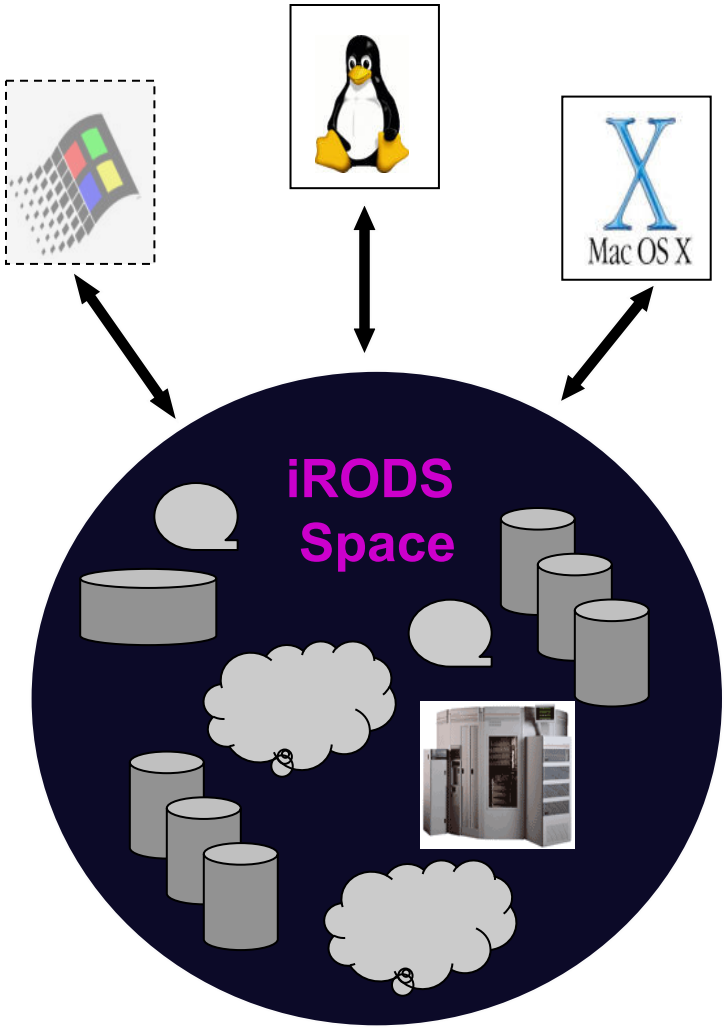
How to Build µServices



D·I·C·E

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

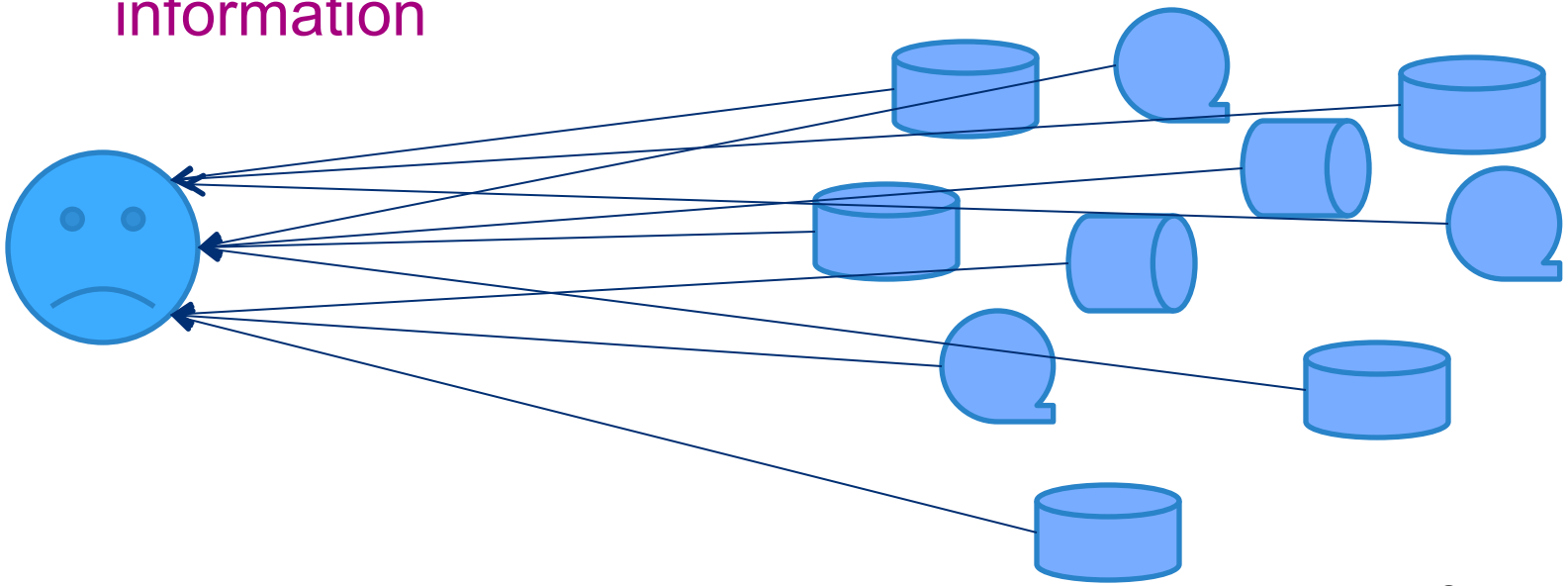
NSF

UCSD

i·R·O·D·S

What is a Data Grid? (hardware)

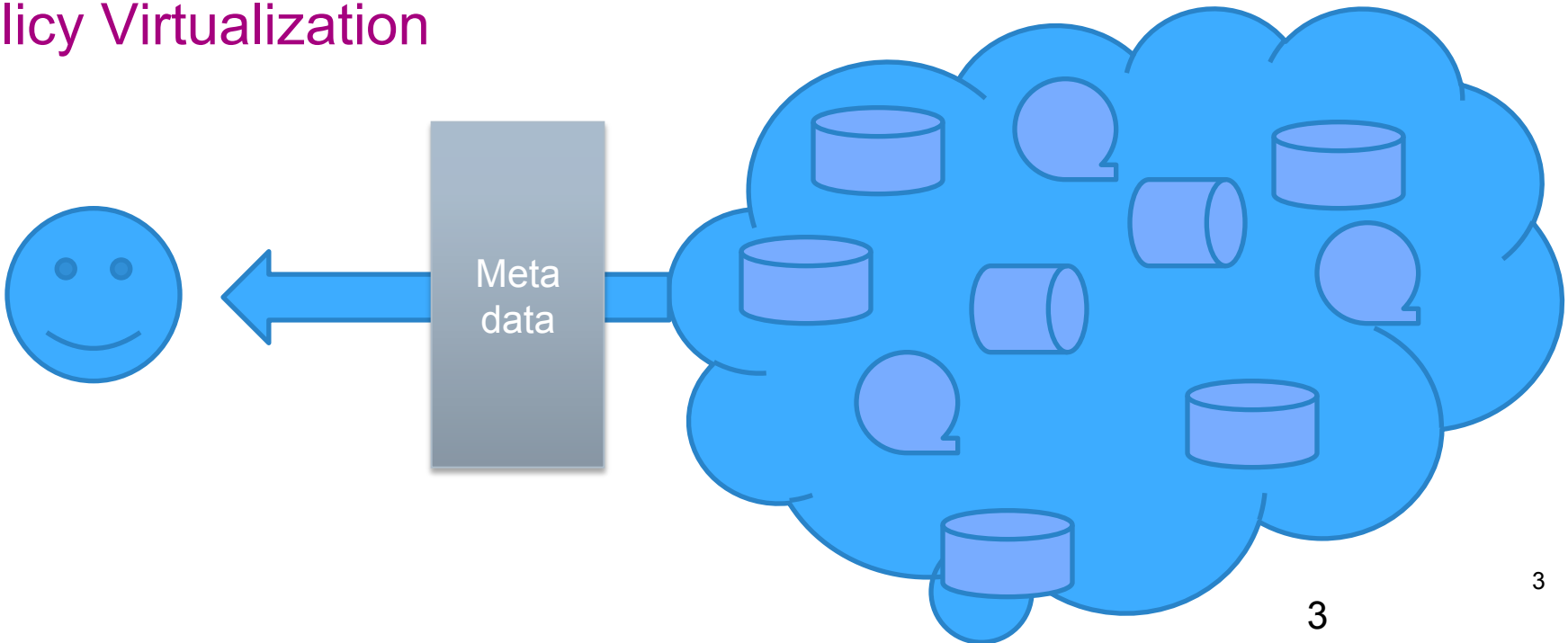
- Geographically distributed heterogeneous resources that are managed autonomously
- Active with data resources being added and removed
- Users like to share/discover data using contextual information



