Software Engineering for Distributed Data Processing Infrastructures (DCIs)

Etienne URBAH  urbah@lal.in2p3.fr
Oleg LODYGENSKY  lodygens@lal.in2p3.fr
LAL, Univ Paris-Sud, IN2P3/CNRS, Orsay, France
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

Summary

1) Distributed Data Processing

2) Best practices – Software engineering

3) DCI = Federation of Administrative Domains

4) Middleware Stacks for Distributed Data Processing

5) Standards for Distributed Data Processing
1) Distributed Data Processing
1.1) Scope of ‘Data Processing’

Data processing is much more than computing:
1.1) Scope of ‘Data Processing’

Data processing is much more than computing:

- Data acquisition
- Instrument management
- Data, Metadata management
- Access rights to data
- Computing
- Filtering, Sorting
- Graphical display
- Long term curation, ...
1.2) Data Processing on a User Device

On any single user device (PC, Mobile phone, ...) :

- Access rights are very simple,
- But storage and computing power are limited.
1.3) Data Processing needs to be distributed

Depending on the workflow of data processing, infrastructure types are more or less appropriate, with a tradeoff between development effort, performance and cost:

Costly supercomputers are mandatory for workflows requiring huge core memory and/or highly coupled parallelism. Development effort is high for efficient highly coupled parallelism.

DCIs such as EGI and OSG are optimal for secure data sharing, and for workflows using large data sets with medium core memory and small parallelism. Development effort is low for single applications.

Desktop grids are optimal for workflows requiring heavy uncoupled parallelism, moderate core memory and moderate data sets (< 100 MB). In the near future, desktop grids will also permit secure storage of data sets of any size, but transfer rate will stay limited. Development effort for single applications highly depends on the grid middleware (high for BOINC, low for XtremWeb).
2) Best practices – Software engineering

• WHO ?
• WHAT ?
• WHEN ?
• WHERE ?
• HOW ?
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

2.1) Bad practices to avoid

In order to harness the complexity of distributed data processing:

**Beware of buzzwords**

Passive sentences should also be avoided:
Each of us is responsible for avoiding passive sentences.
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

2.2) Best practices – Software engineering

In order to harness the complexity of distributed data processing:

Best practices and Software engineering already exist:

- **UML**

- **CMMI-DEV**
  [http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm](http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm)

- **ITIL**
  [http://www.itil-officialsite.com/AboutITIL/WhatisITIL.aspx](http://www.itil-officialsite.com/AboutITIL/WhatisITIL.aspx)

etc ...
2.3) Best practices – Simple questions

In order to *harness the complexity* of *distributed data processing*: Begin by asking very *simple questions*:

**WHO ?** (Who are the actors, stakeholders, providers, beneficiaries, deciders, ... ?)

**WHAT ?** (What is the question, the subject, the topic, the issue, the problem, ... ?)

**WHEN ?** (Is it in the past, the present, the future? May it happen several times? What is the global time frame? Is it transient or persistent? Are there interaction sequences? ...)

**WHERE ?** (Is it local, regional, national, world-wide? Does it stay at the same place, or may it move? Is it localized, or distributed? Are there distance issues? ...)

**HOW ?** (What are the interactions between the members of WHO and WHAT? Which event triggers which interaction? Do these interactions occur in a precise order (WHEN ?), at locations (WHERE ?), Who is responsible for...)

Authors: E. Urbah, O. Lodygensky
2.4) Software engineering – UML

In order to harness the complexity of distributed data processing:

UML permits to create meaningful diagrams, which software engineers can understand, criticize and improve:

- **Use Case** Diagrams: *Who is responsible for what?*
- **Class** Diagrams: *Which relationships between which entities?*
- **Collaboration** Diagrams: *Which entities have which interactions?*
- **Sequence** Diagrams: *Which messages in which order between which entities?*
- **State** Diagrams: *Which transitions between which states?*
- **Deployment** Diagrams: *Which components on which machines?*
- **Activity** Diagrams: *Which logic flow for which process?
2.5) Software engineering – First steps

Using **Best practices** and **Software engineering** to harness the complexity of **distributed data processing**:

- Clearly **distinguish and describe** relationships between **concepts**

- An extensive **glossary** is available at [http://www.ogf.org/documents/GFD.181.pdf](http://www.ogf.org/documents/GFD.181.pdf)


- **Robustness** is an intrinsic property of a resource.

