CRIC

Technical view on the system

Alexey Anisenkov (BINP)

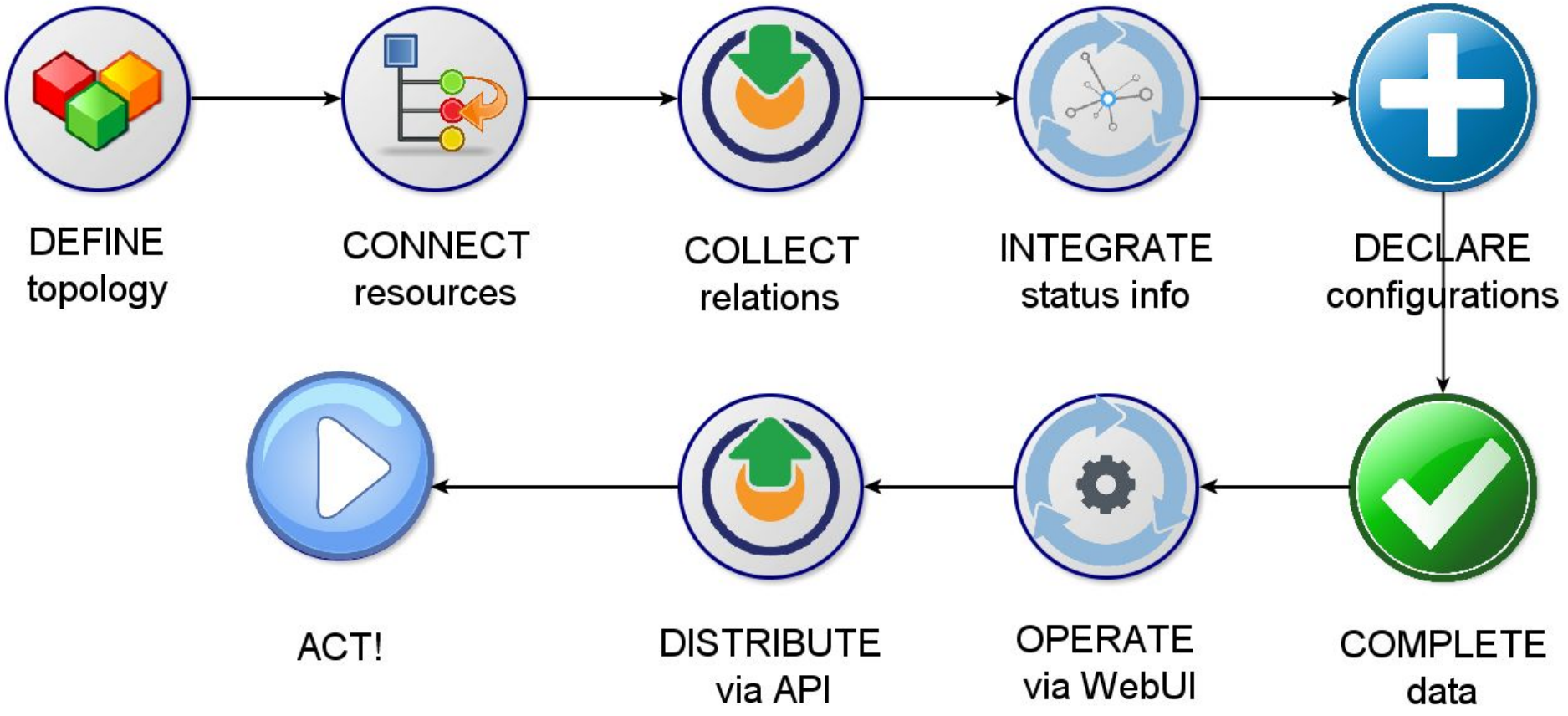
CRIC (Computing Resource Information Catalogue)
The Concept

- CRIC is an Information system
  - CRIC is NOT a monitoring system
  - is NOT a Content Management system (e.g wiki based)
  - is NOT a static Web Application exposed something
  - is NOT a unified Configuration system able to define object structures on fly from WebUI

