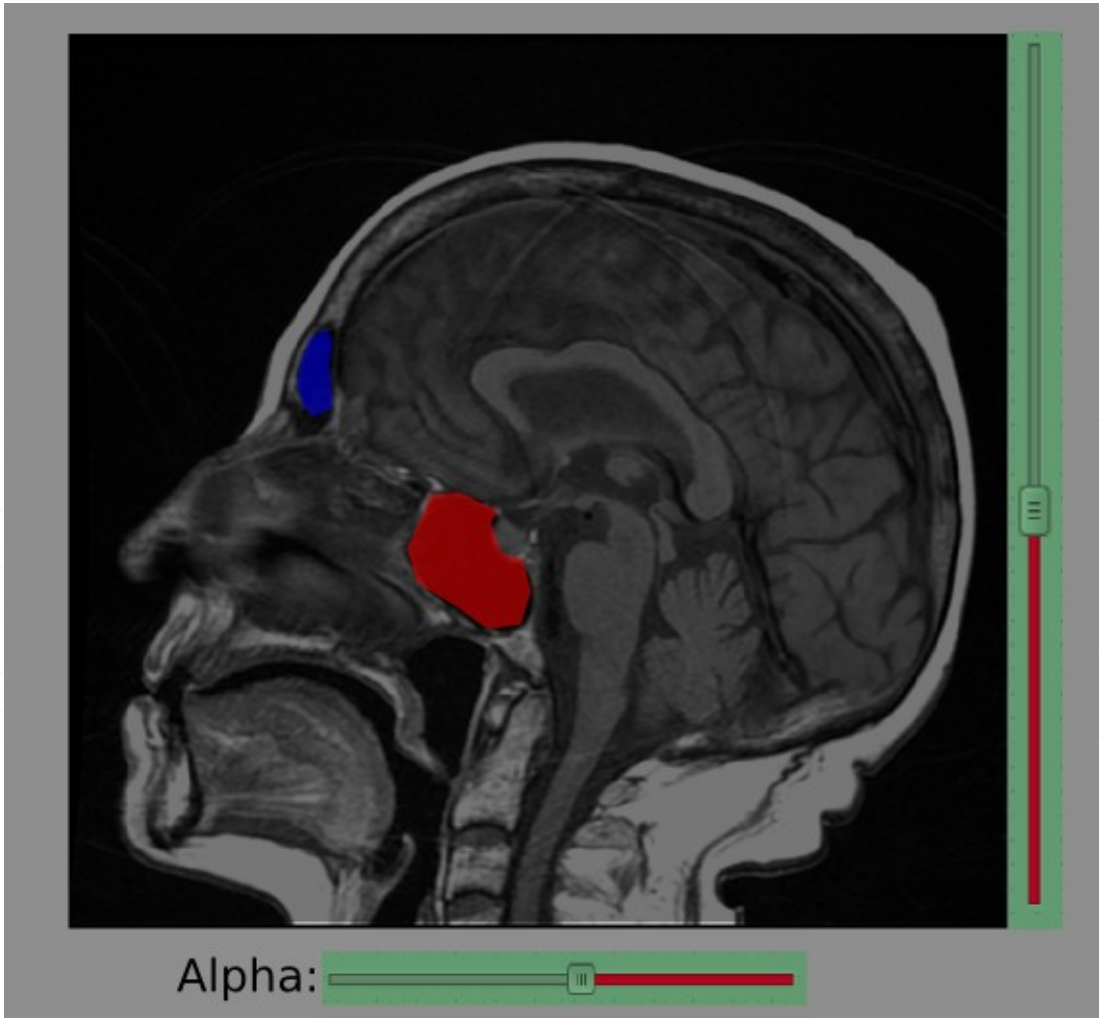
- Portal use cases
- Past experience: GridBeans
- Assumptions
- Current state of the portal
- Proposed roadmap

Approach to the portal

- UNICORE Portal can be considered as an easier to use URC replacement
  - No need to install or update
  - Can be preconfigured by its administrator for a concrete Grid infrastructure
- In this talk we look at the Portal also from a domain perspective
  - *Easy button* (tm) approach
  - Dedicated to a well defined group of users with a concrete, not generic requirements.
  - Sometimes sophisticated features needed.
- We need to support both worlds.