2

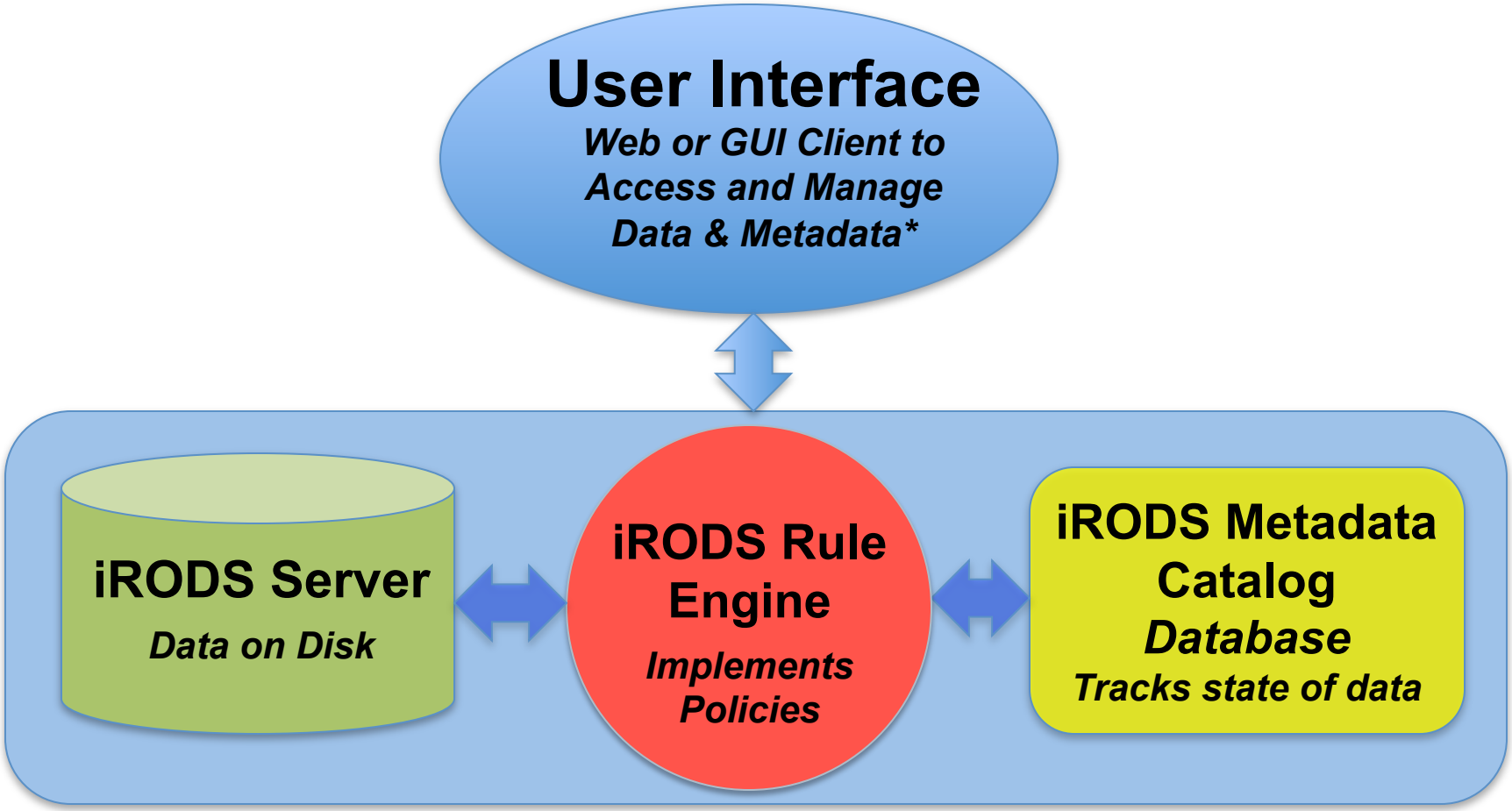
What is a Data Grid? (data collection)

- Data Grid – a network of data resources that is presented as a single, accessible collection of data.
- Data Grid – provisions for associating metadata & annotations
- Data Grid – enables discovery, access & server-side processing
- Metadata-based data virtualization
- Policy Virtualization



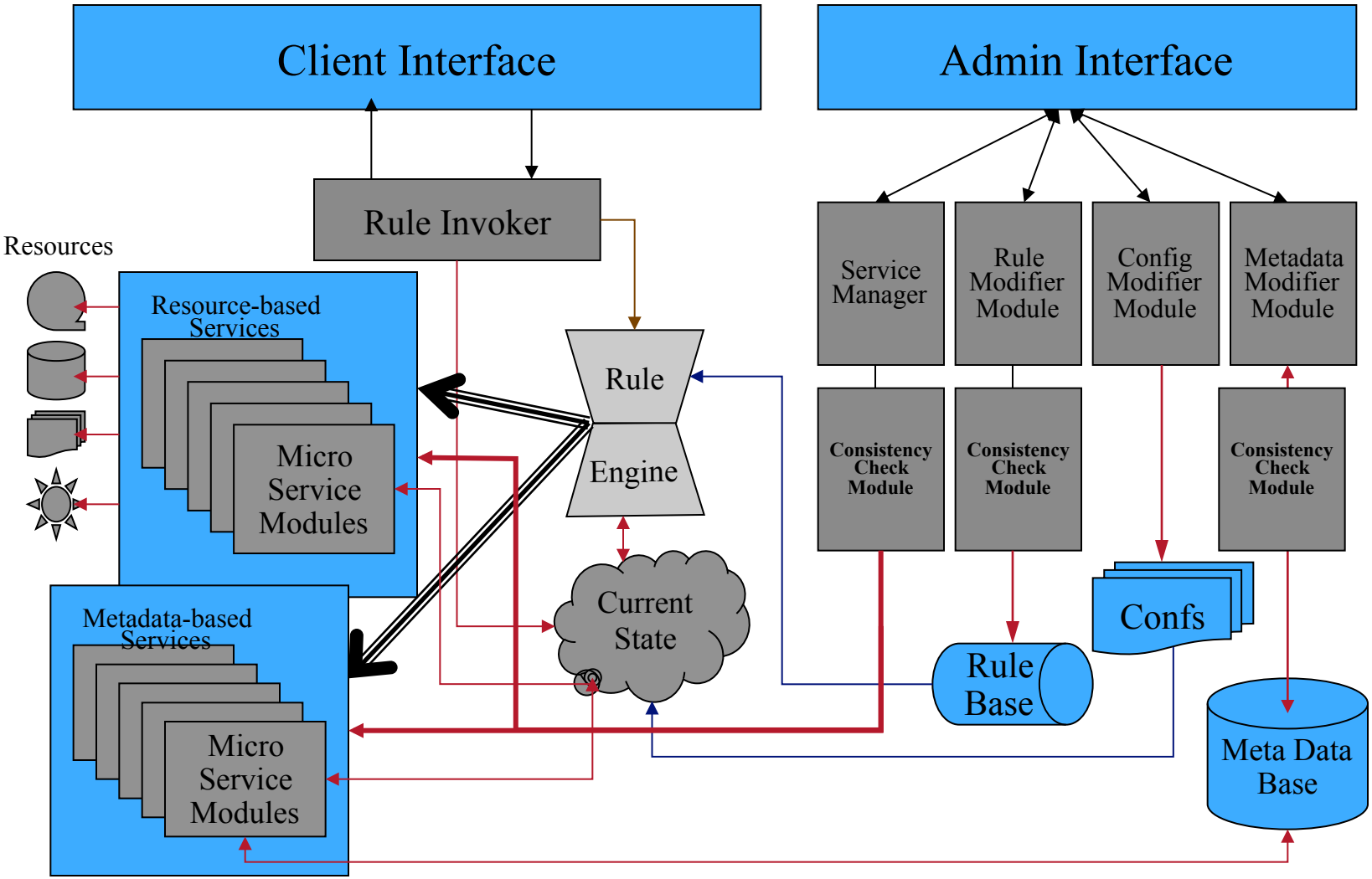
3

Overview of iRODS Components



*Access data with: Web-based Browser, iRODS GUI, Command Line clients, Dspace, Fedora, Kepler workflow, WebDAV, user level file system, etc.

iRODS Architecture



Micro Services (msi)

- Compiled C Functions
- Short and Well-defined functionality
- Should have a clear semantics about what it does
- Examples:
  - Metadata Extraction from DICOM images
  - Access Control Permission Changed for a User
  - Replicate a file from Source to Destination
- Usage:

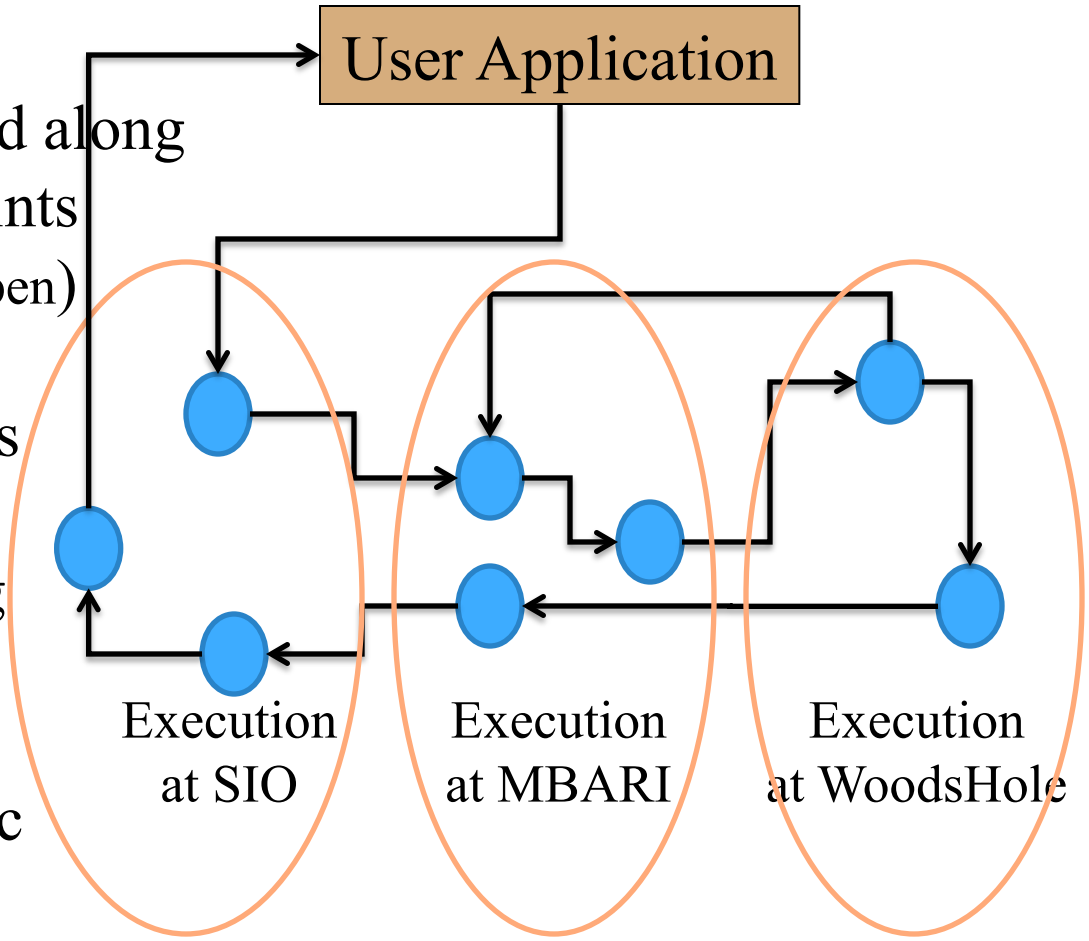
```
RuleName {
    on (Condition) {
        MS1 ::: RMS1;
        MS2 ::: RMS2;
    }
}
```

Policy Virtualization with iRODS

Micro-Services
- Functions with well-defined semantics
- Transactional - recovery
- Context of application carried along

Rules: Policy Enforcement Points
- Triggered by events (say OnOpen)
- Conditional execution of alternative rule declarations
- System constructs: loops, recursion, branching

Workflows
- Distributed Execution
- Immediate, Deferred, Periodic
- Remote, Data-Aware



Policy Enforcement Points (71)

ACTION	PRE-ACTION POLICY	POST-ACTION POLICY
acCreateUser	acPreProcForCreateUser	acPostProcForCreateUser
acDeleteUser	acPreProcForDeleteUser	acPostProcForDeleteUser
acGetUserbyDN	acPreProcForModifyUser	acPostProcForModifyUser
acTrashPolicy	acPreProcForModifyUserGroup	acPostProcForModifyUserGroup
acAclPolicy	acChkHostAccessControl	acPostProcForDelete
acSetCreateConditions	acPreProcForCollCreate	acPostProcForCollCreate
acDataDeletePolicy	acPreProcForRmColl	acPostProcForRmColl
acRenameLocalZone	acPreProcForModifyAVUMetadata	acPostProcForModifyAVUMetadata
acSetRescSchemeForCreate	acPreProcForModifyCollMeta	acPostProcForModifyCollMeta
acRescQuotaPolicy	acPreProcForModifyDataObjMeta	acPostProcForModifyDataObjMeta
acSetMultiReplPerResc	acPreProcForModifyAccessControl	acPostProcForModifyAccessControl
acSetNumThreads	acPreprocForDataObjOpen	acPostProcForOpen
acVacuum	acPreProcForObjRename	acPostProcForObjRename
acSetResourceList	acPreProcForCreateResource	acPostProcForCreateResource
acSetCopyNumber	acPreProcForDeleteResource	acPostProcForDeleteResource
acVerifyChecksum	acPreProcForModifyResource	acPostProcForModifyResource
acCreateUserZoneCollections	acPreProcForModifyResourceGroup	acPostProcForModifyResourceGroup
acDeleteUserZoneCollections	acPreProcForCreateToken	acPostProcForCreateToken
acPurgeFiles	acPreProcForDeleteToken	acPostProcForDeleteToken
acRegisterData	acNoChkFilePathPerm	acPostProcForFilePathReg
acGetIcatResults	acPreProcForGenQuery	acPostProcForGenQuery
acSetPublicUserPolicy	acSetReServerNumProc	acPostProcForPut
acCreateDefaultCollections	acSetVaultPathPolicy	acPostProcForCopy
acDeleteDefaultCollections		acPostProcForCreate

Micro-services - How many are needed?

print_hello
print_hello_arg
msiVacuum
msiQuota
msiDeleteUnusedAVUs
msiGoodFailure
msiSetResource
msiCheckPermission
msiCheckOwner
msiCreateUser
msiCreateCollByAdmin
msiSendMail
recover_print_hello
msiCommit
msiRollback
msiDeleteCollByAdmin
msiDeleteUser
msiAddUserToGroup
msiSetDefaultResc
msiSetRescSortScheme
msiSysReplDataObj
msiStageDataObj
msiSetDataObjPreferredResc
msiSetDataObjAvoidResc
msiSortDataObj
msiSysChksumDataObj
msiSetDataTypeFromExt
msiSetNoDirectRescInp
msiSetNumThreads
msiDeleteDisallowed

msiOprDisallowed
msiDataObjCreate
msiDataObjOpen
msiDataObjClose
msiDataObjLseek
msiDataObjRead
msiDataObjWrite
msiDataObjUnlink
msiDataObjRepl
msiDataObjCopy
msiExtractNaraMetadata
msiSetMultiReplPerResc
msiAdmChangeCoreIRB
msiAdmShowIRB
msiAdmShowDVM
msiAdmShowFNM
msiAdmAppendToTopOfCoreIRB
msiAdmClearAppRuleStruct
msiAdmAddAppRuleStruct
msiGetObjType
msiAssociateKeyValuePairsToObj
msiExtractTemplateMDFromBuf
msiReadMDTemplateIntoTagStruct
msiDataObjPut
msiDataObjGet
msiDataObjChksum
msiDataObjPhymv
msiDataObjRename
msiDataObjTrim
msiCollCreate

msiRmColl
msiReplColl
msiCollRepl
msiPhyPathReg
msiObjStat
msiDataObjRsync
msiCollRsync
msiFreeBuffer
msiNoChkFilePathPerm
msiNoTrashCan
msiSetPublicUserOpr
whileExec
forExec
delayExec
remoteExec
forEachExec
msiSleep
writeString
writeLine
writeBytesBuf
writePosInt
writeKeyValPairs
msiGetDiffTime
msiGetSystemTime
msiHumanToSystemTime
msiStrToBytesBuf
msiApplyDCMetadataTemplate
msiListEnabledMS
msiSendStdoutAsEmail
msiPrintKeyValPair

msiGetValByKey
msiAddKeyVal
assign
ifExec
break
applyAllRules
msiExecStrCondQuery
msiExecStrCondQueryWithOptions
msiExecGenQuery
msiMakeQuery
msiMakeGenQuery
msiGetMoreRows
msiAddSelectFieldToGenQuery
msiAddConditionToGenQuery
msiPrintGenQueryOutToBuffer
msiExecCmd
msiSetGraftPathScheme
msiSetRandomScheme
msiCheckHostAccessControl
msiGetIcatTime
msiGetTaggedValueFromString
msiXmsgServerConnect
msiXmsgCreateStream
msiCreateXmsgInp
msiSendXmsg
msiRcvXmsg
msiXmsgServerDisConnect
msiString2KeyValPair
msiStrArray2String
msiRdaToStdout

Micro-services (229)

msiRdaToDataObj
msiRdaNoResults
msiRdaCommit
msiAW1
msiRdaRollback
msiRenameLocalZone
msiRenameCollection
msiAclPolicy
msiRemoveKeyValuePairsFromObj
msiDataObjPutWithOptions
msiDataObjReplWithOptions
msiDataObjChksumWithOptions
msiDataObjGetWithOptions
msiSetReServerNumProc
msiGetStdoutInExecCmdOut
msiGetStderrInExecCmdOut
msiAddKeyValToMspStr
msiPrintGenQueryInp
msiTarFileExtract
msiTarFileCreate
msiPhyBundleColl
msiWriteRodsLog
msiServerMonPerf
msiFlushMonStat
msiDigestMonStat
msiSplitPath
msiGetSessionVarValue
msiAutoReplicateService
msiDataObjAutoMove
msiGetContInxFromGenQueryOut

msiSetACL
msiSetRescQuotaPolicy
msiAdmReadRulesFromFileIntoStruct
msiAdmInsertRulesFromStructIntoDB
msiGetRulesFromDBIntoStruct
msiAdmWriteRulesFromStructIntoFile
writeXMsg
readXMsg
msiSetReplComment
msiSetBulkPutPostProcPolicy
msiVerifyOwner
msiVerifyAVU
msiVerifyACL
msiVerifyExpiry
msiVerifyDataType
msiVerifyFileSizeRange
msiVerifySubCollOwner
msiVerifySubCollACL
msiVerifySubCollAVU
msiListFields
msiHiThere
msiTestWritePosInt
msiListColl
msiTestForEachExec
msiGetFormattedSystemTime
msiPropertiesNew
msiPropertiesClear
msiPropertiesClone
msiPropertiesAdd
msiPropertiesRemove

msiPropertiesGet
msiPropertiesSet
msiPropertiesExists
msiPropertiesToString
msiPropertiesFromString
msiRecursiveCollCopy
msiGetDataObjACL
msiGetCollectionACL
msiGetDataObjAVUs
msiGetDataObjPSmeta
msiGetCollectionPSmeta
msiGetDataObjAIP
msiLoadMetadataFromDataObj
msiExportRecursiveCollMeta
msiCopyAVUMetadata
msiGetUserInfo
msiGetUserACL
msiCreateUserAccountsFromDataObj
msiLoadUserModsFromDataObj
msiDeleteUsersFromDataObj
msiLoadACLFromDataObj
msiGetAuditTrailInfoByUserID
msiGetAuditTrailInfoByObjectID
msiGetAuditTrailInfoByActionID
msiGetAuditTrailInfoByKeywords
msiGetAuditTrailInfoByTimeStamp
msiSetDataType
msiGuessDataType
msiMergeDataCopies
msiIsColl

msiIsData
msiGetCollectionContentsReport
msiGetCollectionSize
msiStructFileBundle
msiCollectionSpider
msiFlagDataObjwithAVU
msiFlagInfectedObjs

What do Micro-Services use?