- **Virtualization** permits flexible on demand management of underlying resources, but only if these underlying resources are already robust, and at the cost of the virtualization layer, which may take months to acquire robustness.

- **Administrative domains** contain a repository permitting easy authentication and authorization of users.

- **Grid computing** is a federation of administrative domains, based on mutual trust, some common operational setup (in particular the trust anchor for authentication and authorization), specialized interoperable middleware stacks, and permit secure data sharing.

- **Cloud Computing** is the on demand management (allocation, instantiation and usage) of internal or external computing resources inside a given administrative domain; but it does currently not span across multiple administrative domains.

- **Interoperability** between clouds does not exist yet, as it requires some common cloud operational setup and interoperable cloud middleware stacks. Work is going on federation of clouds, for instance in [http://contrail-project.eu](http://contrail-project.eu)
2.6) Software engineering – Users

In order to assess User Needs, take care to track all Users:

Users having ICT knowledge are able to express, criticize and validate sound needs and requirements.

**End Users** of DCIs are scientists, with various ICT, Grid and Cloud knowledge. For example:
- Application developers,
- Experienced application users,
- Scientists with no ICT knowledge using a scientific portal,
- ...

**Direct Users** of DCIs are various:
- Developers of scientific applications,
- Integrators of scientific applications for grids,
- Providers of scientific workflow engines,
- Providers of scientific portals,
- Providers of SAGA,
- Site Administrators,
- VO Administrators, ...

Authors: E. Urbah, O. Lodygensky
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

3) DCI = Federation of Administrative Domains
3.1) Administrative Domain – Principle

- Each organization (University, Research Institute, Administration, Enterprise, ...) has in general an Administrative Domain managed by an operational team and protected by a firewall.
3.1) Administrative Domain – Principle

- Each organization (University, Research Institute, Administration, Enterprise, ...) has in general an Administrative Domain managed by an operational team and protected by a firewall.

- This Administrative Domain permits that each registered user can use any shared resource of the Administrative Domain following defined access rights.
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

3.2) Payload and Support Functionalities

The end user is primarily interested in 'payload' functionalities:

- Data Acquisition
- Instrument Management
- Data, Metadata Management
- Computing
- ...
3.2) Payload and Support Functionalities

The end user is primarily interested in 'payload' functionalities:
- Data Acquisition
- Instrument Management
- Data, Metadata Management
- Computing
- ...

But management of the shared resources of an Administrative Domain also requires 'support' functionalities:
- Security
- Information (Publication of resource capabilities so that they can be dynamically discovered)
- Monitoring
- Logging and Bookkeeping
- Accounting
- ...
3.2) Payload and Support Functionalities

The end user is primarily interested in 'payload' functionalities:

- Data Acquisition
- Instrument Management
- Data, Metadata Management
- Computing
- ...

But management of the shared resources of an Administrative Domain also requires 'support' functionalities:

- Security
- Information (Publication of resource capabilities so that they can be dynamically discovered)
- Monitoring
- Logging and Bookkeeping
- Accounting
- ...

These functionalities are provided more or less completely and consistently by access and sharing middleware.
3.3) Administrative Domain:
Access and sharing methods

Inside an Administrative Domain, any access or sharing middleware may be used:

Firewall

Authors: E. Urbah, O. Lodygensky
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

3.3) Administrative Domain:
Access and sharing methods

Inside an Administrative Domain, any access or sharing middleware may be used:

- BitDew
- CORBA
- File server
- Database server
- Remote login (SSH, RDP)
- Batch system
- Applicative portal
- Private grid (Condor, BOINC, XtremWeb, ...)
- Private cloud
- ...

Authors: E. Urbah, O. Lodygensky
Inside an Administrative Domain, any access or sharing middleware may be used:

- BitDew
- CORBA
- File server
- Database server
- Remote login (SSH, RDP)
- Batch system
- Applicative portal
- Private grid (Condor, BOINC, XtremWeb, ...)
- Private cloud
- ...