Use cases: SinusMed

- Image analysis of series of CT images of patient's head.
- Application recognizes and marks air-filled areas (sinuses) in the whole series allowing for obtaining 3D image.
- Useful for further processing: measuring air volume, air flow etc.



Use cases: SinusMed

- A single, *atomic* application.
  - Rather big input and output (couple of hundreds of Mbs)
- Requires:
  - Output visualization, including the output of early stages of processing, to recognize malformed input parameters.
  - Intuitive management of previous simulations, input and output sets.
  - Very simple management of resource requirements.
    - Actually should be fully automated: *the fastest track to results*.
- Future: part of multistep processing (not a workflow!)
- Very good example of a simple application that should be done right.

Use cases: fighting cancer with genomic research

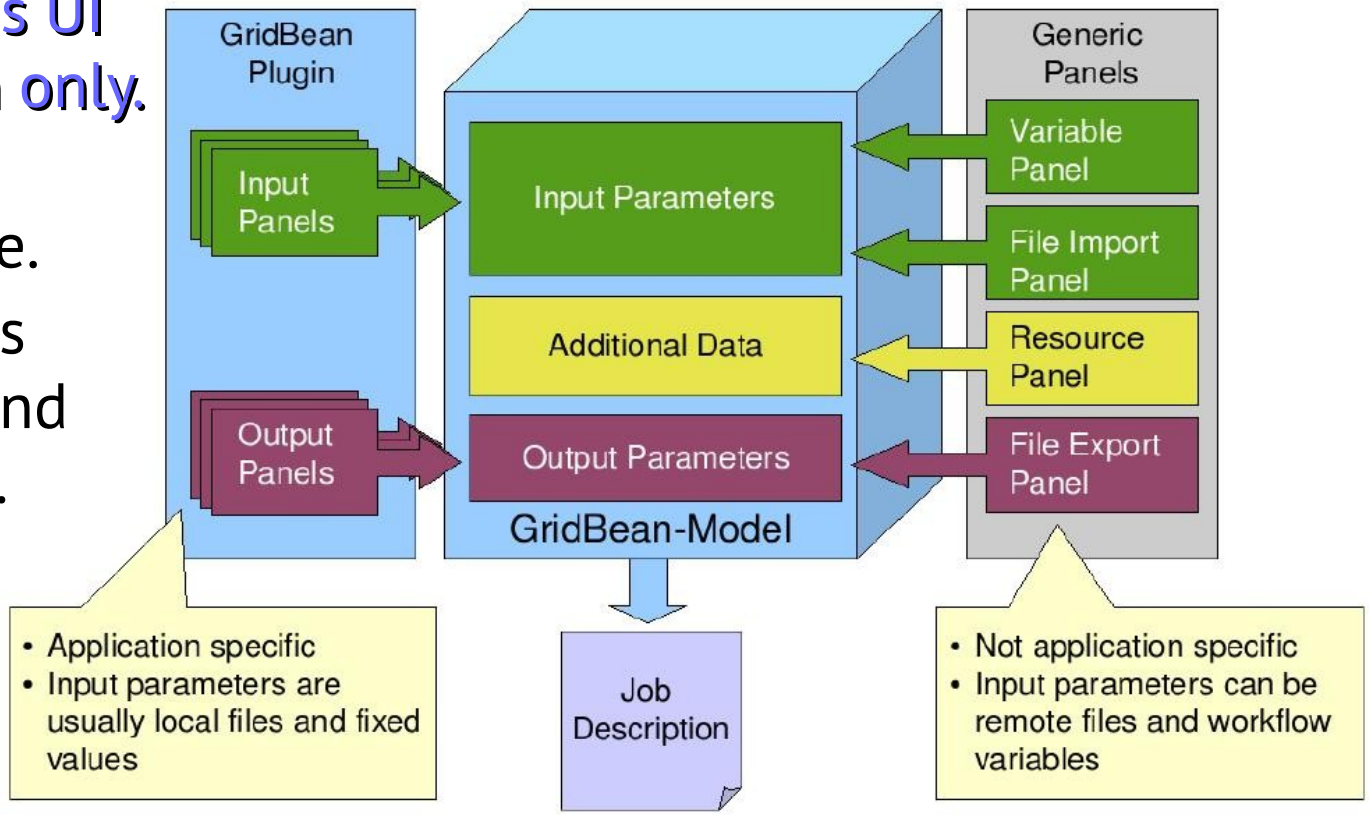
- Determination of differences between tumor and non-tumor genome sequences of tissues obtained from patients diagnosed with colorectal cancer.
- Internally: execution of a complex workflow, with structure dependent on particular simulation requirements.
- Requirements:
  - API-based preparation of a workflow and its submission.
  - Simulations management: coherent view, rebuild of input of submitted simulations. Simulation = complete workflow.
  - Simple control of selected resource requirements.
  - Sneak peek of output being generated.