- Micro-services communicate through:
  - Arguments/Parameters
    - Inside a rule from one micro-service to another rule or micro-service
  - Session Memory – white board ($-variables)
    - Stores common (context) information
      - User and resource information
      - More information about Data or collection of interest

---

  - Persistent Memory – iCat
    - Query an iCAT for information (coded inside the micro-service)
  - XMessages – out-of-band communications
    - Like a Post Office

msParam_t :

All μServices have only two types of parameters

Params 1…(n-1) are of the type msParam_t

Param n is of the type ruleExecInfo_t

msParam_t is defined as:

```
typedef struct MsParam {
    char *label;
    char *type;
    void *inOutStruct;
    bytesBuf_t *inpOutBuf; } msParam_t;
```

ruleExecInfo_t is the "white board" used for passing session-oriented parameters that can be used by the Rule Engine and the micro-services.

```
int msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam,
                   ruleExecInfo_t *rei);
int msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam,
                   msiParam_t *tagParam, ruleExecInfo_t *rei);
```

Parameter Passing

- Part of the MicroService Signature

```
int findObjType ( msiParam_t *objInParam ,
                  msiParam_t *typeOutParam ,
                  ruleExecInfo_t *rei );
int ingestBulkMD ( msiParam_t *objInParam,
                   msiParam_t *typInParam,
                   msiParam_t
                       *keyValuePairsInParam,
                   ruleExecInfo_t *rei);
```

- When used in a rule the “rei” parameter is implicit.

WhiteBoard ($) Variables

- They are stored in a structure: rei
- Some common ones that are of interest
  - objPath collection-path name of data object
  - rescName name of resource
  - userNameClient name of user
- How to Use them:
  - Condition checking:

    $objPath like /zone/home/sekar/nvo/*
  - Parameter passing:

    findObjType($objPath,*Type)
  - assign($rescName, duke-samqfs)
- You can find the $-variable names in:
  - server/config/reConfigs/core.dvm

WhiteBoard: ruleExecInfo (*rei)

- A large data structure shared when invoking a rule
- Implicitly used throughout the rule processing
- MicroServices can access values/structs in the *rei and also set values in the *rei structure
- The structure is defined in reGlobalsExtern.h and it can be extended if necessary
- Contains various important structures used in the iRODS data management:
  - *rsComm - client-server communication structure
  - *doi - dataObject information
  - *rescGrp - resource (group) informations
  - *uoic - client user information
  - and others ….
- The rule invoking function should set the proper values…

Data Components

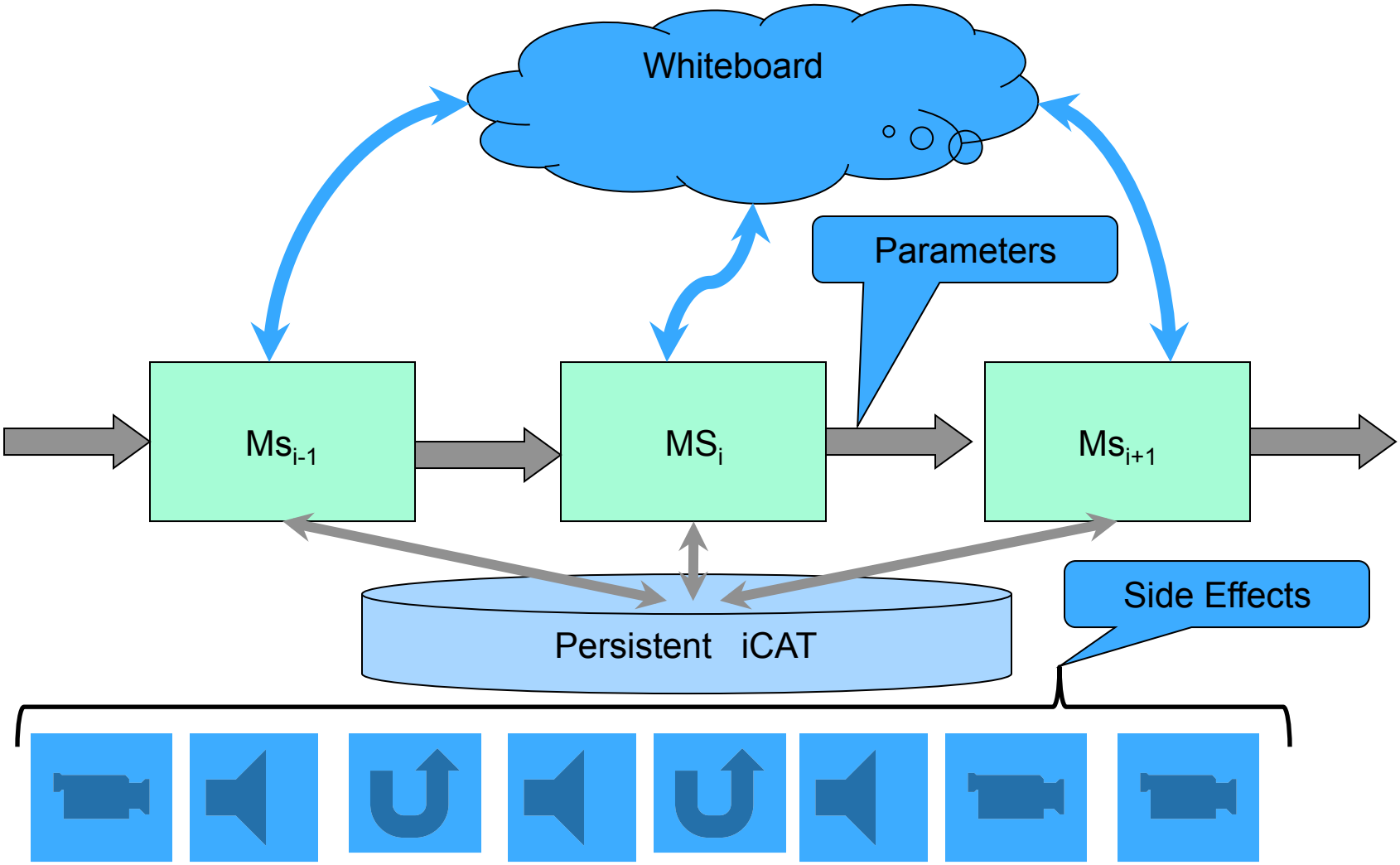


Figure 1: Micro-services and their data components and side-effects

Launchng μServices: What do we learn here?

Development of Modules

- Extending iRODS with new code

Implementation of μServices

- How each module is coded to be μ-compliant

Enabling & Linking of Modules

- Make it work with iRODS server

Testing of μServices

- From the command line.

More involved example

- As a take home

Development of Modules

Modules and μServices

- Module is a .way of adapting iRODS with your own extensions
  - Micro-services in modules can be used like any other micro-services
- Easy to upgrade with newer releases of iRODS
- Easy to manage your own rules/micro-services and required external libraries.
- Easy to share with others – publish your micro-services for others to use.
  - We can help with publication on iRODS Wiki
- Modules are directories in the iRODS source tree.
  - They have a particular structure & requirements
  - iRODS setup and make scripts make it easy to integrate modules with the core iRODS

How to create a new module?

- A module is created in /iRODS/modules directory
- Here are some ways to get information about this topic:
- The iRODS wiki page:

https://www.irods.org/index.php/
How_to_create_a_new_module
provides information about this.