  CRIC is an Information Model based Web Application

- CRIC implements specific Information Model (Computing Model) and provides functionality to operates with it
- Mainly, CRIC could be considered as the framework to implement custom Information model
- The information model behind covers well (at least for ATLAS today) the definition of Computing Resources and the topology of Distributed Computing in general
- Once we need to extend the functionality of CRIC, we need to check if it affects the information model and do upgrade it first

Key capabilities of the system

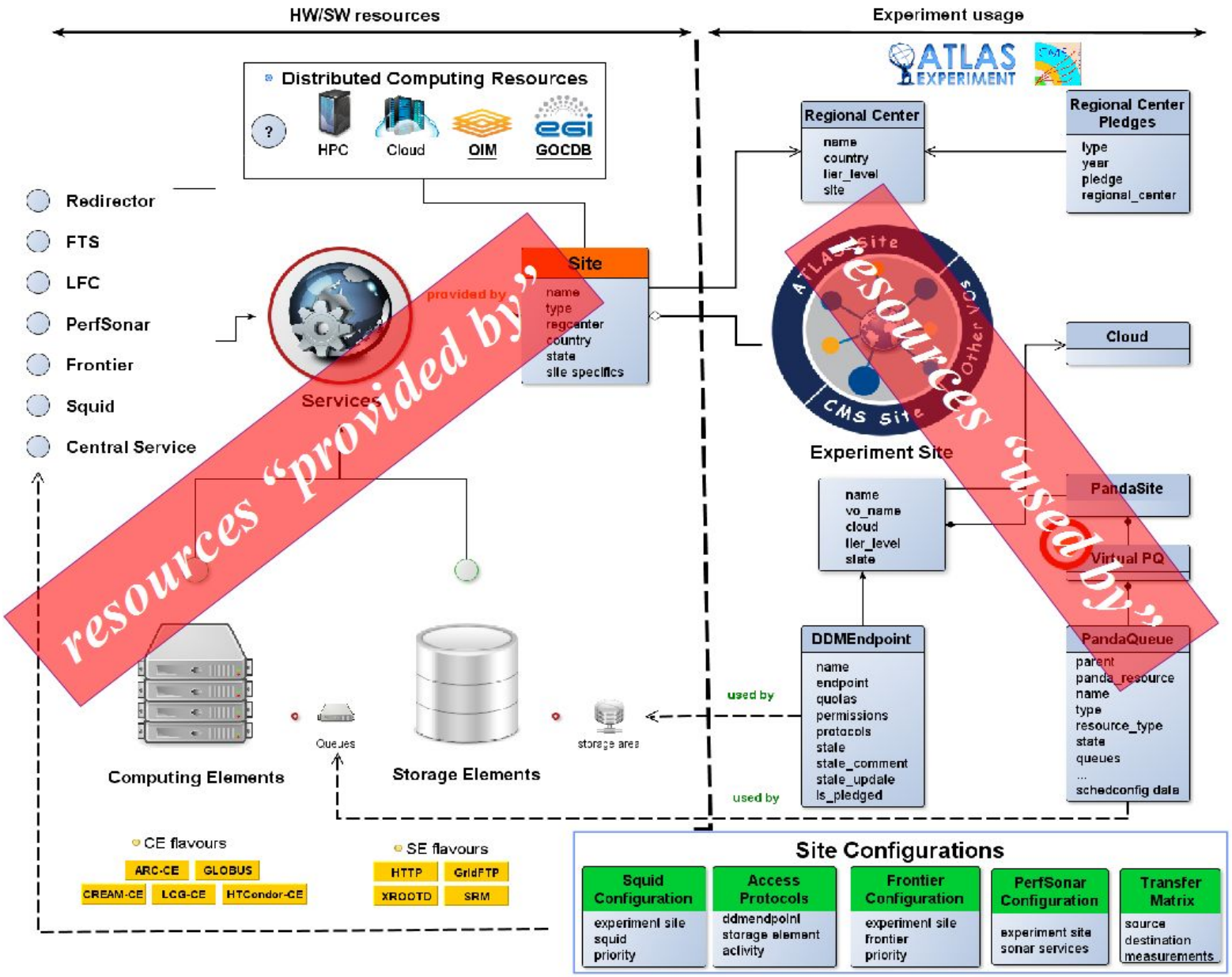


Fundamental concept of the system including Experiment specific declaration

- Clear distinction between resources
  provided by (Sites) and resources *used by* (Experiments)
- Establish relationship between resources to Experiment objects

Providing an abstraction layer from the physical resources the system allows the Experiment to define their own real organization of the resources, experiment specific topology and own services structures.