Use cases: VASP

- The Vienna Ab initio Simulation Package (VASP)
  - Used for atomic scale materials modeling.
  - Computes an approximate solution to the many-body Schrödinger equation.
- GridBean-like use case:
  - Input preparation (simple),
  - Output visualization (Jmol-like).
  - Coherent presentation of all submitted simulations.
  - Automatic submission via broker.
  - Simple control of selected resource requirements.

Learning from the past: GridBeans

- GridBenas model was introduced in Grid Programming Environment, at the beginning of SOA as a universal application integration layer.
- Supports mostly atomic jobs.
- Developer programs UI and job description only.
- Fixed (prepare, run, see results) lifecycle.
- Framework provides resource, variable and files control panels.



By Sandra Bergmann (?) from GB developer guide

What was wrong with GBs?

- Exchangeable UIs (Swing or SWT or...) didn't work.
- Too complicated (extra layer) for simple applications.
- Generic in theory while UNICORE specific in practice.
- What counts:

CLOSED FRAMEWORK

  - *How to make a workflow job?*
  - *How to organize simulations in a customized way?*
  - *How to provide a simpler implementation of resource/file/variables control?*
  - *Change the overall app UI?*
  - *Interact with a job at its runtime?*

Design assumptions

- KISS & YAGNI
  - We are a small developer group, we can't afford overengineered code.
- Flexibility
  - We shouldn't produce a closed API as we can't foresee all the use cases.
  - Instead an open API is needed:

You can do whatever you want, but certain things are easier with our API

Current status of the portal

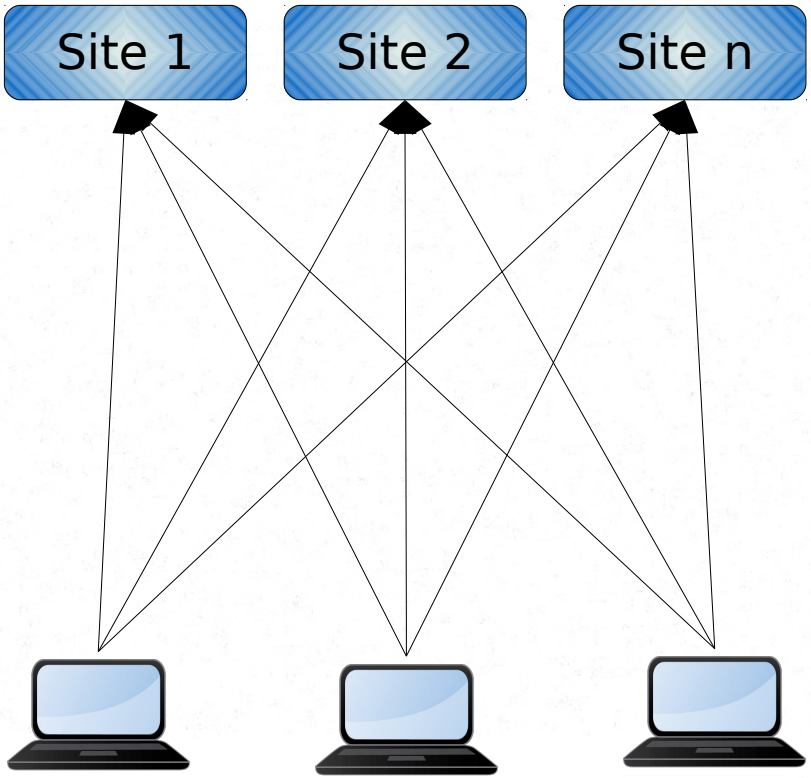
- Custom solution for shared objects – registry type.
  - Controlled in XML, tightly coupled with UI assembly.
- Grid model is imported from URC code
  - Hierarchical *nodes* structure, high use of inheritance and events.
  - Part of code not used (import).
  - Files access abstracted via Apache VFS, per user.
  - Grid status is polled.
- Couple of UI components:
  - Grid (tree), Sites (table), Jobs (table), Data (file browser).
  - Generic job component (similar to Generic GridBean).
- No portal API.