- In the iRODS source, the “doc” directory has files
  HOW_TO_ADD_A_MICROSERVICE.txt
  HOW_TO_ADD_A_MODULE.txt
  HOW_TO_USE_A_MODULES_INFO_TXT.txt

OnMicroServiceHelpers.txt (1)

```
# Create a directory named for the module:
cd modules
mkdir mytest
cd mytest
# Create required sub-directories
mkdir microservices
mkdir microservices/src
mkdir microservices/include
mkdir microservices/obj
# Copy useful files from another module and modify it
#   properties module is a good candidate as it is simple
# You are still in modules/mytest directory
cp ../properties/Makefile   .
cp ../properties/info.txt      .
```

OnMicroServiceHelpers.txt (2)

```
# Now open and modify info.txt  file to suit your needs
# change name, descriptions, creator, etc
# enable the module!!!!!!
vi info.txt
# Now open and modify Makefile   file to suit your needs
# first we brute-force change all occurrences of "properties" to "mytest"
# then we basically modify information about:
# what flags to use,  external libraries to link , source files to complie
vi Makefile
# In our case, I am planning to create and use a file called mytestMS.c
# to write my micro-services and hence no changes are needed
# except for the brute-force changes.
# In more exotic cases you may have to change a lot of other things.
# See the wiki page or READ files in iRODS/doc files
# See other modules (eg. hdf5 or XML for more involved modules)
```

OnMicroServiceHelpers.txt (3)

```
# Next we need to setup the information to help iRODS use the
# micro-services in the module. This is done by adding two files
# microservices.header and microservices.tablen
# in microservices/include directory
cd microservices/include
vi microservices.header

# Add these two lines in microservices.header
# I am assuming that I will need an include file called mytestMS.h

/* Mytest module microservices */
#include "mytestMS.h“
```

OnMicroServiceHelpers.txt (4)

```
# Next we set up the microservices.table file.
vi microservices.table

# add these three lines
# I am assuming that my micro-service is named "msiMyTestOne"
# and needs two parameters

/* Mytest  module microservices */
/* Service name                 # args          Service function */
{ "msiMyTestOne",                 2,            (funcPtr) msiMyTestOne },

# the comma at the end of the line  is NEEDED…..
# if I have other micro-services, I will add them in separate lines.
```

NEXT WE WRITE THE MICRO_SERVICE

Creation of μService

OnMicroServiceHelpers.txt (5)

The Function:

Steps in Writing a micro-service in
a. Name it and Create the signature
b. Add the required test-macro call
c. Check the integrity of all the parameters
d. Coerce all input parameters into local variable
e. Add the core part of your micro-service functionality
f. Write out the output parameters in msParam_t format
g. return 0 for success

Other helpful information:

h. add copyrights
i. add file level comments
j add include lines as needed
k. add DOXYGEN-style comments for each micro-service

The Function - in mytestMS.c

```
Int     msiMyTestOne(msParam_t *inString, msParam_t *outString,  ruleExecInfo_t *rei )
{
  /*   local variable */
  char *mystr;
  int i;
  RE_TEST_MACRO( "    Calling msiMyTestOne" );    /* test MACRO call */
  /*  check  input parameters. Return error if necessary */
  if (inString ==  NULL || strcmp(inString->type , STR_MS_T) != 0 || inString->inOutStruct == NULL)
    return(USER_PARAM_TYPE_ERR);
  /*  coerce input to local variables */
  mystr = (char *) inString->inOutStruct;
  /* do the work */    /* basically checking if a string is a positive integer */
  for (i = 0; i < strlen(mystr); i++ ) {
    if(!isdigit(mystr[i])) {
      i = -1;
      break;
    }
  }
  /* write output in msParam */
  if (i < 0)
       fillStrIngMsParam (outString, "THIS IS NOT  A POSITIVE INTEGER");
  else
       fillStrIngMsParam (outString, "THIS IS A POSITIVE INTEGER");
  return(0);
}
```

The Header Part - mytestMS.c

Put in the copyright information

```
/*** Copyright (c), The University of North Carolina            ***
 *** For more information please refer to files in the COPYRIGHT directory ***/
```

Put in the information used by comments extractor

```
/**
 * @file  mytestMS.c
 *
 * @brief testing micro-service writing
 *
 *
 * @author  AR
 */
```

Add the required header and nclude files

```
#include "rsApiHandler.h“   /*needed only if you are calling iRODS server-functions*/
#include "mytestMS.h"
}
```

The Doxygen for the micro-service- mytestMS.c

```
/**
 * \fn msiMyTestOne( msParam_t *inString, msParam_t *outString, ruleExecInfo_t *rei )
 * \brief test
 * \module mytest
 * \since 3.2
 * \author  AR
 * \date    2011
 * \remark
 * \note test for finding if a string represents a positive integer
 * \usage None
 * \param[in] inString - a STR_MS_T, input value
 * \param[out] outString - a STR_MS_T, output value
 * \param[in,out] rei - The RuleExecInfo structure that is automatically
 *    handled by the rule engine. The user does not include rei as a
 *    parameter in the rule invocation.
 *
 * \DolVarDependence none
 * \DolVarModified none
 * \iCatAttrDependence none
 * \iCatAttrModified none
 * \sideeffect none
 *
 * \return integer
 * \retval 0 on success
 * \pre none
 * \post none
 * \sa none
 * \bug  no known bugs
**/
```

Finally the include file- mytestMS.h

#Create this file in the directory modules/mytest/microservices/include

#Put in the copyright information

```
/*** Copyright (c), The University of North Carolina            ***
 *** For more information please refer to files in the COPYRIGHT directory ***/
```

#Put in the information used by comments extractor

```
/**
 * @file        mytestMS.h
 *
 * @brief       Declarations for the msiMyTest microservices.
 */

#ifndef MYTESTMS_H            /* this makes sure that this file does not get included more than once */
#define MYTESTMS_H

#include "rods.h““                              /*these are needed only if you are calling iRODS server-functions*/
#include "reGlobalsExtern.h"
#include "rsGlobalExtern.h"
#include "rcGlobalExtern.h"
```

#Finally the Signature

```
/* This micro service takes in a string and tells whether it is an integer or not */
int msiMyTestOne (msParam_t *inString, msParam_t *outString, ruleExecInfo_t* rei );

#endif  /* MYTESTMS_H */
```

Enabling a Module

Three ways to enable a module

- Preliminaries
  - Make sure the iRODS server is stopped (./irodsctl istop)
- First Method
  - Enable the module in info.txt (Enable yes)
  - Run ./irodssetup or ./irodsupgrade
  - Accept existing setup values in iRODS/config/irods.config
- Second Method
  - Change iRODS configuration to enable the module
  - ./configure --enable-module_name (example: ./configure --enable-mytest)
  - Run ./irodssetup or ./irodsupgrade
- Third Method
  - Edit config/config.mk directly (discouraged) and add your module's name to the MODULES variable
  - Example: MODULES= msoDrivers mytest
  - Run ./irodssetup or ./irodsupgrade

Three ways to enable a module

- Preliminaries- setting up
  - Make sure you have enabled the module in info.txt
  - Make sure the iRODS server is stopped
  - Run gmake clean at the iRODS source directory
- First Method
  - Run irodssetup or irodsupgrade
  - Allow it to use old setup files
- Second Method
  - Change iRODS configururation to enable the module
  - ./configure --enable-mytest
- Third Method
  - Edit config/config.mk directly (discouraged) and add your module's name to the MODULES variable
  - MODULES= msoDrivers mytest
- Finally compile and link
  - Run gmake
  - Start iRODS server

Questions?

Check out www.irods.org

Mail to irod-chat for more help

iRODS - Open Source Software

http://irods.diceresearch.org

NSF OCI-0940841 "DataNet Federation Consortium"
NSF OCI-1032732 "SDCI Data Improvement: Improvement and Sustainability of iRODS Data Grid Software for Multi-Disciplinary Community Driven Application"
NSF OCI-0848296 "NARA Transcontinental Persistent Archives Prototype"
NSF SDCI-0721400 "Data Grids for Community Driven Applications"

D·I·C·E i·R·O·D·S THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL UCSD NSF THE NATIONAL ARCHIVES ARCHIVES.GOV renci

Design of μService

(A more involved example)

Design: Prologue

Problem Statement: Extract metadata from an Email and associate with the Email file.

Generic Solution Steps:

How to

- Extract Metadata Attr-Value pairs from one file
- Based on a Template defined in a second file, and
- Associate the metadata to a third file?

Problem Break up:

Sounds like 3 μServices!

Design: Lets look at the input files

Sample Input File (in our case also Metadata File):

Date: Thu, 01 Feb 2007, 22:33:35 +000

From: adil hasan <a.hasan@rl.ac.uk>

To: Michael Wan <mwan@sdsc.edu> ……

Template Files contain tags that are used to identify keyword/value pairs in a document

Sample Tags:

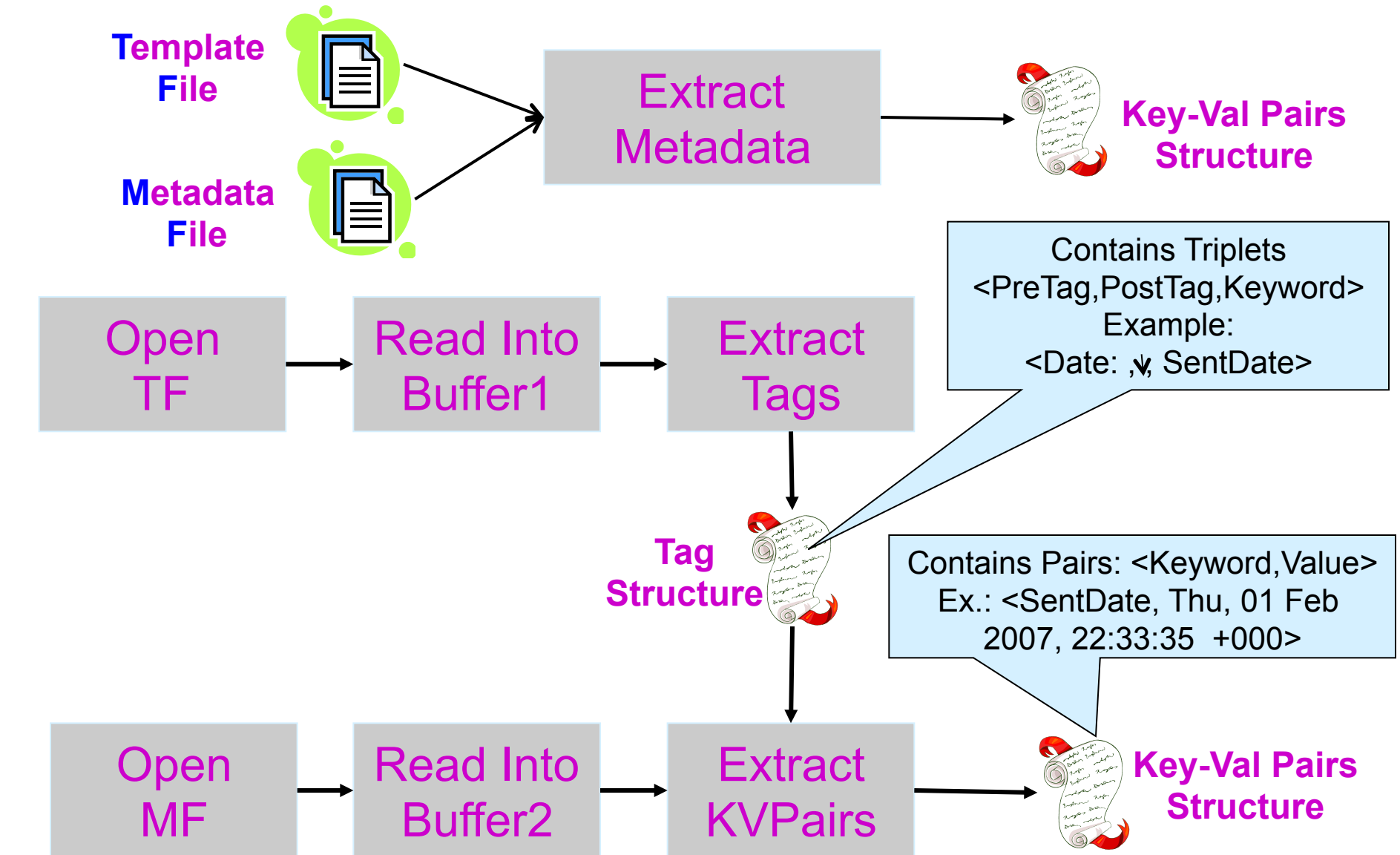
<PRETAG>Date: </PRETAG>SentDate<POSTTAG></POSTTAG>

<PRETAG>From: </PRETAG>Sender<POSTTAG></POSTTAG>

Meaning: Whatever is found between "Date :" and "↓" provides the "value" for the keyword: "SentDate"

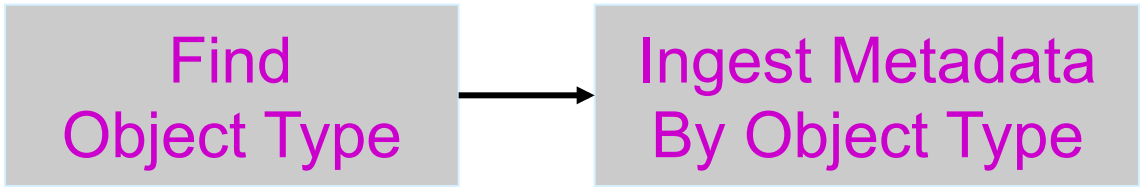
Metadata Files provide the actual metadata that need to be ingested.

Design: Extract Metadata



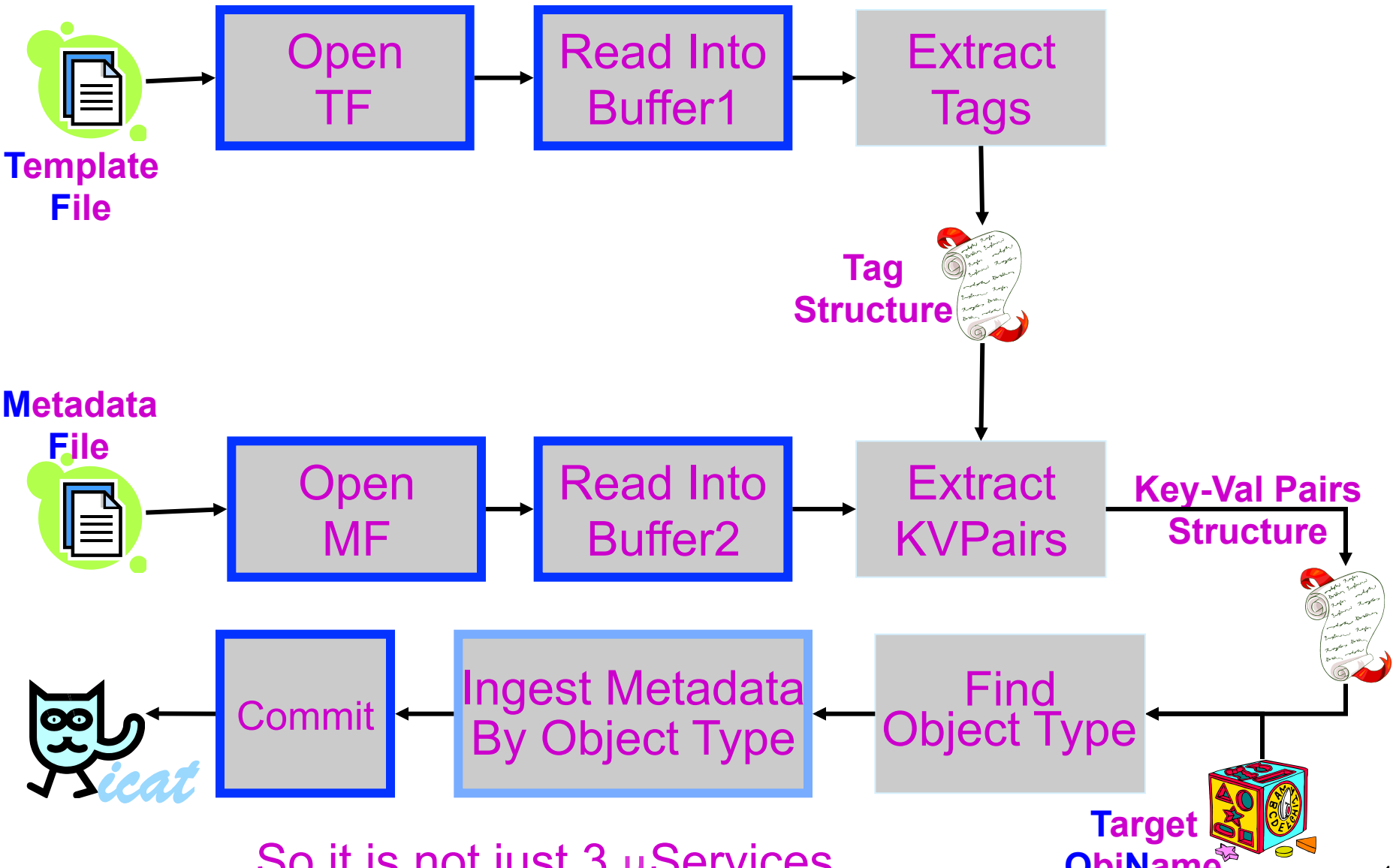
Design: Ingest Metadata

Instead of writing a μService for file-metadata ingestion only, I just designed a μService that can be applied to any iRODS object (data, collection, resource, user or token or metadata)



Question: How to convert a C-function into a μService?

Design: Epilogue



So it is not just 3 μServices

Implementation of μServices

Implementation: Prologue

Four Easy Steps:

- Define the signature of the μService
- Register the μService as an invokable method by the rule engine
- Create the μService
  - This may need other function calls (new and old)
- Describe the μService

We will look at two Examples:

- "FindObjectType" μService: msiGetObjType
- "Extract Tag" μService: msiReadMDTemplateIntoTagStruct

Implementation: Signature Definitions

All μServices have only two types of parameters

Params 1…(n-1) are of the type msParam_t

Param n is of the type ruleExecInfo_t

msParam_t is defined as:

```
typedef struct MsParam {
    char *label;
    char *type;
    void *inOutStruct;
    bytesBuf_t *inpOutBuf; } msParam_t;
```

ruleExecInfo_t is the "white board" used for passing session-oriented parameters that can be used by the Rule Engine and the micro-services.

```
int msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam,
                   ruleExecInfo_t *rei);
int msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam,
                   msiParam_t *tagParam, ruleExecInfo_t *rei);
```

Implementation:
Registration: "FindObjectType"

The Rule Engine only executes μServices that are enumerated in the list structure:

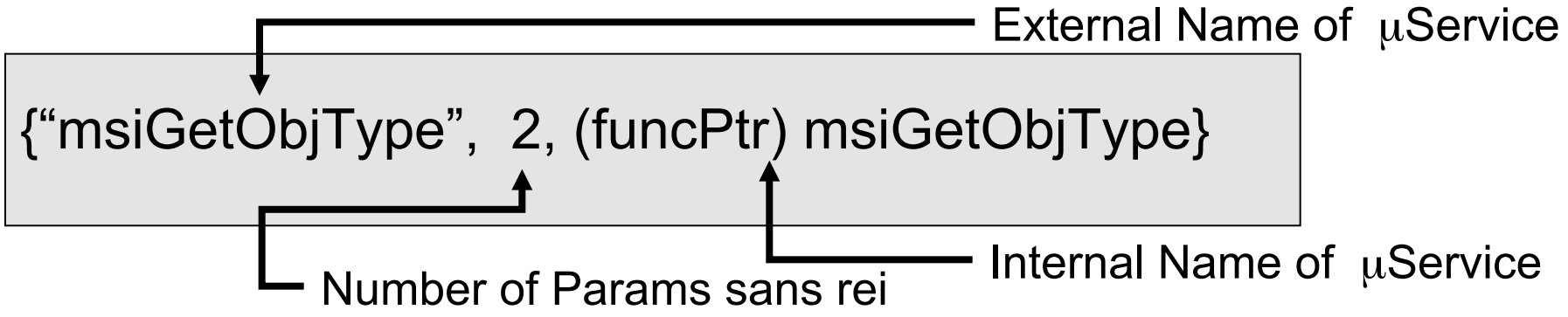
```
microsdef_t   MicrosTable[ ] = {} ;
```