Standardization is optional.
3.4) Clouds
3.4) Clouds provide on-demand resources

- A cloud provides on-demand resources with a rich and consistent set of data processing functionalities, and a powerful and consistent interface, but limited to ONE cloud provider (no interoperability).
- It permits flexible extension of an Administrative Domain.
- Cost of a commercial cloud is affordable for permanent data storage and computing, but very high for the transfer of large amount of data.
- Currently, clouds do NOT manage federation of Administrative Domains. Work is going on federation of clouds, for instance in http://contrail-project.eu
3.5) DCI = Federation of Administrative Domains

The goal is sharing of data.
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

3.5) DCI = Federation of Administrative Domains

The goal is sharing of data.
The principle is mutual trust.
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

3.5) DCI = Federation of Administrative Domains

The goal is sharing of data. 
The principle is mutual trust.

The method is agreement on common middleware stacks and operational procedures. Resources stay locally managed.

Authors: E. Urbah, O. Lodygensky
**Software Engineering for Distributed Data Processing Infrastructures (DCIs)**

**3.5) DCI = Federation of Administrative Domains**

---

**The goal is sharing of data.**

The principle is mutual trust.

---

The method is agreement on common middleware stacks and operational procedures. Resources stay locally managed.

The middleware provides consistent access rights to all users according to their DCI credentials.
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

3.5) DCI = Federation of Administrative Domains

- Some scientific, academic and research organizations, which already own data related resources, increasingly need to securely share these resources with others.
- In order to federate their private resources into a production infrastructure, these organizations have had to establish mutual trust, adopted compatible middleware stacks and procedures integrated through operations teams to bring their resources together into a Distributed Computing Infrastructure (DCI).
- The recurring feature of the various DCIs that offer production resources (e.g. EGI, DEISA, PRACE, etc.) is that each one integrates multiple locally managed administrative domains into a usable environment.
- The middleware deployed by each DCI provides its users, according to their DCI credentials, consistent access rights to all resources managed by that DCI.
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

4) Middleware Stacks for Distributed Data Processing

Authors: E. Urbah, O. Lodygensky
4.1) Various DCI middleware stacks

- Examples for DCI middleware stacks are:
  - eXtreme (EGI, EELA),  
  - ARC (NDGF),  
  - UNICORE (DEISA-PRACE),  
  - Globus (TeraGrid, NGS),  
  - XtremWeb (LAL),  
  - Naregi (RENKEI),  
  - Genesis (XCG), ...

  BOINC (many desktop grids) manages many resource donors, but only 1 user.

- Interoperation is provided by 3G Bridge + ad-hoc plugins.

- Interoperability between the various middleware stacks (which is not achieved yet) requires careful standardization. Used standards:
  - General ICT standards (IETF, Oasis, DMTF, SNIA, ...),
  - Standards specific for DCIs:


Remark: The GCM model of ETSI is not relevant for DCIs.
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

4.2) DCI middleware stacks – Poor interoperability

Each area in white is a grid infrastructure, now called DCI.

Interoperability, as shown in green and red, is desired.

But it is not achieved yet.

This diagram was created by Morris RIEDEL (Jülich Supercomputing Centre)
5) Standards for Distributed Data Processing
5.1) Identity of a User on a DCI

• Each DCI spreads across several Administrative Domains.
• So (login, password) does NOT work anymore.
• Each user must be identified by a certificate.
• Each middleware component using (login, password) must be adapted.
• In the USA, the InCommon federation provides SSL certificates.
• X509 certificates are the most robust:
  – Such certificates are provided to a user, after approval of his/her organization, by a Certificate Authority.
  – Each Administrative Domain must maintain a copy of the list of Certificate Authorities.
  – The list of CAs published by IGTF (International Grid Trust Federation) is a de facto standard.
5.2) Security Layer of DCI Middleware Stacks

