CYCLADES IST-2000-25456
An Open Collaborative Virtual Archive Environment

System Testing Report D4.2.1

Delivery Type: R
Number: D4.2.1
Contractual Date of Delivery: month 24
Actual Date of Delivery: April 22, 2003
Task: WP4

Name of Responsible: Tom Gross

Fraunhofer FIT
Schloss Birlinghoven
53754 St. Augustin
Germany

E-Mail: tom.gross@fit.fraunhofer.de

Contributors

CNR-IEI:	Henri Avancini, Leonardo Candela, M. Elena Renda, Umberto Straccia
University of Dortmund:	Gudrun Fischer, Sascha Kriewel, Ting Li, Saadia Malik
FORTH:	Nikos Papadopoulos, Dimitris Plexousakis
FhG-FIT:	Tom Gross, Thomas Kreifelts, Wido Wirsam, Silvio Caizzi

Abstract: This report provides detailed information on the plan and results of the tests of the individual components and of the integrated Cyclades system.

Contents

Chapter 1

Introduction

1.1 Systems tested

The present report documents the testing of the CYCLADES Deliverables D3.1.1 - D3.7.1, i. e. the Working Prototypes of the CYCLADES services, and the testing of Deliverable D4.1.1, the Open Archives Working Service Environment which consists of the integration of the service components into one environment.

The goal of CYCLADES is to provide an integrated environment for scholars and groups of scholars that want to use, in a highly personalized and flexible way, *open archives*, i. e. electronic archives of documents compliant with the Open Archives Initiative[1] (OAI) standard.

The CYCLADES system consists of the following services:

- Collaborative Work Service
- Search and Browse Service
- Access Service
- Collection Service
- Filtering and Recommendation Service
- Mediator Service

The Collaborative Work Service provides a folder-based environment for managing metadata records, queries, external documents, and annotations. Furthermore, it supports collaboration between CYCLADES users by way of folder-sharing in communities and projects. One component of this service is the Rating Management Service, which manages ratings.

The Search and Browse Service supports the activity of searching records from the various collections, of formulating and reusing queries, and browsing schemas, attribute values, and metadata records.

The Access Service is in charge of interfacing with the underlying metadata archives. Only archives adhering to the Open Archives specification will be accounted for.

The Collection Service manages collections, thus allowing a partitioning of the information space according to the users' interests, and making the individual archives transparent to the user.

The Filtering and Recommendation Service provides personalized filtering of queries and query results, provides recommendations of records, collections, users, and communities deemed relevant to the user's interests.

[1] http://www.openarchives.org

The Mediator Service acts as a registry for the other services and provides security, i. e. it checks if a user is entitled to use the system, and ensures that the other services are only called after proper authentication.

The services of the CYCLADES system communicate via HTTP, using XML-RPC [2]. XML-RPC is a simple protocol for implementing cross-platform, distributed applications. As its name suggests, communication between the distributed applications is done via remote procedure calls. The XML-RPC protocol is based on Internet standards: method calls and responses are transmitted using HTTP, and the bodies of the calls and responses are encoded in XML.

Most of the services (i. e. the Collaborative Work Service, the Search and Browse Service, the Access Service (for archive management), and the Collection Service (for collection management)) provide their own user interfaces. The Mediator Service itself provides the registration and login interface, and a system administration interface (for assigning access rights, etc.). Additionally, the Mediator Service integrates the user interfaces of the other services, and makes sure that those services and their interfaces are called only for authorized users, and only via the Mediator Service.

1.2 Test strategy

Following the system architecture, the CYCLADES test strategy has two phases:

- component testing
- integrated system testing

The result of component testing are debugged stand-alone service components, the result of integrated system testing is a debugged version of the complete CYCLADES environment.

1.2.1 Component testing

The component testing phase was directly based on the results of WP3, the working prototypes of each CYCLADES service, which form the single components of the CYCLADES system. During this testing phase, the CYCLADES service components were tested in a stand-alone manner, i. e. without communication with other components.

The API of each service component was tested on conformance to the specification. An up-to-date API specification of each service component is also part of this report. The tests were based on a number of test data per API method. Attention was also paid to robustness, i. e. the ability to deal with "wrong" input data. Partners who developed a component carried out a comprehensive test of their own API. Additionally, developers of components which call another component, also tested the methods they call.

For components that have a graphical user interface (GUI), the component test included a test of this GUI. In general, the GUI tests were carried out by the developers of a component. The GUI tests ensured that the use cases which have been described in the "Process flow" sections of Deliverable D3.0.1 work according to specification.

The components were tested running on a developing partner's host, the testing was carried over the Internet via HTTP, either from an XML-RPC client for the API or a Web browser for the GUI.

Errors that were detected during the component tests were reported back to the developers of the component who in turn took care of fixing the bugs. At the end of the component testing phase, new versions of the components were generated where necessary. The debugged service components served as basis for the system integration.

[2]http://www.xmlrpc.com/spec

1.2.2 Integrated system testing

After the Cyclades system services had been successfully tested, the components were integrated into a first version of the Cyclades working service environment. This was done by having the components communicate using the inter-service communication protocol XML-RPC making use of the diverse service APIs, and by directly invoking the graphical user interfaces of other components at the integrated Cyclades graphical user interface. The single service components resided on a developing partner's host.

1.2.3 Test report details

Every test of a single component or the integrated system is documented in a test report. Test reports include the following data:

- Name and email address of the tester
- Date and time when the test was performed
- Scope of test, e. g. for component API testing the name and version of the service component and the part of the API that has been tested
- Test environment including operating system and type of test client, e. g. programming language and XML-RPC library used for component API testing, or Web browser used for GUI testing
- Test plan, i. e. the list of test cases giving method/use case and input parameters used
- Test log listing the outcome of the test case executions (manual testing or automated testing via scripts)
- Summary of test results
- Recommendation for action when errors had been encountered

Where detailed test plans or test logs were too substantial to be included in this report, an excerpt and the location of the complete data is given.

Chapter 2

AS Component Test

2.1 Introduction

The Access Service (AS) harvests and indexes metadata records from open archives and makes them available for other CYCLADES services. For this purpose, it provides

- metadata harvesting and indexing
- persistent storage of metadata records
- management of archive information

The AS API consists of 17 methods. For archive management, the AS provides a GUI which supports the use cases described in sections 2.2.2 of Deliverable D3.0.1, i. e. registering an archive, editing archive information, and deleting an archive.