(can be found in the file reAction.h or reAction.table)

For the μService

```
int msiGetObjType (msiParam_t *objParam, msiParam_t
    *typeParam,  ruleExecInfo_t *rei);
```

We add the following to the ActionTable



Implementation: Registration "ExtractTag"

For the μService

```
int msiReadMDTemplateIntoTagStruct (msiParam_t
  *bufParam, msiParam_t *tagParam, ruleExecInfo_t *rei);
```

We add the following to the ActionTable

```
{"msiReadMDTemplateIntoTagStruct",
        2, (funcPtr) msiReadMDTemplateIntoTagStruct}
```

Implementation: Creation "FindObjectType"

Here we program the code for the μService

```
int  msiGetObjType (msiParam_t  *objParam,
            msiParam_t *typeParam, ruleExecInfo_t *rei)
{
   char*   objName;
   char    objType[MAX_NAME_LEN];
   int       i;
   RE_TEST_MACRO("Looping back on msiGetObjType");
   if (strcmp(objParam->type, STR_MS_T) != 0)
          return(USER_PARAM_TYPE_ERROR);
   objName = (char *) objParam->inpOutStruct;
   i = getObjType (rei->rsComm, objName, objType);
   if (i < 0)   return(i);
   fillStrInMsParam(typeParam, objType);
   return(0);
}
```

Local Variables

Needed for Loop Back Testing of Workflow and Rules

Type Checking

Internal Function that is used for finding types of Objects. This routine makes calls to iCAT to find the type of the Object

Returning value being malloc'd into Param Structure and type-cast properly

Implementation:
Describe "FindObjectType"

We want to provide enough material for users to call the μService and for a program to identify it automatically in the future.

```
/**
 * \fn   GetObjType
 * \author  Arcot Rajasekar
 * \date   2007-02-01
 * \brief   this function finds from the iCat the type of a given object
 * \param[in]     objParam   is a msParam of type STR_MS_T
 * \param[out]  typeParam is a msParam of type STR_MS_T
 * \return integer
 * \retval 0 on success
 * \retval USER_PARAM_TYP_ERROR when input  param does not match
     the type
 * \retval from getObjType
 * \sa   getObjType
 * \post
 * \pre
 * \bug  no known bugs
**/
```

Implementation: Creation "Extract Tag"

Here we program the code for the μService

```
int  msiReadMDTemplateIntoTagStruct (msiParam_t  *bufParam,
                        msiParam_t *tagParam, ruleExecInfo_t *rei)
{
   bytesBuf_t   *tmplObjBuf;
   tagStruct_t  *tagValues;
    /*other internal variables are defined here */
 RE_TEST_MACRO("Looping back on msiReadMDTemplateIntoTagStruct");
  if (strcmp(bufParam->type, BUF_LEN_MS_T) != 0 || bufParam->inpOutBuf == NULL)
          return(USER_PARAM_TYPE_ERROR);
  tmplObjBuf = (bytesBuf_t   *) bufParam->inpOutBuf;
   tagValues = (tagStruct_t *) mallocAndZero(sizeof(tagStruct_t));
   /*   the main code segment that reads the buffer and identifies the
        the  <preTag, KeyWord, postTag>  triples goes in here. The triplets
        are store in tagValues.  */
   if (tagValues->len == 0)  { free(tagValues ); return(NO_VALUES_FOUND); }
  tagParam->inOutStruct = (void *) tagValues;
  tagParam->type = (char *) strdup(TagStruct_MS_T);
  return(0);
}
```

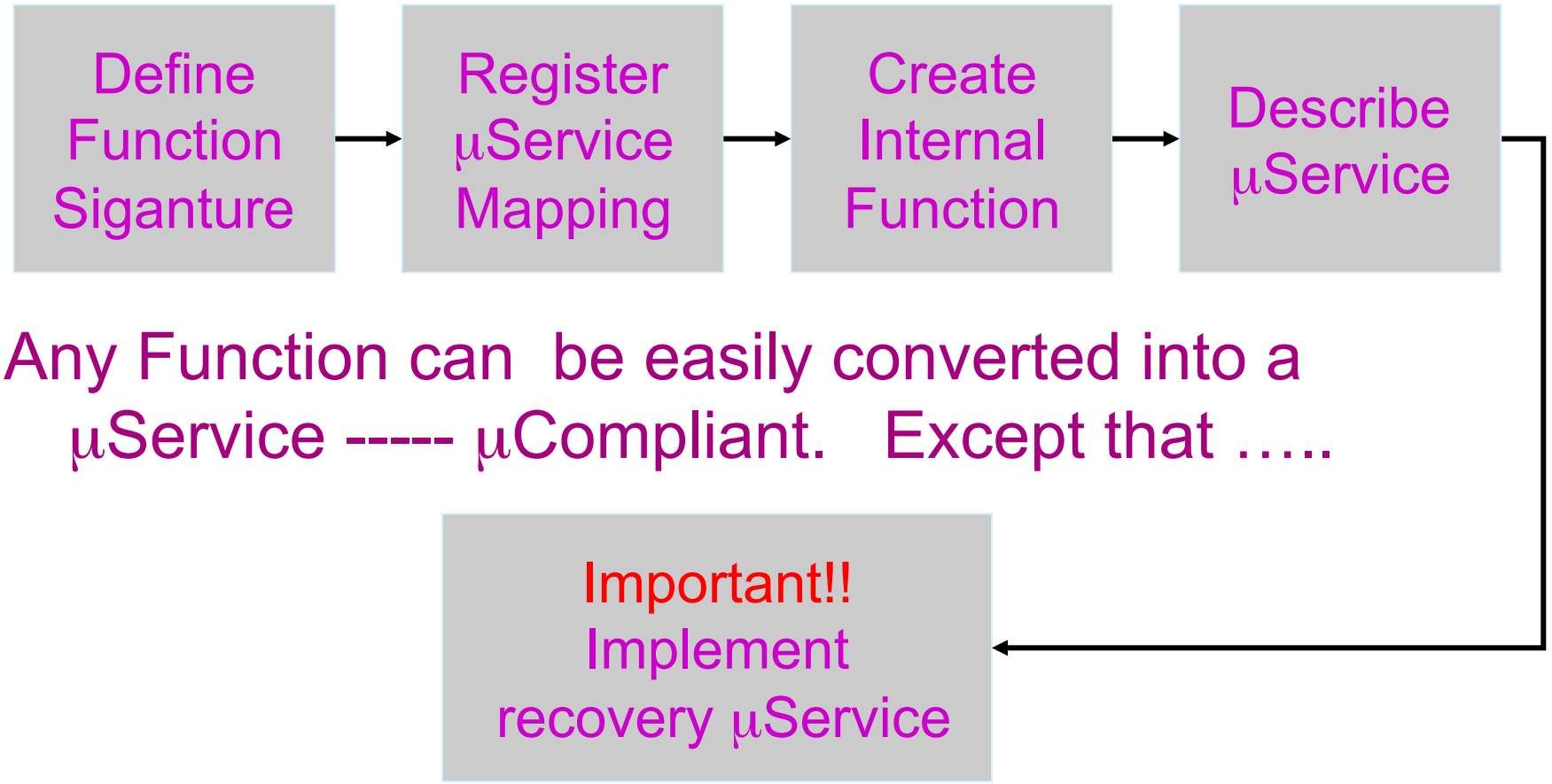
Implementation: Describe "Extract Tag"

We want to provide enough material for users to call the μService and for a program to identify it automatically in the future.

```
/**
 * \fn msiReadMDTemplateIntoTagStruct
 * \author  Arcot Rajasekar
 * \date   2007-02-01
 * \brief   this function parses a buffer containing a template-style file
 *  and stores the tags  in a tag structure.
 * \note  the template buffer should contain triplets be of the form
 *  <PRETAG>re1</PRETAG>kw<POSTTAG>re2</POSTTAG>
 *  re1 identifies the pre-string and re2 identifies the post-string,  and any value
 *  between re1 and re2 in a metadata buffer can be associated with keyword kw.
 * \param[in] bufParam   is a msParam of type BUF_MS_T
 * \param[out] tagParam is a msParam of type TagStruct_MS_T
 * \return integer
 * \retval 0 on success
 * \retval USER_PARAM_TYP_ERROR when input  param don't match the type
 * \retval INVALID_REGEXP if the tags are not correct
 * \retval NO_VALUES_FOUND if there are no tags identified
 * \retval from addTagStruct
 * \sa addTagStruct
 * \post
 * \pre
 * \bug  no known bugs
**/
```

Implementation: Epilogue

RECAP:



Any Function can be easily converted into a μService ----- μCompliant. Except that …..