Portal code stats

Module	NCSS	% of total code	
Applet integration	1316	3%	
Authentication	2382	6%	
UI	9091	23%	
Workflow	10809	27%	
Core	16347	41%	In this nodes: 13%
TOTAL	**39945**		

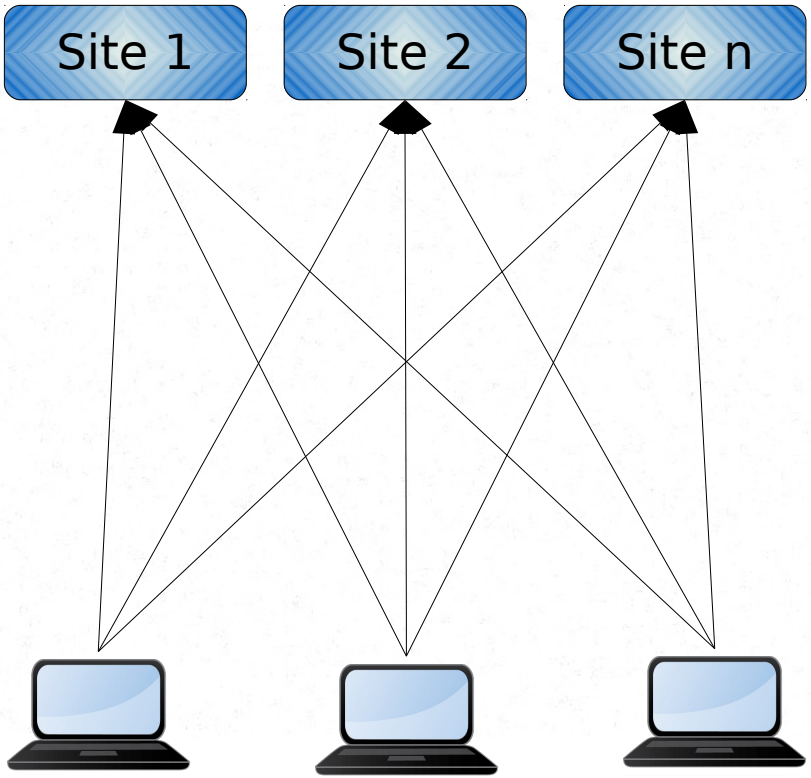
Communication flow

URC case