2.2 AS API test

2.2.1 AS API specification

The Access Service provides an API to the other CYCLADES services, as well as to its own GUI. The methods of this API may be called using the inter-service communication protocol XML-RPC. For every method also the services are listed that call this method according to the service interaction as specified in Deliverable D3.0.1.

public:

- **Method:** getId
  Signature: String getId()
  Description: this method returns the ID of the service
  Parameters:
  Output: Access Service ID
  Calling services: any

- **Method:** getArchives
  Signature: Archive* getArchives()
  Description: this method exports the registered archives
  Parameters:

Output: list of Archive
Calling service: SBS

- **Method:** getIndexedTermsAndWeights
  Signature: (term,weight)* getIndexedTermsAndWeights(recordId*,maxTermNo)
  Description: this method exports the indexed terms and their respective weights
  Parameters:

Input:	list of recordId:	the list of ids of the records for which the indexed terms and weights are to be determined
	maxTermNo:	the maximum number of terms to be returned (-1 for all terms)

  Output: list of pairs (term, weight)
  Calling service: FRS

- **Method:** search
  Signature: (Record,(term,weight)*)* search(query,maxRecordNo,maxTermNo,timeStamp)
  Description: this method determines the records corresponding to *query* and returns at most *maxRecordNo* records gathered after the time specified by *timeStamp*, and for each record, *maxTermNo* indexed terms with their weights.
  Parameters:

Input:	query:	the query string
	maxRecordNo:	the maximum number of records to be returned
	maxTermNo:	the maximum number of terms to be returned with each record
	timeStamp:	a timestamp specifying a date and time, only records gathered after this time will be considered (formats YYYY-MM-DD hh:mm:ss and YYYYY-MM-DD)

  Output: list of records, each with a list of pairs (term, weight) associated
  Calling services: CS, FRS, SBS

- **Method:** getArchiveDescription
  Signature: (oaiId,textualDescription,keyword*,schema*,language*,temporalCoverage) getArchiveDescription(oaiId)
  Description: this method returns a complete description of the archive specified by *oaiId*
  Parameters: Input: oaiId: the id of the archive

Output:	oaiId:	the id of the archive (string)
	textualDescription:	a string describing the archive
	list of keywords:	a list of strings that describe the archive's topic
	list of schemas:	a list of schema names (strings) of the metadata schemas available in the archive
	list of language:	a list of languages (strings) available in the archive
	temporalCoverage:	a list of pairs (fromYear,toYear) describing the intervals of time covered by the archive

  Calling services: CS, SBS

- **Method:** getSchemasForArchives
  Signature: Schema* getSchemasForArchives(archiveId*)
  Description: this method exports a list of all schemas that the specified archives supply (if no archive ids are specified, then all metadata schemas of all archives are listed)
  Parameters:
  Input: list of archiveId: the list of archive ids (strings)
  Output: a list of Schema objects
  Calling services: CS, SBS

- **Method:** getAttributes
  Signature: Attributee* getAttributes(schemaName)
  Description: this method returns the list of attributes that the Schema *schemaName* contains
  Parameters:

Input:	schemaName:	the name of the metadata schema

  Output: a list of Attribute objects
  Calling services: CS, SBS

- **Method:** getAttributeValues
  Signature: value* getAttributeValues(archiveId*,schemaName,attributeName,maxNo)
  Description: this method returns a list of *maxNo* attribute values from the archives specified, and for the given *schemaName* and *attributeName*
  Parameters:

Input:	list of archiveId:	the list ids of the archives (strings)
	schemaName:	the name of the metadata schema
	attributeName:	the name of the metadata attribute
	maxNo:	the maximum number of values to be returned

  Output: a list of values, their type according to the type of the attribute
  Calling service: SBS

- **Method:** getRecords
  Signature: Record* getRecords(recordId*)
  Description: this method returns the full records for the given record ids
  Parameters:

Input:	list of recordId:	the list ids of the requested records

  Output: a list of records
  Calling services: CWS, FRS, SBS

- **Method:** deleteUser
  Signature: void deleteUser(userId)
  Description: this method deletes a user from the AS database by changing the ownership of her archives to 'system'
  Parameters:

Input:	userId:	a user identifier.

  Calling service: MS

service internal:

These methods are called by the AS GUI, they are not available to other services.

- **Method:** registerArchive
  Signature: Archive registerArchive(url,owner)
  Description: creates a new archive object and collects initial data from the data provider at *url*, and remembers the user *owner* as the owner of this archive (Caution: the archive information is not yet saved to persistent storage)
  Parameters:

Input:	url:	the URL of the new archive's data provider
	owner:	the ID of the user who registers this archive

  Output: a new (not yet persistent) Archive object

- **Method:** getArchive
  Signature: Archive getArchive(archiveId,owner)
  Description: gets the archive with ID *archiveId*, and, if the owner is *owner*, loads the archive into memory for later changes
  Parameters:

Input:	archiveId:	the ID of the archive
	owner:	the ID of the user who owns this archive

  Output: an Archive object

- **Method:** saveArchive
  Signature: void saveArchive(Archive)
  Description: saves the Archive to persistent storage
  Parameters:

Input:	Archive:	an Archive object

- **Method:** deleteArchive
  Signature: void deleteArchive(archiveId,owner)
  Description: deletes the archive with the ID *archiveId*, if it is owned by the user *owner*
  Parameters:

Input:	archiveId:	the ID of the archive
	owner:	the ID of the user who wants to delete this archive

- **Method:** getArchivesForUser
  Signature: Archive* getArchivesForUser(owner)
  Description: gets the archive that are owned by *owner*
  Parameters:

Input:	owner:	the ID of the user

  Output: a list of Archive objects

- **Method:** forgetTempForUser
  Signature: void forgetTempForUser(owner)
  Description: clears all data for user *owner* from memory (temporary archive information that the user has not saved explicitly)
  Parameters:

Input:	owner:	the ID of the user

- **Method:** getMetadataAttributeTerms
  Signature: (term,weight)* getMetadataAttributeTerms(oaiId,schema,attributeName,maxTerm)
  Description: this method returns for the archive with the ID *oaiId*, for the *schema* and the specified *attribute*, at most *maxTerm* indexed terms with their weights
  Parameters:

Input:	oaiId:	the id of the archive
	schema:	the name of the schema (string)
	attributeName:	the name of the metadata attribute (string)
	maxTerm:	the maximum number of terms to be returned

  Output: a list of pairs (term,weight)

The class definitions that were used in some API signatures (Archive, Schema, Attribute) and that translate to XML-RPC `<struct>`s are as follows:

- **Archive**

  - **id:** the unique ID of the archive, a string
  - **oaiId:** the ID of the archive in open archives, a string
  - **textualDescription:** a textual description of the archive, entered by the archive registrator (string)
  - **schemas:** a list of Schema objects, containing the metadata schemas that the archive exports
  - **url:** the main URL of the archive (string)
  - **owner:** the ID of the user that owns the archive (string)
  - **adminEmail:** the e-mail address of the person responsible for the archive, extracted from the Identify record (string, optional)
  - **repositoryName:** the name of the archive, extracted from the Identify record (string, optional)
  - **identifyRecord:** the Identify record of the archive (string, optional)
  - **mirrors:** a list of further URLs for the archive (list of strings)
  - **languages:** a list of languages that the archive data
  - **temporalCoverage:** a list of pairs (year,year) that specify the temporal coverage of the archive, the archive registrator enters this data during registration
  - **keywords:** a list of keywords describing the archive content, entered by the archive registrator (list of strings)

- Schema
  - **id:** the unique ID of the schema (string)
  - **name:** the name of the schema (string)
  - **url:** the URL of a DTD or namespace for the schema (string)
  - **attributes:** a list of the attributes of the schema, each attribute being an object of class Attribute

- Attribute
  - **id:** the unique ID of the attribute (string)
  - **name:** the name of the attribute (string)
  - **type:** the abstract datatype of the attribute (string)
  - **predicates:** a list of the predicates available for this attribute, each predicate being an object of class Predicate

- Predicate
  - **id:** the unique ID of the predicate (string)
  - **name:** the name of the predicate (string)
  - **description:** a free text description of the predicate (string)

The class Record is described in the CWS chapter (3).

2.2.2 AS API test

In the following we list the API methods by calling services, also indicating test responsibilities for the methods listed.