- If you are lucky, all software components using (login, password) are maintained in a consistent security layer.
- This security layer must now handle X509 certificates.
- This permit any user, depending on his/her access rights, to use shared resources or share his/her own ones. In particular, he/she can submit jobs (tasks, activities) on remote resources.
- But a job executed on a remote resource often has to process data in another location, with the access rights of the job submitter.
- This is a complicated problem called ‘delegation’, which the security layer also has to solve.
- Each instance of this security layer must work consistently in each of the various Administrative Domains of one given DCI, and this requires very careful standardization.
5.3) GLUE 2.0 (OGF) permits to describe DCI entities
5.3) GLUE 2.0 (OGF) permits to describe DCI entities

An **Endpoint** accepts an **Activity** (Job or Data) following the **AccessPolicy** defined for the **UserDomain** (VO, VRC) containing the user having submitted the **Activity**.
5.3) GLUE 2.0 (OGF) permits to describe DCI entities

An **Endpoint** accepts an **Activity** (Job or Data) following the **AccessPolicy** defined for the **UserDomain** (VO, VRC) containing the user having submitted the **Activity**.

This **Activity** is allocated to the **Share** defined by the **MappingPolicy** for the **UserDomain**, and transmitted to a **Manager** (Batch system) for execution on a **Resource** (consistent set of physical machines).
5.3) GLUE 2.0 (OGF) permits to describe DCI entities

An Endpoint accepts an Activity (Job or Data) following the AccessPolicy defined for the UserDomain (VO, VRC) containing the user having submitted the Activity.

This Activity is allocated to the Share defined by the MappingPolicy for the UserDomain, and transmitted to a Manager (Batch system) for execution on a Resource (consistent set of physical machines).

A Service, managed inside an AdminDomain, aggregates a consistent set of Endpoints, Shares and Managers.
5.3) GLUE 2.0 (OGF) permits to describe DCI entities

An **Endpoint** accepts an **Activity** (Job or Data) following the **AccessPolicy** defined for the **UserDomain** (VO, VRC) containing the user having submitted the **Activity**.

This **Activity** is allocated to the **Share** defined by the **MappingPolicy** for the **UserDomain**, and transmitted to a **Manager** (Batch system) for execution on a **Resource** (consistent set of physical machines).

A **Service**, managed inside an **AdminDomain**, aggregates a consistent set of **Endpoints**, **Shares** and **Managers**.

The **Service** should keep persistent **ActivityRecords** (Logging and Accounting), but this is **not** well standardized yet.
5.4) Standardization of Support Functionalities

- **Security**: As shown in a previous slide, there are a lot of standards. These should be consolidated.

- **Information**: GLUE 2.0 (OGF) is an official standard. It should be used as foundation.

- **Monitoring**: 'OCCI Req. Table 7' (OGF) is a specification permitting to define a standard.

- **VM Image**: OVF = Open Virtualization Format (DMTF) is an official standard.

- **Log**: 'Activity Instance Documents Specification' (OGF) is a proposal for a standard.

- **Accounting**: UR (OGF) is an official standard, but I do **not** know in which extent it is implemented.
5.5) Standardization of Payload Functionalities

Data: GridFTP, ByteIO, SRM, DMI (OGF), CDMI (SNIA)

Instrument: DORII (not a standard yet)

Cloud: OCCI (OGF)

Job: JSDL, OGSA-BES, HPC profiles (OGF)

- These standards exist, but are not consistent between each other, nor consistent with ‘support’ standards.

- They should be consolidated using GLUE 2.0
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

5.6) Useful official and de facto standards for DCIs
Software Engineering for Distributed Data Processing Infrastructures (DCIs)

5.7) Problem of the backend interfaces

- For each functionality, it is necessary to standardize the client interface.
- But interoperability between different DCIs requires usage of backend interfaces permitting access to resources.
- Then, these backend interfaces also have to be standardized.
- For each functionality, I see only two solutions:
  - The standard describing the client interface also describes the backend interface (this is heavy and monolithic)
  - We have to completely separate the definition of the data flowing through the interface and the definition of the dialog protocol, and/or define a specific standard for the backend interface (this generates a lot of standards to create and maintain)

- See next slide for the concept.
- Could anybody propose a solution to this problem?
5.7) Problem of the backend interfaces