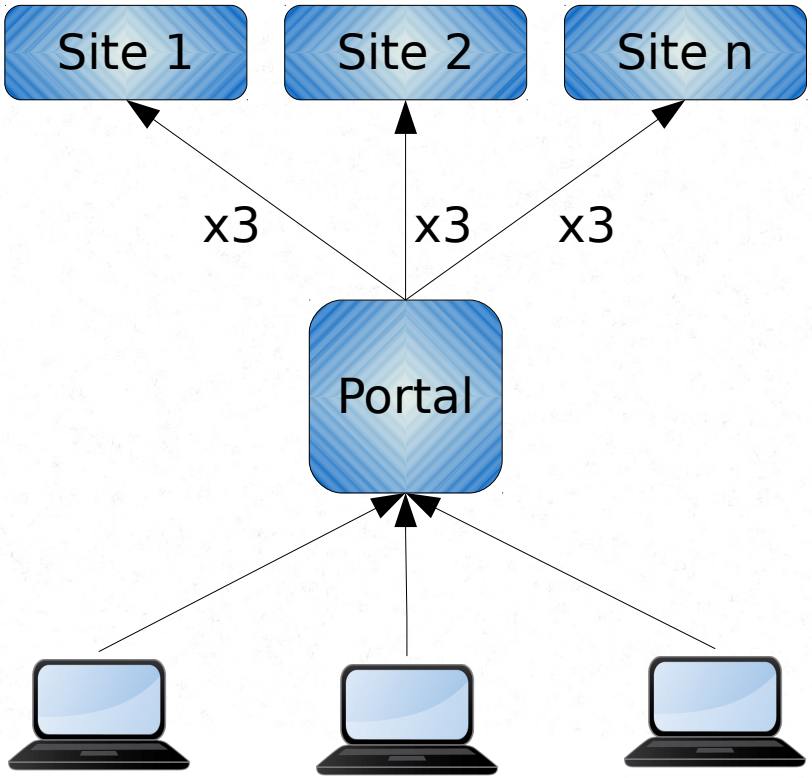


Communication flow

URC case

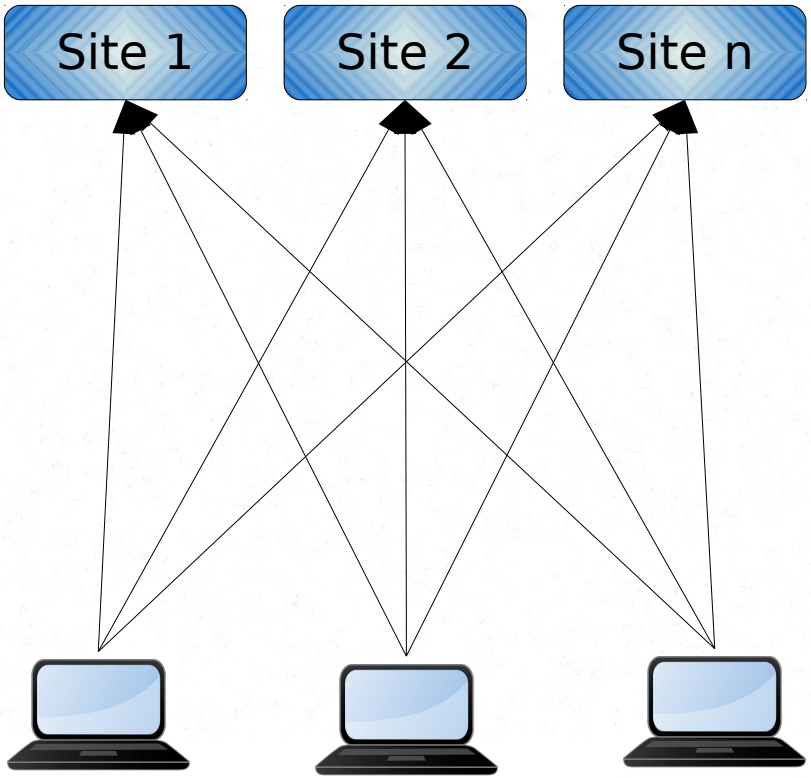


Portal case

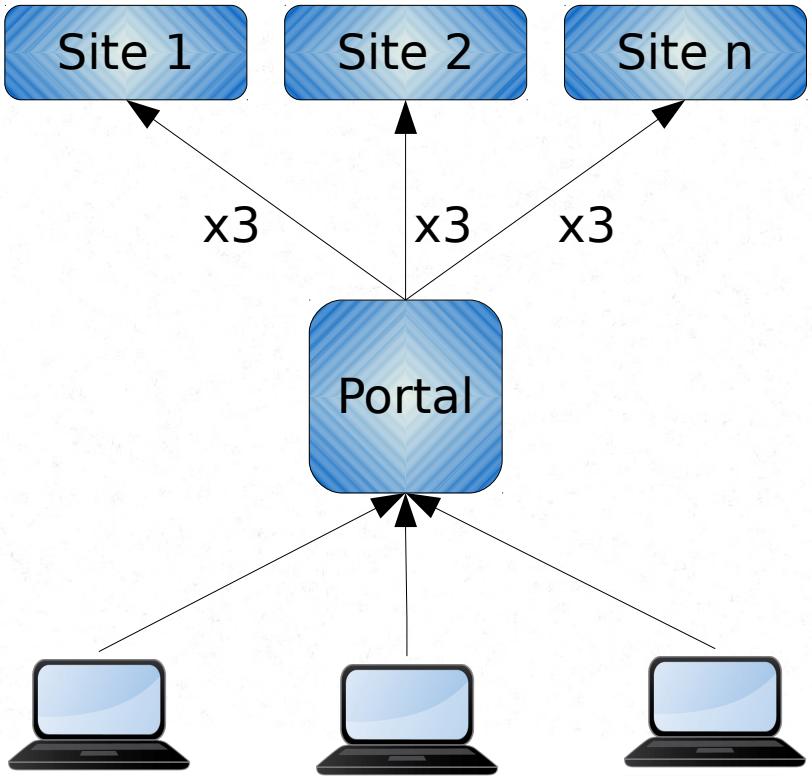


Communication flow

URC case



Portal case



10 users x 5 sites x 20jobs = 1000 x getProperties / minute (or more)