- **MS** (FORTH)
  - deleteUser
- **FRS** (CNR)
  - getRecords
  - search
  - getIndexedTermsAndWeights
- **CS** (CNR)
  - getId
  - getArchiveDescription
  - getSchemasForArchives
  - getAttributes
  - getAttributeValues
- **SBS** (UNIDO/UNIDUE)
  - getArchives
- **internal** (UNIDO/UNIDUE)
  - registerArchive
  - getArchive
  - saveArchive
  - deleteArchive
  - getArchivesForUser
  - forgetTempForUser
  - getMetadataAttributeTerms

AS API test (UNIDO/UNIDUE)

- **Tester:** Saadia Malik (malik@is.informatik.uni-duisburg.de)
- **Test date:** 6 March 2003
- **Scope of test:**
  Testing CWS API methods called from AS
- **Test environment:**
  Test client was a Java[1] application running on Debian Linux, the XML-RPC implementation of Apache XML Project.

[1]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)

- **Test plan:**

  Each method was called first with valid parameters, then with invalid parameters of the right type, then with a wrong number of parameters, and finally, with a wrong parameter type.

  With the valid method calls, the semantic result was checked:

  – *getId* did return the service ID
  – *getArchives* the archives known to the AC were returned
  – *getArchiveDescription* returned Archive Description including textual Description,list of keywords, list of schemas,list of language, temporal coverage
  – *getSchemasForArchive* returned the schemas that the specfied archives supply, or all metadata schemas, according to input
  – *getAttribute* gives attribute list of the specified archive
  – *getAttributeValues* returns correct Attribute values of specified archiveId, schemaName,attributeName and maxNo
  – *deleteUser* deletes the specified User
  – *registerArchive* creates a new archive object and collects initial data from data provider at url, and remembers the user owner as the owner of this archive
  – *getArchive* returns correctly the Archive depending on the archive ID
  – *saveArchive* saves the specified archive correctly
  – *deleteArchive* deletes the archive if it is owned by the user specified
  – *getArchivesForUser* gives the list of Archives of the specified user
  – *forgetTempForUser* clears all the data for the user owner from memory
  – *getMetadataAttributeTerms* returns the correct index terms for the schema, attribute, and archive specified

  With the intentionally invalid calls, it was checked whether the method returned the appropriate error code and message.

- **Test log:**

  An excerpt of the last test log is shown below.

```
...

============================================
07.03.2003 15:55:15: Method.3=getArchiveDescription
07.03.2003 15:55:15: comment.3=No parameter is given
07.03.2003 15:55:15: flags.3=

07.03.2003 15:55:15: TestService: XML-RPC Fault #10001 -
org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(getArchiveDescription):
got 0, expected 1
07.03.2003 15:55:15: TestService: spent 66 millis for request

============================================
07.03.2003 15:55:15: Method.4=getArchiveDescription
07.03.2003 15:55:15: comment.4=Incorrect parameter type is given:
Integer instead of String
07.03.2003 15:55:15: flags.4=
07.03.2003 15:55:15: param.4.1.type=Integer
07.03.2003 15:55:15: param.4.1.value=10
07.03.2003 15:55:15: TestService: XML-RPC Fault #10002 -
org.apache.xmlrpc.XmlRpcException: Bad parameter type
07.03.(getArchiveDescription):
parameter #01 was java.lang.Integer, expected String
07.03.2003 15:55:15: TestService: spent 7 millis for request

==========================================
07.03.2003 15:55:15: Method.5=getArchiveDescription
07.03.2003 15:55:15: comment.5=existing archive id is given as paramater
07.03.2003 15:55:15: flags.5=

07.03.2003 15:55:15: param.5.1.type=String
07.03.2003 15:55:15: param.5.1.value=AIM25
07.03.2003 15:55:15: TestService: getArchiveDescription returned
[AC143_AIM25,no description available,[],[AC143_oai_dc],[],[]]
07.03.2003 15:55:15: TestService: spent 49 millis for request
...

07.03.2003 16:28:24: Method.12=getAttributes
07.03.2003 16:28:24: comment.12=Correct schema name is given
07.03.2003 16:28:24: flags.12=

07.03.2003 16:28:24: param.12.1.type=String
07.03.2003 16:28:24: param.12.1.value=oai_dc
07.03.2003 16:28:24: TestService: getAttributes returned
[[name=contributor,predicates=[
[name=$le$,description=less or equal,id=AC143_$le$],
[name=$lt$,description=less than,id=AC143_$lt$],
[name=$ge$,description=greater or equal,id=AC143_$ge$],
[name=$le$,description=greater than,id=AC143_$le$],
[name=$eq$,description=equal,id=AC143_$eq$],
[name=$ne$,description=not equal,id=AC143_$ne$],
[name=$cw$,description=contains,id=AC143_$cw$]],
type=text,id=AC143_contributor],
[name=coverage,predicates=[
[name=$le$,description=less or equal,id=AC143_$le$],
[name=$lt$,description=less than,id=AC143_$lt$],
[name=$ge$,description=greater or equal,id=AC143_$ge$],
[name=$le$,description=greater than,id=AC143_$le$],
[name=$eq$,description=equal,id=AC143_$eq$],
[name=$ne$,description=not equal,id=AC143_$ne$],
[name=$cw$,description=contains,id=AC143_$cw$]],
type=text,id=AC143_coverage],
[name=creator,predicates=[
...

==========================================
07.03.2003 16:28:25: Method.17=getAttributeValues
07.03.2003 16:28:25: comment.17=By specifying values
07.03.2003 16:28:25: flags.17=

07.03.2003 16:28:25: param.17.1.type=Vector
```

```
07.03.2003 16:28:25: param.17.1.1.type=String
07.03.2003 16:28:25: param.17.1.1.value=AIM25
07.03.2003 16:28:25: param.17.2.type=String
07.03.2003 16:28:25: param.17.2.value=oai_dc
07.03.2003 16:28:25: param.17.3.type=String
07.03.2003 16:28:25: param.17.3.value=description
07.03.2003 16:28:25: param.17.4.type=Integer
07.03.2003 16:28:25: param.17.4.value=1
07.03.2003 16:28:25: TestService: getAttributeValues returned
[A detailed and informative series of typescript letters,
1880-1901, from Francis Hall to his father relating to his life and
activities in South Africa (1880-1891) and East Africa (1892-1901).
It also includes typescript copies of four letters, 1883-1884,
from Francis's brother Albert Lambert Hall to their father,
07.03.miscellaneous letters received, and extracts from Hall's diary,
07.03.1893-1901.]
07.03.2003 16:28:25: TestService: spent 430 millis for request
...

07.03.2003 16:28:26: Method.23=registerArchive
07.03.2003 16:28:26: comment.23=correct URL and owner are given
07.03.2003 16:28:26: flags.23=

07.03.2003 16:28:26: param.23.1.type=String
07.03.2003 16:28:26: param.23.1.value=http://publications.uu.se/portal/OAI
07.03.2003 16:28:26: param.23.2.type=String
07.03.2003 16:28:26: param.23.2.value=system
07.03.2003 16:28:26: TestService: registerArchive returned
[url=http://publications.uu.se/portal/OAI,id=AC143_DiVA.se,
oaiId=DiVA.se,owner=system,mirrors=[],schemas=[[attributes=[],name=oai_dc,
url=http://www.openarchives.org/OAI/2.0/oai_dc.xsd,id=AC143_oai_dc]],
languages=[],keywords=[],identifyRecord=
...
```

- **Test summary:**

  All methods called from the AS were tested. All the methods worked according to the specification for all sets of correct input parameters. Correct calls did not produce errors, incorrect calls produced the appropriate error codes and messages.

