The Storage Abstraction Service of the SPAR system
Agenda

- Objective and key requirements
- The global architecture
- The Storage Abstraction Service
- Implementation with iRods
- Conclusion
Main missions:
- to build up the collections
- to preserve them forever
- to communicate them to the public

Legal deposit:
- legal deposit since 1537 for printed materials
- 1648: engravings and maps
- 1793: musical scores
- 1925: photos
- 1938: phonograms
- 1941: posters
- 1975: videograms and multimedia documents
- 1992: audiovisual and electronic documents
- 2006: Web legal deposit
Digital archiving at BnF

Production applications

- Preservation digitization
- Record management
- WEB Archiving

Dissemination applications

- wayback

Production applications:
- Preservation
- Digitization
- Record management

Dissemination applications:
- gallica
- wayback
Key requirements

- OAIS compliance (ISO 14721:2003)
- modularity and distributivity
- abstraction
- use of well known formats and standards
- use of Open Source technical building blocks
Implementation

Pre Ingest
- OAI repository (descriptive metadata catalog)

Ingest
- Identity management
- Solon (rights metadata management)

Preservation Administration
- Data management

Rights management
- Access

Storage
- Storage Abstraction Service (SAS)

Users

Producer

SPAR - Infrastructure

SPAR - Realization

SPAR - Distribution Archiving & Preservation System (SPAR)
Agenda

- Objective and key requirements
- The global architecture
- The Storage Abstraction Service
- Implementation with iRods
- Conclusion
Modularity of SPAR
The infrastructure

SSS : secondary storage

SSP : main storage

SSC : access storage

SSS-B : backup secondary storage

SSB : backup storage

Servers

Main site

Backup site
Agenda

- Objective and key requirements
- The global architecture
- The Storage Abstraction Service
- Implementation with iRods
- Conclusion
Requirements for the Storage abstraction service

- Abstract the infrastructure with:
  - storage unit
    - aggregates storage element to defines an abstract storage defined by a class of service
      - mean time (read/write)
      - number of copies …
  - record
    - simple bit stream (no semantics)
    - some properties: checksum, logical name, lastAuditDate
Main concepts for the SAS

- **A record** is a information managed by the SAS. Each record is hosted on one and only one storage unit.

- **A copy (or replica):** Depending on the needs, a record can have multiple copies on the SAS. All copies are strictly identical (integrity control). The SAS only exposes one record with multiple copies. Each copy is written physically in a storage element.

- **A storage unit:** It handles an entity to capture the characteristics of a particular storage. Each unit is declared by an administrator who defines which elements of storage are linked as well as the number of copies. Every record in one storage unit has the same characteristics of storage.

- **A storage element:** it is an entity very closed to the hardware (media, disk, tapes), except that it represents a part of a physical media. For example: a partition of a disk array or a pool of tapes…
Global design

SPAR - Realization

Handles multiple copies

Main storage
Media replication

Backup storage
Media migration

SPAR - Infrastructure

SAS

Software viewpoint
Infrastructure viewpoint

AQS
AIP

02/02/2009
Distributed Archiving & Preservation System (SPAR)
Logical Data Model

**record**
- recordName
- recordCategory
- recordStatus
- auditStatus
- currentSize
- md5sum
- creationDate
- lastUpdateDate
- lastAuditDate

**replica**
- replicaName
- replicaStatus
- physicalLocation
- md5sum
- creationDate
- lastUpdateDate

**storageUnit**
- storageUnitName
- storageUnitStatus
- weight
- currentSize
- maxRecordNumber
- maxRecordSize
- storageCapacity
- availabilityPerDay
- availabilityPerYear
- timeStoreRecord
- timeGetRecord
- timeCopyRecord
- firstStorageElementName
- storageType
- creationDate
- lastUpdateDate

**readLock**
- lockId
- owner
- creationDate

**writeLock**
- lockId
- owner
- creationDate
Agenda

- Objective and key requirements
- The global architecture
- The Storage Abstraction Service
- Implementation with iRods
- Conclusion
Implementation choice

- iRods
  - scalable and proven
  - rules based

- Use of the rules to configure a storage unit
  - rule for put
  - rule for audit
  - rule for get

- General rules for
  - refreshment migration
  - duplication migration
Technical implementation: iRODS
iRODS : Logical objects ⇔ Physical copies

- SPAR - Realization
- SPAR_SAS
  - Java.runtime
  - iRODS Client
- API POSIX
- SAM-FS
- WebServices, mail
- Administration

Clients

- iCAT
- µService
- rule
- iRODS Client

iRODS

- Physical copies
- Logical objects

Main site

Backup site

Module SAS

SPAR - Realization
Dialog between Storage module and SAS

→ Storing an AIP
  - corresponding SLA search (storage requirements)
  - retrieval of available storage units with compatible class of service
  - choose of the least expensive storage unit
  - store the AIP as a record

→ Audit of an AIP
  - if audit time has expired,
  - asked the SAS for an audit of the copies
  - retrieved the package itself for internal audit