Proposed portal roadmap

Foundation

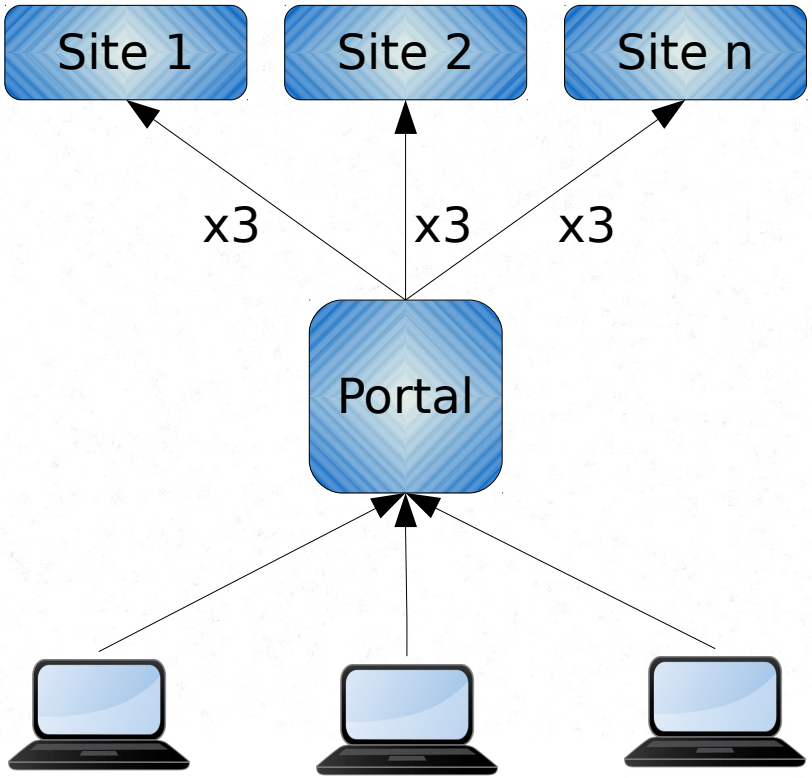
- Use standard solutions as architecture foundation: IoC (e.g. Spring) instead of custom code
  - Less code to maintain, less verbose code, app singletons dependencies, dependency cycles control, more assembly features.
- Decouple UI assembly from dependency and singletons.

The state

- Refactor grid state model so that:
  - It implements what is needed (cleanup of URC specific code)
  - The Grid topology view can be build for the portal once, not per user.
    - Faster, always available, less resource usage.
  - The cache is reliable, fast.
  - Internal events system is simpler (current is terribly heavy weight)