AS API test (CNR)

- **Tester:** Henri Avancini (avancini@iei.pi.cnr.it)
- **Test date:** 24 February 2003
- **Scope of test:** Testing AS API[2] methods called from FRS.
- **Test environment:** Test client was a Java[3] application running on Linux (2.4.18-19.8.0), the XML-RPC implementation of Apache XML Project[4].

[2]`http://cyclades.cs.uni-dortmund.de:15200`
[3]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)
[4]http://xml.apache.org/xmlrpc

- **Test plan:** Test is divided in two stages. First, all user and folder identifiers are gathering from MS and CWS respectively (using *MS getUserIds* method and *CWS getFolders* method). Second, a user defined number of iterations are executed testing AS methods.

  Each iteration consist of:

  – Select both a random[5] valid user and folder identifier.

  – Call using AS methods: *getRecords, getIndexedTermsAndWeights, search.* Write down returned values. "Record identifiers" are gathered from CWS using *getRecords* method. "Maximun number of records" to be retrieved as well as "Maximun number of terms" are generated randomly as previously presented. "Query" is readed from a user defined file.

- **Test log:** An excerpt of test log is show bellow (file: frsCalledAPITest_detailed.log). The complete test log is available on-line[6].

```
Method called: frsCalledAPITest
Mon Feb 24 12:21:05 CET 2003
Initialize {AS, CWS, MS, RMS}Clients.

Method called: frsCalledAPITest
Mon Feb 24 12:21:06 CET 2003
Starting {AS, CWS, MS, RMS} API test..
Query file: query
Min.loop:   10

Method called: msclient.getUserIds ()
[CW665_7225, CW665_13620, CW665_6971, CW665_7131, CW665_7471, ...]
...
Loop: 1
Selected user: CW665_10244
Selected folder: CW665_12163

cwsclient.getRecords. Folder: CW665_12163
[[AC832_oai_dc_oai:caltechcstr:00000027, ],
[AC832_oai_dc_oai:caltechcstr:00000348, ],
[AC832_oai_dc_oai:caltechcstr:00000367, ],
[AC832_oai_dc_oai:RIACS:00000040, ]]
...
Subset of record ID selected:
[AC832_oai_dc_oai:caltechcstr:00000027, AC832_oai_dc_oai:caltechcstr:00000367,
AC832_oai_dc_oai:caltechcstr:00000348, AC832_oai_dc_oai:RIACS:00000040]

asclient.getRecords (recordIDs):
[{name=Logic from Programming Language Semantics, ...
, id=AC832_oai_dc_oai:caltechcstr:00000027}, {name=Synthesizing Dynamic ...
, id=AC832_oai_dc_oai:RIACS:00000040}]

asclient.getIndexedTermsAndWeights:
maxTermNo: 7. Terms & weights:
[[logic, 0.21334425], [finit, 0.178864], [program, 0.17145575],
[the, 0.17012675], [of, 0.16318175], [formula, 0.15183675], [ltl, 0.1495975]]
```

[5]http://java.sun.com/j2se/1.4/docs/api/java/util/Random.html
[6]http://project.iei.pi.cnr.it:8080/FRS/publicLogs/

```
asclient.search:
query:
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE query SYSTEM
       "http://localhost:15210/ac/query.dtd"><query schema="dc">
   <collection-query><condition weight="+" field="title">
   <field-condition predicate="$cw$" value="logic"/>
   </condition></collection-query></query>
maxRecordNo2AS: 5. maxTermNo: 3. timestamp: -9223372036854775808
Response:
[[{name=Synthesis and Biological Evaluation of bis ...
, id=AC832_oai_dc_oai:CPS:medichem/0204001}, ...
[{name=New ecological state of the ground after the war in Bosnia ...
, id=AC832_oai_dc_oai:CPS:envchem/0107002},

{AS, CWS, MS, RMS} API calls sumatory: 45
```

'FRSServer.log' contains FRS server side log, which correspond to the test period. The file 'frsCalledAPITest.log' contains console outputs from the client side and error messages. Lastly, 'startfrs.sh.log' file contains console outputs from the server side and error messages.

- **Test summary:** All methods called from FRS were tested. All methods tested worked according to specification for all set of correct input parameters. A total of 120 API calls were executed[7]. No errors occurred during the test run, except for the ones produced by adhoc bad parameters, e.g. call with a bad record identifier.

2.3 AS GUI test

The graphical user interface of the Access Service allows a user to register a new archive, to edit the information of an existing archive she owns, or to delete an existing archive she owns.

2.3.1 AS GUI description

The AS GUI is used for archive management. It presents a (possibly empty) list of archive objects owned by the user (figure 2.1).

Each archive object can be selected for multi-object actions. At the moment, the only possible multi-object action is deletion, however, more actions may be added in the future. Multi-object actions are available as buttons above the list of archives.

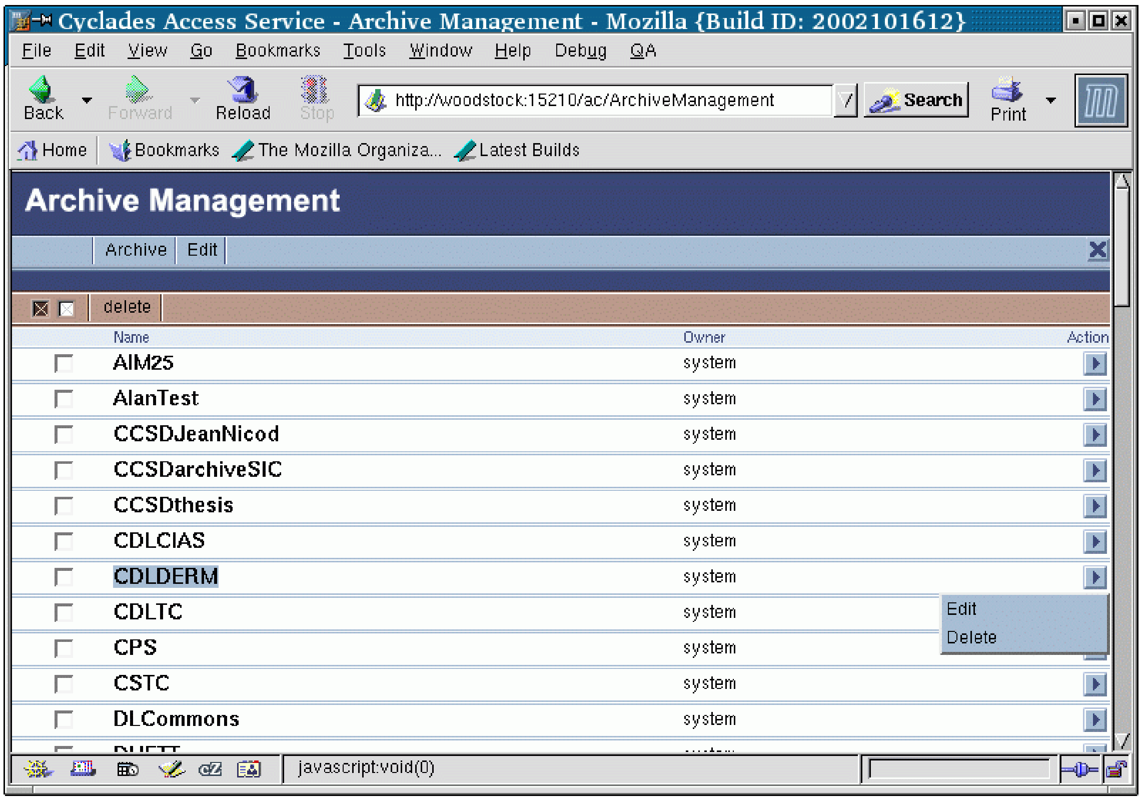
At the right end of each entry in the archives list, the user can open an actions menu containing the following options:

- *edit* edit the information stored for this archive, e. g. description, keywords, etc.
- *delete* delete this archive

The same funtionality can also be found in the *Edit* menu. Choosing *Edit* leads to a page with the archive information which can be edited and saved (or discarded) by the user.