Fundamental concept of the system including Experiment specific declaration



Architecture of the system

http://cric-dev.cern.ch

- CRIC is based on client-server architecture and offers 2 independent services:
  - ➢ API service (REST-full GET export: JSON, etc; POST update)
    -- mainly used to export data, bulk updates and operated data programmatically
  - ➢ WebUI portal (AJAX support, JQuery widgets, etc)
    -- mainly used to navigate, browse and declare objects

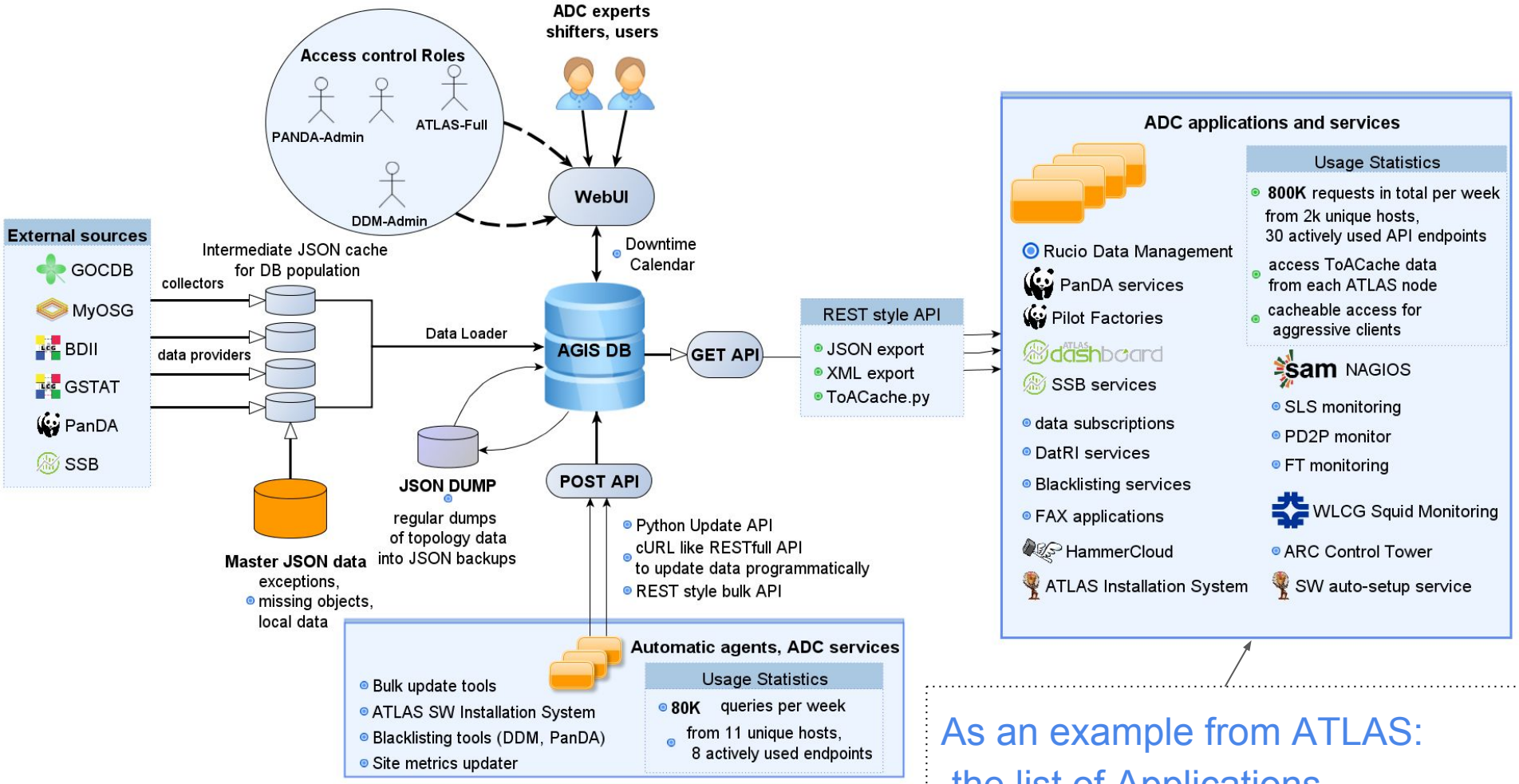
  The services could be hosted on different nodes, LB supports, etc
  The sevices for sure use same persistent DB instance