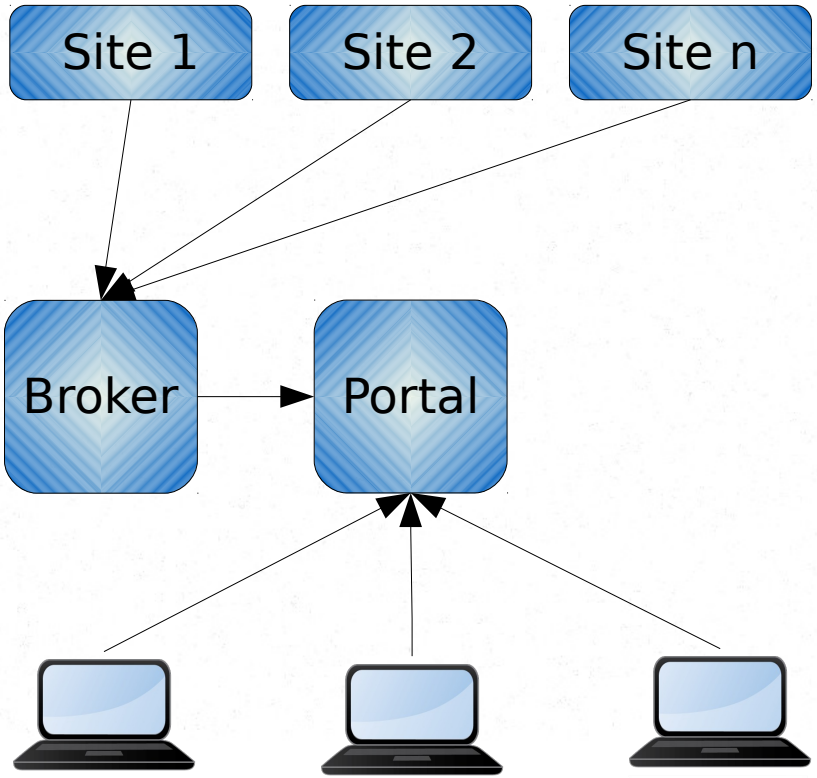
Do not poll

Current polling model

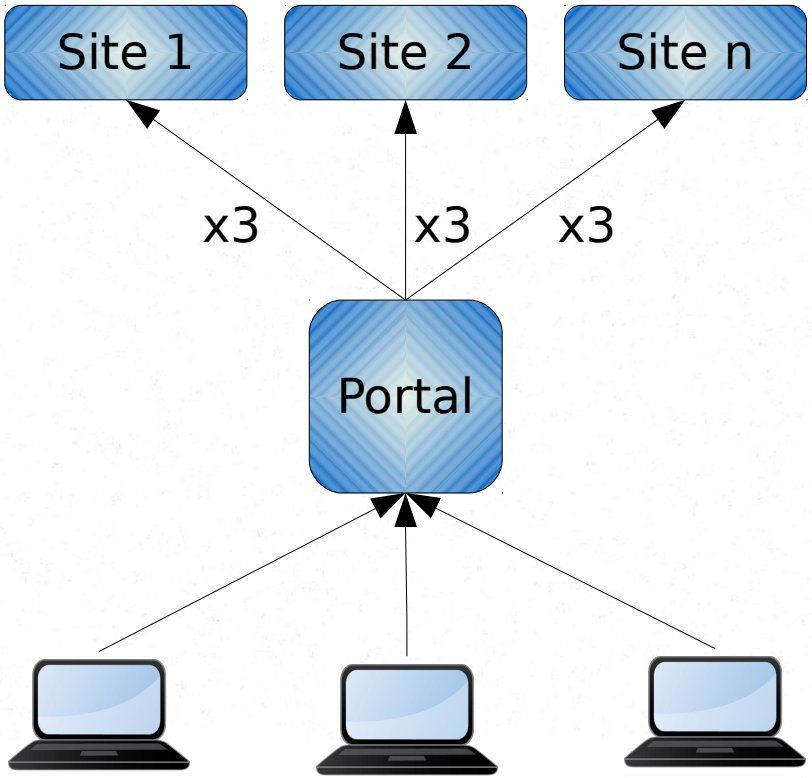


Do not poll

Proposed events model



Current polling model



Requires support at server side!

Reusable UI components

- Single root package
- Fully API configurable
- Flexible

- Jobs Table Viewer
  - Selectable columns, filterable contents (by application, by tag), no need for manual refresh.
  - Support for worflows – can be another component (Tree table?)
- Resources selection components
  - Simple one with label like presentation
  - Control on resources being shown

Reusable UI components (2)

- File monitoring component
  - Ability to provide custom handler.
- File imports and exports component.
- Variables component.
- Upload to the Grid component.
- Download from the Grid component.
- Data Manager can be useful but is very complicated (too many storages).
- The need for the Grid Browser and Sites Browser is minimal.
  - Grid admins only?

High level API

- Possibility to easily perform common tasks:
  - Discover Grid state, jobs
  - Get notifications about updates
  - NOT Grid browser oriented. E.g. getAllJobs, instead of get all job-type children of an enumeration node...
- Gridlet API can be used as a base.

The last mile

- For typical applications a GridBean-like framework can be provided.
- 10x simpler:
  - Generic UI, where app integrator can select with few lines of code which modules are needed (submit button, resources panel and file imports)
  - Should provide common look and fill for apps in the portal and promote good UI practices.
- The only goal should be: make simple app integration easier. No more, no less.