The *Archive* menu contains only one option, i. e. *Register new*, for registering a new archive. After choosing this option, the user is presented first with an input form for the new archiv'e URL, and then with the archive information edit form.

[7]Total number of calls from FRS. As AS, CWS, MS, RMS test were made together, because of his dependencies, this number correspond to the sumatory of calls made from FRS to other CYCLADES services.


Cyclades Access Service - Archive Management - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
Back
Forward
Reload
Stop
http://woodstock:15210/ac/ArchiveManagement
Search
Print
Home
Bookmarks
The Mozilla Organiza...
Latest Builds
Archive Management
Archive
Edit
delete
Edit
Delete
javascript:void(0)

Name	Owner	Action
AIM25	system	
AlanTest	system	
CCSDJeanNicod	system	
CCSDarchiveSIC	system	
CCSDthesis	system	
CDLCIAS	system	
CDLDERM	system	
CDLTC	system	
CPS	system	
CSTC	system	
DLCommons	system	

Figure 2.1: AS Graphical User Interface

2.3.2 AS GUI test plan

The AS GUI test plan is based on three use cases for archive management listed in Deliverable D3.0.1. It is verified that the GUI elements invoke the desired action and use cases works. For this purpose at first each use case is divided into different steps and then test cases are derived.

- register a new archive
  - When register a new archive is invoked, the user is prompted for the archive URL.

    Test:
    1. Cancel Registration
    2. Register without giving URL information
    3. Try to register an invalid url
    4. Give incomplete url information
    5. Give valid information.
  - Then initial archive information is shown in editable form, where the following fields are editable: description, mirror, keyword, language. Registration can be completed using the 'Save' button, or cancelledn using the 'Cancel' button.

    Test:
    1. Cancel Registration
    2. Save the form without editing information.
    3. Give very long value for each field and save
    4. Give valid information and save the form

  - Success or failure message comes and in case of success message, the user is also informed that e-mail notification will be sent when the archive is searchable.

    Check:

    1. E-mail received
    2. Is the archive registerd

  - Register the same archive again.

- Edit Archive Information Edit archive information can be invoked either by clicking the Edit menu option, or by choosing one of the actions available in pop menu with each archive

  Check:

  1. Is edit archive interface invoked in both ways
  2. Is information edited saved
  3. On aborting is information remained unchanged

- Delete an Archive This operation can be performed for multiple archives and can be invoked by a menu option, or from the actions popup menu available with each archive.

  Test:

  1. Can it be invoked in either of above mentioned ways
  2. Can all the archives be selected with select All option
  3. Can all the archives be unselected with select None menu option.
  4. Is Delete All archives functionality ok
  5. Is archive deleted by using the pop menu option

2.3.3 AS GUI test results

- **Tester:** Saadia Malik malik@is.informatik.uni-duisburg.de
- **test date:**March 21,2003
- **scope of test:** Complete GUI of AS was tested according to the described test plan
- **Test environment:**The test client was Mozilla/5.0 running under Linux
- **Test log:** An excerpt of the test log is presented below:

```
Register Archive
----------------
Case:
        Enter to Archive Managment Environment
Action:
         Click Menu bar item  Archive Managment

Result:
        OK
Checked:
        Archive Managment windows comes contains listing of registered archives

-----------------------------------------------------------------------------
Case:
        Regsiter New Archive
Action
```

```
        clicked on Register -> New Archive

Result:
        OK
Checked:
        A screen comes asking for URL

-----------------------------------------------------------------------------
Case:
        Cancel the archive registration
Action:
        Cancel Button is clicked

Result:
        OK
Checked:
        takes to home page register archive
-----------------------------------------------------------------------------
Case:   Register none URL

Action
        Ok Button is clicked without giving URL
Result:
        OK
Checked:
         Msg comes 'Please specify valid URL'
-----------------------------------------------------------------------------
Case:
        Invalid URL is tried to register
Action:
       Register archive OK Button is clciked
       Parameter
       name:URL
       Value:347984

Result:
         OK
Checked:
        Proper message was comming 'Please specify the valid URL'
-----------------------------------------------------------------------------
Case:
        Register archive by giving incomplete URL
Action:
        Regsiter Archive OK button is clicked
        Parameters:
        Name:URL
        Value:http://memory.loc.gov/
Result:
        OK
Checked:
        Message comes 'Sorry could not get archive information'
-----------------------------------------------------------------------------
Case: Regsiter Archive with Valid URL
      Parameters:
```

```
      name:URL
      Value:http://www.bsz-bw.de/cgi-bin/oai20_send.pl
Result:
        OK

Checked:
        Proper message comes,archive is registered, an e-mail
        notification is received
-----------------------------------------------------------------------------
Case:
        Confirm the archive regsiteration
        Parameters
        mirror=,language=,keword=
Action:
        Register Archive -> click save button
Result
        OK

Checked:
        Archive is registered with given information.

-----------------------------------------------------------------------------
```

- **Test Summary:** All test case functions worked according to the use case descriptions and produced no errors.

Chapter 3

CWS Component Test

3.1 Introduction

The Collaborative Work Service (CWS) stores the private, community and project folders of the registered Cyclades users along with their contents which may be other folders, metadata records, queries and discussion forums. Project folders may contain also other material. The service supports

- folder and contents management,
- collaboration between users by way of folder sharing in communities and projects, discussion forums and mutual awareness,
- recommendations management.

The CWS has an API supplying 21 methods as well as a graphical user interface (GUI) with a rich set of functionality described in the 20 use cases of Deliverable D3.0.1, sections 3.2.1 - 3.2.20, where many use cases comprise several separate functions.

3.2 CWS API test

3.2.1 CWS API specification

The Collaborative Work Service provides an API to the other Cyclades services. The methods of this API may be called using the inter-service communication protocol XML-RPC. For every method also the services are listed that call this method according to the service interaction as specified in Deliverable D3.0.1.

- **Method:** createUser
  Signature: (userId, homeFolderId) createUser(name, password, emailAddress)
  Description: this method is invoked in order to create a new user within the CWS with the given name, password and email address.
  Parameters:

Input:	name:	a valid user name (unique, longer than 2 characters, no blanks or at-signs (@)).
	password:	a password.
	emailAddress:	an email address conforming to RFC 822.
Output:	userId:	the identifier of the newly created user.
	homeFolderId:	the identifier of the user's home folder.