- Various Integrated collectors run by crons to populate DB from external sources
- Internal cron jobs to maintain the system (e.g. clean up db cron)

- The system supports information protection: Authorization is required to modify the data: Groups, Roles and list of specific permissions could be directly associated to user (At the moment authorization through SSL certificates is applied)

Architecture of the system

- General overview of the system services



As an example from ATLAS:
the list of Applications
(customers) using AGIS for today

Implementation details

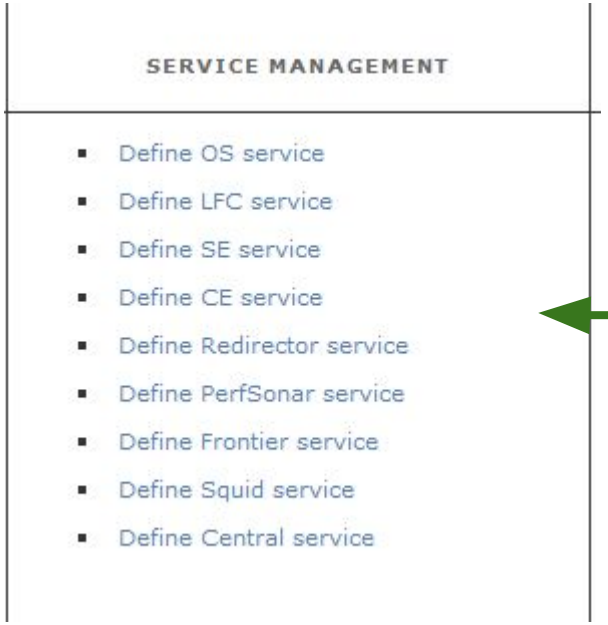
- The system is built on Django:
  a powerful and flexible framework for the development of custom Web application written in Python
- Module based implementation:
  Scalable Django approach makes logic isolation into different applications very effectively
  - plugin based approach
  - Shareable applications that could be just reused in separate projects
    So that we can make common “core” and attach various plugins into it
- MVT based django implementation (Model-View-Controller)
  - **Model**: database layer mapped to python objects (using built-in Django ORM - Object Relational Mapping) -- no direct dependency to specific database backed
  - **View**: user interface layer splitted into business logic code (called views in Django) and HTML page templates: views render html templates
  - **Controller**: a set of custom middlewares and django cores enabled the processing of user request and finally evaluating attached view function to handle user response
- High level design and logic separation benefits:

Implementation details (2)

- Core functionalities already implemented (and used in AGIS)
  - Base form processing and data validations
    - Simple forms
    - Forms with changes confirmations
    - Example forms
    - Support of separated processing of non models fields
    - Auto-completion fields, info fields, messages..
    - Etc..
  - Generic data views
    - Own implementation of (downtime) Calendar integrated into WebUI
    - Interactive sortable table views (Jquery datatable based)
    - Tree based views
    - Built in JSON exports in WebUI (to fetch data client side on WebUI)

An example of WebUI: Service creation

- The system automatically collects CE (SRM, PerfSonar, etc) services from GOCDB/OIM
- What should be done if expected service (for whatever reasons) not defined in the system
  - ➢ contact atlas-adc-agis@cern.ch in case you consider this as a bug
  - ➢ otherwise just inject missing service manually into the system using WebUI