→ Retrieve of an AIP
  - get back the first ok copy
  - if one is found bad, launch an audit
Storing an AIP

sasPUT(*rodsPath,*localFilePath,*mainResource,*inputChecksum)

acObjPutWithDateAndChksumAsAVUs(
  *rodsPath,*mainResource,*localFilePath,
  *inputChecksum,*outstatus)

assign(*replicasAttributeCondition,
  RESC_NAME = '*mainResource' 
  AND META_RESC_ATTR_NAME = 'replicaResources')

acGetValueForResourceMetaAttribute(
  *replicasAttributeCondition,*replList)

ifExec(*replList != none,
  forEachExec(*replList,
    writeLine(stdout,"replicating to *replList ")
    delayExec(<PLUSET>1m</PLUSET>,
      msiDataObjRepl(*rodsPath,*replList,*replStatus),nop),
    nop)
  ,
  nop,nop,nop
)

Save replica 1 with checksum test
Query for replica resources
Get the list
Iterate to plan the creation of each replica
Audit an AIP

sasAUDIT(*rodsName,*parentColl,*stageDir)||
assign(*rodsPath,null)##
acGetFullPathFromDataObjParentCollection(*rodsName,*parentColl,*rodsPath)##
assign(*locationsCondition,COLL_NAME LIKE '*parentColl/%' AND DATA_NAME = '*rodsName')##
assign(*storedChecksumCondition,*locationsCondition AND META_DATA_ATTR_NAME = 'MD5SUM')##
acGetValueForDataObjMetaAttribute(*storedChecksumCondition,*objStoredChksum)##
acGetDataObjLocations(*locationsCondition,*matchingObjects)##
forEachExec(*matchingObjects,
 msiGetValByKey(*matchingObjects,RESC_LOC,*objReplicaHost),
 msiGetValByKey(*matchingObjects,DATA_PATH,*objPhysicalPath),
 msiGetValByKey(*matchingObjects,RESC_NAME,*currRescName),
 remoteExec(*objReplicaHost,null,
    acGetPhysicalDataObjMD5SUM(*objPhysicalPath,*objReplicaHost,*objPhysicalMD5),nop)##
ifExec(*objStoredChksum == *objPhysicalMD5,
    writeLine(stdout,"input and computed MD5 checksums match"),
    acReplaceStaleReplica(*rodsPath,*currRescName,*replStatus),
    nop)
writeLine(stdout,"getting *rodsName to *stageDir/*rodsName")##
msiDataObjGet(*rodsPath,*stageDir/*rodsName,*getStatus)

Retrieve all the replicas
Calculate the checksum
Replace the bad replica
Get a good replica for functional audit
Retrieving an AIP

sasGET(*rodsName,*parentColl,*stageDir) ||
 assign(*rodsPath,null)##
 acGetFullPathFromDataObjParentCollection(*rodsName,*parentColl,*rodsPath)##
 assign(*locationsCondition,DATA_NAME = '*rodsName')##
 assign(*storedChecksumCondition,*locationsCondition AND META_DATA_ATTR_NAME = 'MD5SUM')##
 assign(*goodReplicaEncountered,0)##
 acGetValueForDataObjMetaAttribute(*storedChecksumCondition,*objStoredChksum)##
 acGetDataObjLocations(*locationsCondition,*matchingObjects)##
 forEachExec(*matchingObjects,
   ifExec(*goodReplicaEncountered == 0,
     msiGetValByKey(*matchingObjects,RESC_LOC,*objReplicaHost)##
     msiGetValByKey(*matchingObjects,DATA_PATH,*objPhysicalPath)##
     msiGetValByKey(*matchingObjects,RESC_NAME,*currRescName)##
     remoteExec(*objReplicaHost,null,
       acGetPhysicalDataObjMD5SUM(*objPhysicalPath,*objReplicaHost,*objPhysicalMD5),nop)##
     ifExec(*objStoredChksum == *objPhysicalMD5,
       assign(*goodReplicaEncountered,1),nop,
       delayExec(<PLUSET>1m</PLUSET>,
         acReplaceStaleReplica(*rodsPath,*currRescName,*replStatus),nop
       ),nop
     ),nop,nop,nop
   )##
   msiDataObjGet(*rodsPath,*stageDir/*rodsName,*getStatus)
)

No good replica yet found

Calculate the checksum

Plan the replacement of the bad replica

Get the first good replica
 Agenda

- Objective and key requirements
- The global architecture
- The Storage Abstraction Service
- Implementation with iRods
- Conclusion
Conclusion

- Goal for the archived objects
  - definition of an open model
  - completeness of the description
  - self-supporting package

- Ways of dealing with the permanency
  - modularity
  - abstraction
  - use of well known formats and standards
  - use of Open Source technical building blocks
Thank you for your attention

Questions?

More information: http://bibnum.bnf.fr/spar

Thomas Ledoux
thomas.ledoux_AT_bnf.fr