Calling service: MS

- **Method:** updatePasswd
  Signature: void updatePasswd(name, password)
  Description: this method is invoked in order to set a new password for the user with the given name.
  Parameters:
  Input: name: a user name of an existing user.
  password: a password.
  Calling service: MS

- **Method:** getUserInfo
  Signature: UserInfo* getUserInfo(userIds)
  Description: this method is invoked to get information stored about users in the CWS (i. e. user identifier, full name, organization, phone and fax numbers, postal address, URL of homepage and image, and email address).
  Parameters:
  Input: userIds: a list of user identifiers
  Output: a list of UserInfo objects.
  Calling service: AS, CS, MS

- **Method:** deleteUser
  Signature: void deleteUser(userId)
  Description: this method deletes a user (including her recommendations, personal collection list, address book) and calls RMS' internal function anonymizeUserRatings, which sets the userId of all ratings of the respective user to 'anonymous'.
  Parameters:
  Input: userId: a user identifier.
  Calling service: MS

- **Method:** getFolders
  Signature: folderId* getFolders(userId)
  Description: this method may be invoked in order to get a list of identifiers of folders to which a specific user has access.
  Parameters:
  Input: userId: a user identifier.
  Output: a list of folder identifiers.
  Calling service: FRS

- **Method:** getName
  Signature: name getName(folderId)
  Description: this method may be invoked in order to get the name of a folder.
  Parameters:
  Input: folderId: a folder identifier.
  Output: a string containing the folder name.
  Calling service: FRS

- **Method:** getDescription
  Signature: description getDescription(folderId)
  Description: this method may be invoked in order to get the description of a folder.
  Parameters:

Input: folderId: a folder identifier.
Output: a string containing the folder description.
Calling service: FRS

- **Method:** getMembers
  Signature: userId* getMembers(folderId)
  Description: this method may be invoked in order to get the identifiers of the users who have access to a folder.
  Parameters:
  Input: folderId: a folder identifier.
  Output: a list of user identifiers.
  Calling service: FRS

- **Method:** getRecords
  Signature: recordId* getRecords(folderId, timestamp)
  Description: this method may be invoked in order to get the identifiers and classifier labels of the records that have been saved into, or moved to, a folder since a certain time.
  Parameters:
  Input: folderId: a folder identifier.
  timestamp: a point in time in UTC.
  Output: a list of record identifiers.
  Calling service: FRS

- **Method:** getQueries
  Signature: queries getQueries(folderId)
  Description: this method may be invoked in order to get the queries that are contained in a folder.
  Parameters:
  Input: folderId: a folder identifier.
  Output: a list of Query objects.
  Calling service: FRS

- **Method:** getParents
  Signature: folderId* getParents(folderId, userId)
  Description: this method may be invoked in order to get the identifiers of the folders that contain a given folder, and of which the user with the given identifier is a member. If the user identifier is an empty string, the identifiers of all folders are returned that contain the given folder.
  Parameters:
  Input: folderId: a folder identifier.
  userId: a user identifier.
  Output: a list of folder identifiers or a list containing an empty string, if the given folder is not contained in any folder.
  Calling service: FRS

- **Method:** getChildren
  Signature: folderId* getChildren(folderId)
  Description: this method may be invoked in order to get the identifiers of the folders that are contained in a given folder.
  Parameters:
  Input: folderId: a folder identifier.

Output: a list of folder identifiers.
Calling service: FRS

- **Method:** getCollections
  Signature: collectionId* getCollections(folderId)
  Description: this method may be invoked in order to get the identifiers of the collections that are associated to a folder.
  Parameters:
  Input: folderId: a folder identifier.
  Output: a list of collection identifiers.
  Calling service: FRS, SBS

- **Method:** getCommunity
  Signature: communityId getCommunity(folderId)
  Description: this method may be invoked in order to get the identifier of the community to which a folder belongs.
  Parameters:
  Input: folderId: a folder identifier.
  Output: a community identifier or an empty string if the folder does not belong to a community.
  Calling service: FRS

- **Method:** saveQuery
  Signature: void saveQuery(folderId, userId, query)
  Description: this method may be invoked in order to save a query in a folder on behalf of a user.
  Parameters:
  Input: folderId: a folder identifier.
  userId: a user identifier.
  query: a Query object.
  Calling service: SBS

- **Method:** saveResults
  Signature: void saveResults(folderId, userId, records)
  Description: this method may be invoked in order to save a list of records in a folder on behalf of a user.
  Parameters:
  Input: folderId: a folder identifier.
  userId: a user identifier.
  records: a list of Record objects.
  Calling service: SBS

- **Method:** saveRecommendedRecords
  Signature: boolean saveRecommendedRecords(folderId, recordIds)
  Description: this method may be invoked in order to save a list of recommended records for a folder.
  Parameters:
  Input: folderId: a folder identifier.
  recordIds: a list of recordIdentifiers.
  Output: false if record recommendations for this folder are not welcome, true otherwise.
  Calling service: FRS

- **Method:** saveRecommendedUsers
  Signature: boolean saveRecommendedUsers(folderId, userIds)
  Description: this method may be invoked in order to store a list of user recommendations for a folder.
  Parameters:

Input:	folderId:	a folder identifier.
	userIds:	a list of user identifiers.

  Output: false if user recommendations for this folder are not welcome, true otherwise.
  Calling service: FRS

- **Method:** saveRecommendedCommunities
  Signature: boolean saveRecommendedCommunities(folderId, communityIds)
  Description: this method may be invoked in order to store a list of community recommendations for a folder.
  Parameters:

Input:	folderId:	a folder identifier.
	communityIds:	a list of community identifiers.

  Output: false if community recommendations for this folder are not welcome, true otherwise.
  Calling service: FRS

- **Method:** saveRecommendedCollections
  Signature: boolean saveRecommendedCollections(folderId, collectionIds)
  Description: this method may be invoked in order to store a list of collection recommendations for a folder.
  Parameters:

Input:	folderId:	a folder identifier.
	collectionIds:	a list of collection identifiers.

  Output: false if collection recommendations for this folder are not welcome, true otherwise.
  Calling service: FRS

- **Method:** addModifyCollection
  Signature: void addModifyCollection(collectionId, collectionName, collectionDescription, parentCollectionId)
  Description: this method may be invoked in order to notify of the creation of a new, or the modification of an existing, collection.
  Parameters:

Input:	collectionId:	a collection identifier.
	collectionName:	the collection name.
	collectionDescription:	a description of the collection.
	parentCollectionId:	an identifier of the parent collection.

  Calling service: CS

- **Method:** deleteCollection
  Signature: void deleteCollection(collectionId)
  Description: this method may be invoked in order to notify of the deletion of a collection.
  Parameters:

Input:	collectionId:	a collection identifier.

  Calling service: CS

- **Method:** updatePersonalCollections
  Signature: void updatePersonalCollections(userId, collectionIds)

Description: this method may be invoked in order to notify of the update of a user's personal set of collections.
Parameters:

Input:	userId:	a user identifier.
	collectionIds:	a list of collection identifiers.

Calling service: CS

The class definitions that were used in some API signatures (Record, Query, UserInfo) and that translate to XML-RPC `<struct>`s are as follows:

- **Query**
  - **id:** the unique identifier of the query.
  - **name:** a string containing the name of the query. This attribute is optional, i. e. is only present if the user has given the query a name.
  - **queryString:** a string containing the actual query.
- **Record**
  - **id:** the unique identifier of the record.
  - **name:** a string containing the name of the record, e. g. the title of the document referenced by the record.
  - **metadata:** a string containing the metadata of the record coded in XML.
- **UserInfo**
  - **userId:** the unique identifier of the user.
  - **fullName:** a string containing the first name, (middle name), and last name.
  - **organization:** a string containing the name of the user's organization.
  - **workPhone:** a string containing the user's office phone number.
  - **workFax:** a string containing the user's office fax number.
  - **homePhone:** a string containing the user's home phone number.
  - **mobilePhone:** a string containing the user's mobile phone number.
  - **postalAddress:** a string containing the user's postal address.
  - **homePage:** a string containing the URL of the user's home page.
  - **imageUrl:** a string containing the URL of the user's picture on the WWW.
  - **mailAddress:** a string containing the user's email address.