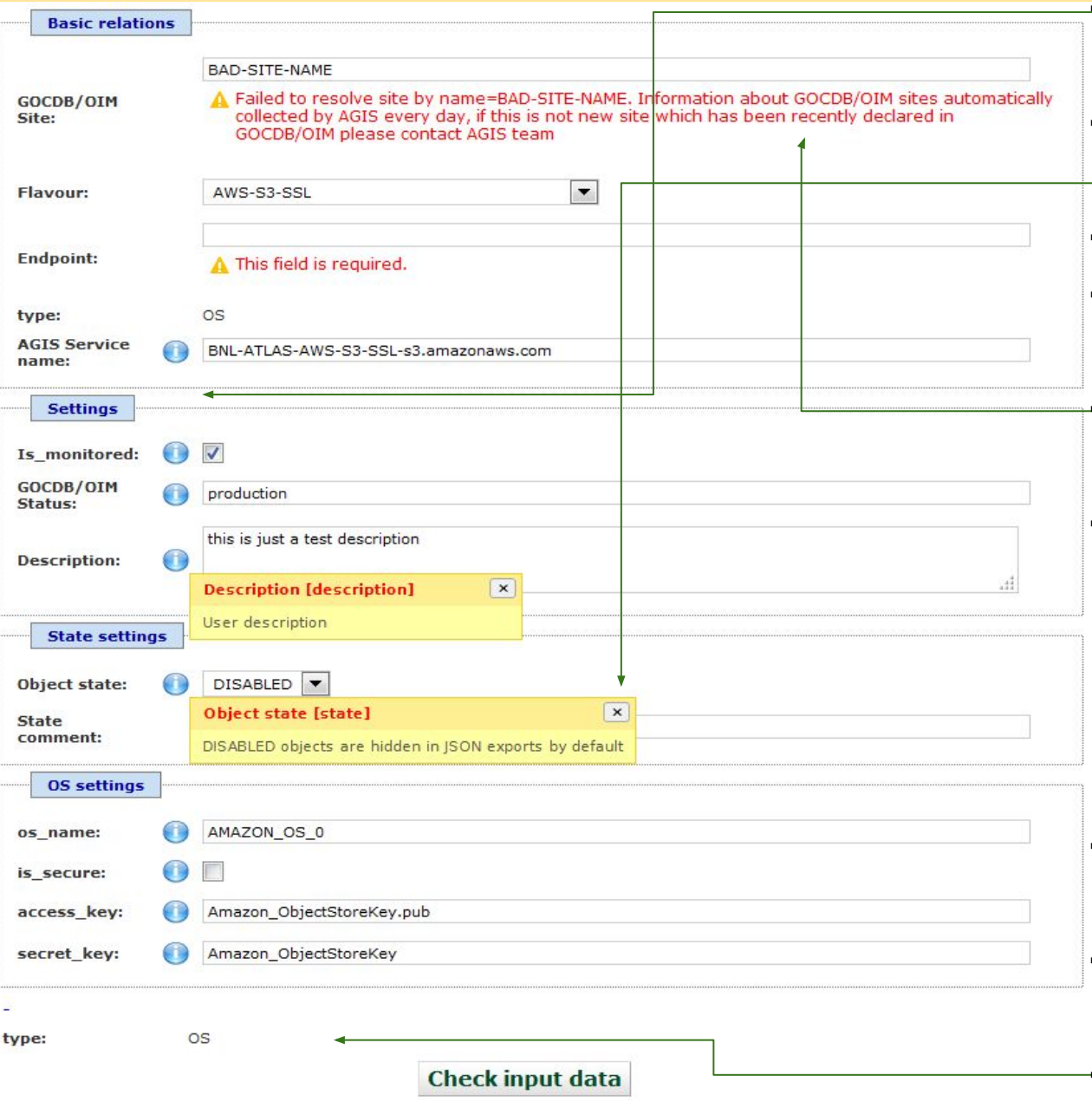
1. Main AGIS page contains the links for manual service declaration

2. next slide: example of CE edit/creation form (new style implementation)

We continuously improving AGIS WebUI,

providing more useful views and forms with incorporated validation and data checks support before injecting it into DB