Testing of μServices

Testing: Prologue

Server-side:

Create a rule out of the workflow, or

Add the μService to an existing rule

Client-side:

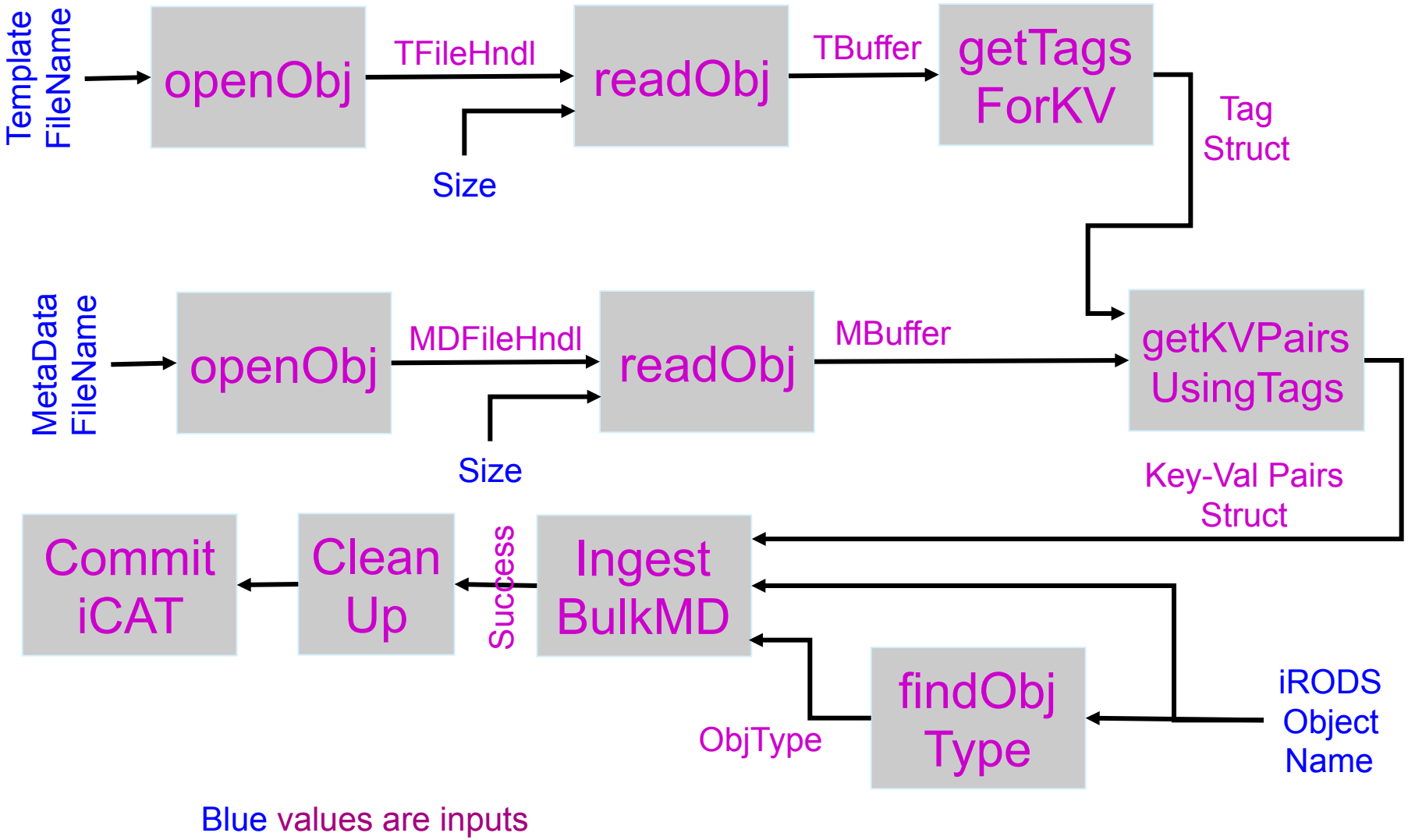
Test the rule using the ***irule*** command

Testing: Micros

msiDataObjOpen opens a iRODS File	openObj
msiDataObjRead reads an open iRODS File	readObj
msiReadMDTemplateIntoTagStruct reads Tag Info into Struct	getTagsForKV
msiExtractTemplateMDFromBuf gets MD using Tag Struct	getKVPairsUsingTags
msiGetObjType finds type of object	findObjType
msiAssociateKeyValuePairsToObj ingests extracted metadata	ingestBulkMD
msiCommit commit transaction in iCAT	commitIcat

External Aliases Help Application Developers and Users

Testing: Workflow Diagram



Testing: CommandLine WorkFlow

Pretty Printed Listing of File "ruleInp5"

```
mDExtract ||  openObj( *A, *T_FD)##getSizeData(*A,*S)##
              readObj( *T_FD, *S, *R1_BUF)##
              getTagsForKV( *R1_BUF, *TSP)##
              openObj( *B, *M_FD)##
              readObj( *M_FD, 10000, *R2_BUF)##
              getKVPairsUsingTags( *R2_BUF, *TSP, *KVP)##
              findObjType( *C, *OTYP)##
              ingestBulkMD( *KVP, *C, *OTYP)##
              closeObj(*T_FD,*J1)##closeObj(*M_FD,*J2)##
              commitIcat
*A=/tempZone/home/rods/Templates/mdTemplate1.txt%
*B=/tempZone/home/rods/test1.email%
*C=/tempZone/home/rods/test2.email
*R1_BUF%*TSP%*R2_BUF%*KVP%*A%*B%*C%*OTYP
```

WorkFlow Rule

Inputs

PrintOuts

How to run it: irule –v –F ruleInp5

Testing: Making a Rule

The rule is very similar to the workflow we had seen in the previous slide.

No Conditions are here

```
mDExtract(*A,*B,*C) ||  openObj( *A, *T_FD)##
                        readObj( *T_FD, 10000, *R1_BUF)##
                        getTagsForKV( *R1_BUF, *TSP)##
                        openObj( *B, *M_FD)##
                        readObj( *M_FD, 10000, *R2_BUF)##
                        getKVPairsUsingTags( *R2_BUF, *TSP, *KVP)##
                        findObjType( *C, *OTYP)##
                        ingestBulkMD( *KVP, *C, *OTYP)##
                        closeObj(*T_FD,*J1)##closeObj(*M_FD,*J2)##
                        commitIcat
```

Recovery Section

```
|closeObj(*T_FD)##nop##
 recover_ getTagsForKV( *R1_BUF, *TSP)##
 closeObj(*M_FD)## nop##
 recover_getKVPairsUsingTags( *R2_BUF, *TSP, *KVP)##
 nop##
 recover_ingestBulkMD( *KVP, *C, *OTYP)##
 nop##nop##rollbackIcat
```

Delaying a μService

One can delay the execution of any μService either in the irule execution or in a rule at the server side.

Example:

The μService msiSysReplDataObj(*R) replicates an existing iRODS file.

In order to delay this, one can use:

```
delayExec(<PLUSET>015m</PLUSET>, msiSysReplDataObj( tgReplResc ),nop)
```

In a rule this might be used as follows:

```
acPostProcessForPut |   $objPath like /tmpZone/home/tg/*  |
  delayExec((<PLUSET>015m</PLUSET>, msiSysReplDataObj( tgReplResc  ), nop)
       |  nop
acPostProcessForPut |   $objPath like /tmpZone/home/nvo/*  |
       msiSysReplDataObj( nvoReplResc )  |  nop
acPostProcessForPut |  | nop | nop
```

Recap: How to build μServices

Create the μService

- Create the micro-service function as needed.

```
int myPetProc(char *in1, int in2,
                    char *out1, int *out2)
{
    ... my favorite code ...
}
```

Create the μService Interface (msi)

- Create the micro-service interface.

```
int   msiMyPetProc(msParam_t *mPin1, msParam_t *mPin2,
          msParam_t *mPout1, msParam_t *mPout2,
          ruleExecInfo_t *rei)
{
    char   *in1,   *out1;
    int     i,  in2,   out2;
    RE_TEST_MACRO (" Calling myPetProc")
    /* the above line is needed for loop back testing using irule -i option */
    if (in1 = parseMspForStr (mPin1) == NULL) return(USER_PARAM_TYPE_ERR);
    if (in2 = parseMspForPosInt(mPin2)   < 0) return (in2);
    i = myPetProc(in1, in2, out1, &out2);
    fillIntInMsParam (mPout2,   out2);
    fillStrInMsParam(mPout1, out1);
    return(i);
 }
```

Register the μService Interface (msi)

- Add the signature

  int msiMyPetProc(msParam_t *mPin1, msParam_t *mPin2,
  msParam_t *mPout1, msParam_t *mPout2,
  ruleExecInfo_t *rei)

  In the file: reAction.header

- Make the micro-service interface visible to the rule engine by adding:

  {"msiMyPetProc",4, (funcPtr) msiMyPetProc},

  In the File: reAction.table

NOTE: Adding a μService to a module is quite different

Demonstration & Conclusion

Creating a Module

- Create a module to extend iRODS

Design of μServices for achieving a goal

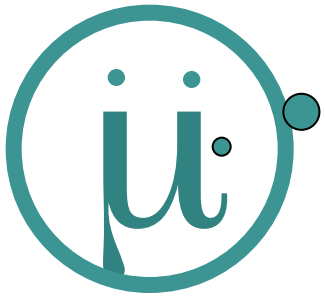
- Extraction &Ingestion of template-identified metadata

Implementation of μServices

- How each module is coded to be μ-compliant

Testing of μServices

- From the command line & as a rule.
- A demo of all the services
  as a workflow



Hope you Enjoyed it !
Any Questions !!