3.2.2 CWS API test

In the following we list the API methods by calling service indicating test responsibilities for the methods listed.

- **MS** (FORTH)
  - createUser
  - updatePasswd
  - getUserInfo
  - deleteUser

- **FRS** (CNR)
  - getFolders
  - getName
  - getDescription
  - getMembers
  - getRecords
  - getQueries
  - getParents
  - getChildren
  - getCollections
  - getCommunity
  - saveRecommendedRecords
  - saveRecommendedUsers
  - saveRecommendedCommunities
  - saveRecommendedCollections
- **SBS** (UNIDO)
  - getCollections
  - saveQuery
  - saveResults
- **CS** (CNR)
  - getUserInfo
  - addModifyCollection
  - deleteCollection
  - updatePersonalCollections
- **AS** (UNIDO)
  - getUserInfo

In the following we summarize the results of the CWS API tests as conducted by FIT and the partners responsible for services calling the CWS API.

CWS API test (FIT)

- **Tester:** Wido Wirsam (wido.wirsam@fit.fraunhofer.de)
- **Test date:** 5 November, 2002 15:12 GMT
- **Scope of test:** Complete API of CWS v0.2[1] was tested via several Python scripts which tested every method of the API with a number of parameter sets that were read from a test data file.
- **Test environment:** Test client was a Python 2.2 script running on Windows XP making use of xmlrpclib module v. 0.9.9, the XML-RPC implementation of Secret Labs AB[2].

[1] http://cosidetti.gmd.de/rpc2/cyc_cws.cgi
[2] http://www.pythonware.com/products/xmlrpc/

- **Test plan:** The CWS XML-RPC test platform provides a tool to automatically create CWS-users. All API-methods that are provided by the CWS-service are performed on these users and their folders.

  The platform consists of the following python programs:

  – definitions.py This configuration file defines a set of values necessary for the creation of the test-users and the behaviour of the testrun.

  – TestFieldGenerator.py This script performs the generation of test-users. All relevant information about the generated users is stored in a log file.

  – logfile.py This class provides access to all information about the users created by the TestFieldGenerator.

  – main.py This script executes all XML-RPC calls provided by the CWS-services on the users generated by the TestFieldGenerator.

  The first step necessary to test the CWS API is to create some dummy users that populate the cyclades system. In the definitions.py file the number of users that will be created can be specified. The run of TestFieldGenerator.py uses the API method 'createUser()' to generate the specified number of users. 'createUser()' returns a unique user-ID and a folder-ID that are stored in a logfile. For each user a password is randomly generated. It is stored in the logfile as well. The generated logfile looks like this (to improve readability some spaces have been removed):

```
ID   userName      userPwd         userMail        userID    userFolder

 0   Karlvugz   cyhpjoagvj   wirsam@web.de   CW111_4882   CW111_4885
 1   Karlwlyv   prmoqmypxa   wirsam@web.de   CW111_4925   CW111_4928
 2   Karljiiu      sdysuff   wirsam@web.de   CW111_4968   CW111_4971
```

  The second step is to call all CWS API methods. Most of the methods require a user-Id or a folder-Id as parameters. All methods of the CWS API that require a user-Id and no folder-Id as parameter are called once for every user, with that specific user-Id as parameter. All methods that require a folder-Id as parameter are called once per folder of every user. Each call of every method and its results are logged the 'Access.txt' file. The 'Access.txt' file looks like this:

```
AccessLogfile created Tue Nov 05 15:12:57 2002

result = s.service.getFolders('CW111_11179')
['CW111_11200', 'CW111_11204', 'CW111_11211', 'CW111_11218']

result = s.service.getUserInfo(['CW111_11179'])
[{'fullName': 'Karlitpqr', 'userId': 'CW111_11179'}]

result = s.service.getUserInfoFromNames(['Karlitpqr'])
[{'fullName': 'Karlitpqr', 'userId': 'CW111_11179'}]

result = s.service.updatePersonalCollections('CW111_11179', testCollections)


result = s.service.getChildren('CW111_11200')
[]

result = s.service.getName('CW111_11200')
```

```
MyCYPrivateFolder

result = s.service.getDescription('CW111_11200')
MyDescr

result = s.service.getMembers('CW111_11200')
['CW111_11179']
```

This ist the roadmap to test the CWS-API with the python test-program:

- open the definitions.py file and modify the value of the parameter 'numMembersToCreate' to the number of users you want to be created.
- run the file TestFieldGenerator.py. Via the API method createUsers() the CWS system is called to create the specified amount of users. For every user one PrivateFolder, one CommunityRootFolder, one CommunityFolder and one ProjectFolder are created. The program produces a file called 'TFGLog*.txt' where * is the smalest number that does not already exists.
  - 'TFGLog*.txt' logs the username, userpassword, userEmail, CWS-userID and CWS-homeFolder of every user created.
- open the 'definitions.py' file again and modify the parameter 'testFieldLogFile' to the filename that just has been created by the run of TestFieldGenerator.py .
- run the file main.py. The program executes every method provided by the CWS- and the RMS-services and produces the following log files:
  - 'Access.txt' logs every command that is executed by main.py as well as the results the methods return.
  - 'Errors.txt' logs every error that occurs during the test run.
- check the log files.

- **Test run:** The CWS API test was performed on the server 'http://cosidetti.gmd.de' which runs the same version of CYCLADES CWS as the 'http://cyclades.gmd.de' server. The 'http://cosidetti.gmd.de' server is not connected to the other services so the CWS API functionality can be tested with no side effects on the other services. The test was performed with 20 computer generated users on November 5th 2002. Every user was generated by the CWS API method:

```
  createUser(name, password, emailAddress)
```

The username and password are automatically generated. For the emailAddress argument a valid eMail address is used. This is the resulting logfile 'TFGLog83.txt' :

```
ID    userName      userPwd         userMail        userID    userFolder

 0   Karlitpqr       zxtbqo    wirsam@web.de   CW111_11179   CW111_11182
 1     Karlmdx       kjooqh    wirsam@web.de   CW111_11222   CW111_11225
 2    Karlthbe      qvuxyun    wirsam@web.de   CW111_11265   CW111_11268
 3  Karllvpzij        cbahu    wirsam@web.de   CW111_11308   CW111_11311
 4     Karlasg       ozjhlt    wirsam@web.de   CW111_11351   CW111_11354
 5  Karlhkrvqf       bttfgc    wirsam@web.de   CW111_11394   CW111_11397
 6    Karlcjfb      vkgvvjn    wirsam@web.de   CW111_11437   CW111_11440
 7    Karluaje       ttkgna    wirsam@web.de   CW111_11480   CW111_11483
 8    Karlselx      bbfsovw    wirsam@web.de   CW111_11523   CW111_11526
 9     Karlngj        dfxdr    wirsam@web.de   CW111_11566   CW111_11569
```