An Example of Basic Form

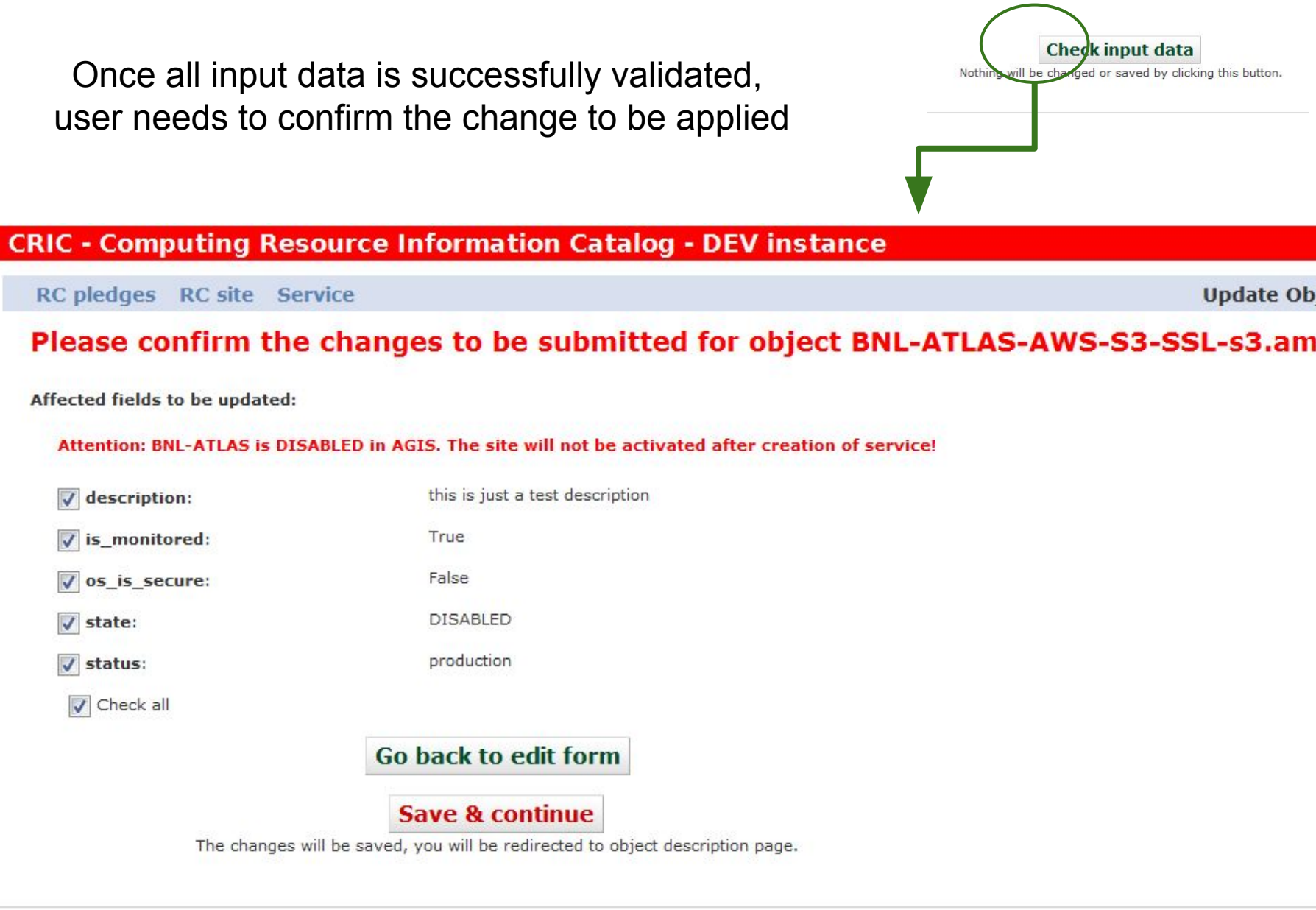


General Features:

- Fields grouped by their meaning into fieldset
- integrated popup tooltips and help messages
- Custom validation logic
- Pre-validation applied before saving into DB
- Highlight errors in case of invalid input
- auto-generated/default values for attributes whenever it possible (e.g if internal service name leave empty, the form will generate appropriate value)
- drop-down menu with possible choices for field
- automatic suggestion while taping
- Read only fields

An example of changes confirmation of form before final commit

Once all input data is successfully validated, user needs to confirm the change to be applied



An example of Authorization: How to get access to the system

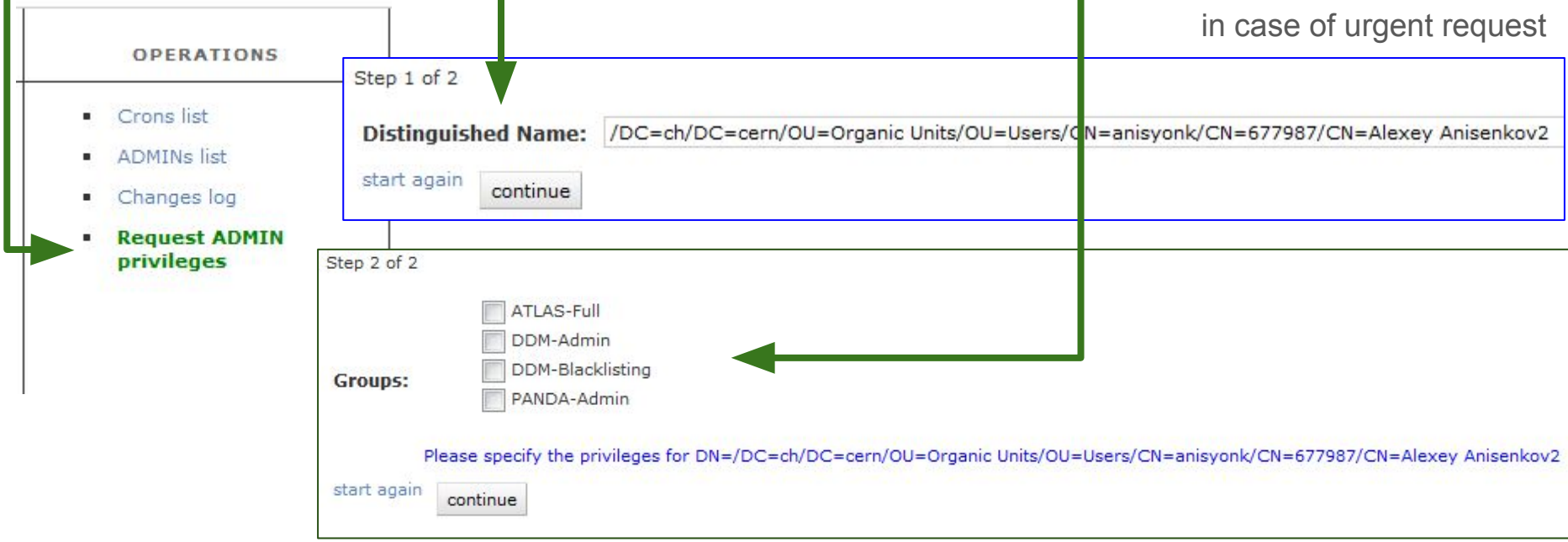
- Your DN has been changed?
- First time using system to modify site specifics (PandaQueues, DDMEndpoints, ATLAS Sites, services, etc)
- Need to ask additional admin privileges to operate with AGIS?

1. Request ADMIN permission from the main page: http://cric-dev.cern.ch

2. Use SSL certificate or input custom DN name to be registered

3. Select required permissions

4. Notify atlas-adc-agis@cern.ch in case of urgent request



Conclusion… (or start?)

- AGIS -> CRIC implements Computing Information model providing fruitful functionality to operate with data through API and WebUI services
- To integrate CMS specific the Information model is need to extended and completed with CMS use-cases
- Once CMS use-cases are identified, CMS specific implementation need to be applied to CRIC (Experiment Part - “used by” concept): shared functionality from the core can be reused as well
- ATLAS is continuously refactoring AGIS and move into the CORE the functionality non specific to ATLAS so that it could be used as from the box by CMS
- Plugin based approach makes the customization as much as possible, but still the core is need to be maintained.