10	Karlcrmi	gskib	wirsam@web.de	CW111_11609	CW111_11612
11	Karlmxdtii	eouvhmn	wirsam@web.de	CW111_11652	CW111_11655
12	Karlbavk	mccmoqa	wirsam@web.de	CW111_11695	CW111_11698
13	Karlxno	nwjlvbw	wirsam@web.de	CW111_11738	CW111_11741
14	Karlcldq	fkiri	wirsam@web.de	CW111_11781	CW111_11784
15	Karlfqk	izlkeo	wirsam@web.de	CW111_11824	CW111_11827
16	Karlbkw	acabwh	wirsam@web.de	CW111_11867	CW111_11870
17	Karlwxrf	yqxjn	wirsam@web.de	CW111_11910	CW111_11913
18	Karlwueige	hyyaodv	wirsam@web.de	CW111_11953	CW111_11956
19	Karloed	vrsqffj	wirsam@web.de	CW111_11996	CW111_11999

The file 'TFGLog83.txt' is available online[3]. For each user four folders were produced: one private folder, one project folder, one community root folder and one community folder. This was done by calling the CWS API method:

```
createFolder(userID, folderID, folderType)
```

In the test run all CWS API methods were called. The calls and the results were stored in a logfile called 'Access.txt'. In the logfile 'Errors.txt' error messages would have been logged if any had occured. Most methods take either a userID or a folderID as argument. Every API method that needs a userID is called once for every user. One example is the method 'getFolders(userID)'. It returns a list of folders that are owned by the user. In our testrun the method returned the four folders that previously were created. The results of these calls were stored during the testrun. Thereafter every method of the CWS API that needs a folderID as argument was called with each of the returned folderIDs for every user. In detail these were the following methods:
called once per user:

```
getFolders()
getUserInfo()
updatePersonalCollections()
```

called once per folder:

```
getName()
getDescription()
getMembers()
getRecords()
getQueries()
getParents()
getChildren()
getCollections()
getCommunity()
saveQuery()
saveResults()
saveRecommendedRecords()
saveRecommendedUsers()
saveRecommendedCommunities()
saveRecommendedCollections()
addModifyCollection()
deleteCollection()
```

[3]http://cyclades.gmd.de/publicLogs/CWS/TFGLog83.txt

the methods:

```
    updatePasswd()
    deleteUser()
```

have been sucessfully tested in the script 'cleanup.py' after the other tests have been completed.

All together 1440 API-Calls have been made.

This is an excerpt of the 'Access.txt' logfile. The complete file is available online[4].

```
AccessLogfile created Tue Nov 05 15:12:57 2002

result = s.service.getFolders('CW111_11179')
['CW111_11200', 'CW111_11204', 'CW111_11211', 'CW111_11218']

result = s.service.getUserInfo(['CW111_11179'])
[{'fullName': 'Karlitpqr', 'userId': 'CW111_11179'}]

result = s.service.getUserInfoFromNames(['Karlitpqr'])
[{'fullName': 'Karlitpqr', 'userId': 'CW111_11179'}]

result = s.service.updatePersonalCollections('CW111_11179', testCollections)


result = s.service.getChildren('CW111_11200')
[]

result = s.service.getName('CW111_11200')
MyCYPrivateFolder

result = s.service.getDescription('CW111_11200')
MyDescr

result = s.service.getMembers('CW111_11200')
['CW111_11179']

result = s.service.getRecords('CW111_11200', xmlrpclib.DateTime(time.time()
-30000000))
[['AC_00001', 'class label 1'], ['AC_00002', 'class label 231']]

result = s.service.getParents('CW111_11200', "")
['CW111_11182']

result = s.service.getCollections('CW111_11200')
['CO111_0005', 'CO111_0003']

result = s.service.getCommunity('CW111_11200')


result = s.service.saveQuery('CW111_11200', 'CW111_11179', query)
```

[4]http://cyclades.gmd.de/publicLogs/CWS/Access.txt

```
result = s.service.saveResults('CW111_11200', 'CW111_11179', results)


result = s.service.saveRecommendedRecords('CW111_11200', results)
<Boolean True at a69ed0>

result = s.service.saveRecommendedUsers('CW111_11200', testUsers)
<Boolean True at a69ed0>

result = s.service.saveRecommendedCommunities('CW111_11200', testCommunities)
<Boolean True at a69ed0>

result = s.service.saveRecommendedCollections('CW111_11200', testCollections)
<Boolean True at a69ed0>
```

During the complete testrun no errors have occured. This is the contens of the error logfile 'Errors.txt':

```
ErrorLogfile created Tue Nov 05 15:12:57 2002

ErrorLogfile closed Tue Nov 05 15:50:34 2002
```

- **Test summary:** All methods worked according to specification for all sets of input parameters. 20 computer generated users were created via the CWS API method 'createUser()'. On these users all available CWS API methods were executed. No Errors occured during the testrun. The method calls and the results produced by the CWS system were logged and made available online.

CWS API test (FORTH)

```
Logs on a CWS API Test
====================================
Date 21 November 2002
Author Nikos Papadopoulos
ICS-FORTH GREECE
====================================
Method: createUser
Parameters: testuser, testuser, testuser@ics.forth.gr
Result: [CW665_19457, CW665_19460]
====================================
Method: createUser
Parameters: testuser, testuser
Result: Bad number of parameters in createUser: expected 3, got 2
====================================
Method: updatePasswd
Parameters: testuser, testuser2
Result:
====================================
Method: updatePasswd
Parameters: testuser
Result: Bad number of parameters in updatePasswd: expected 2, got 1
====================================
Method: deleteUser
Parameters:  CW665_19457 (a correct userID, the one got from createUser before)
Result:
====================================
Method: deleteUser
Parameters:  CW665_29457 (a wrong userID)
Result: Bad user id(s): CW665_29457
====================================
```

CWS API test (CNR)

- **Tester:** Leonardo Candela (candela@iei.pi.cnr.it)
- **Test date:** 14 February 2003
- **Scope of test:** Testing CWS API v0.2[5] methods called from CS.
- **Test environment:** Test client was a Java program running on Windows 2000 making use of Apache XML-RPC v1.0 implementation[6].
- **Test plan:** The first step consists in calling the method `getUserInfo`. This method require as parameters some valid user identifiers that can be acquired via the method `getUserIds` of MS. Returned results can be checked using the CWS GUI.

  Second step consists in calling the method `updatePersonalCollections`. This method require as parameters a valid user identifier and a set of valid collections identifiers. Valid collections identifiers can be acquired via the method `listCollections` of CS. Results of method invocation are checked via the CWS GUI (e.g. trying to create a new private folder a set of collection can be selected among user's personal collections, if any).

  Third step consists in calling the method `addModifyCollection`. This method is used in order to notify the creation of a new, or the modification of an existing, collection and require as parameters a collection identifier, collection name, collection description and parent collection identifier. Results of method invocation are checked via the CWS GUI (e.g. trying to create a new private folder a set of collection can be selected among all collection, if no personal collections are selected).

  The last step consists in calling the method `deleteCollection`. This method is used in order to notify the deletion of a collection and require as parameter a collection identifier. Results of method invocation are checked via the CWS GUI (e.g. trying to create a new private folder a set of collection can be selected among all collection, if no personal collections are selected).
- **Test log:** An excerpt of test log is shown below. The complete test log is available on-line[7].

```
CWS run log - created Fri Feb 14 09:16:59 CET 2003
Method: getUserInfo
Parameters:
   CW665_7131
Result:
   [{... userId=CW665_7131, fullName=Leonardo Candela, ...
...
Method: updatePersonalCollections
```

[5]http://cyclades.gmd.de/cgi-bin/cyc_cws.cyc
[6]http://xml.apache.org/xmlrpc
[7]http://project.iei.pi.cnr.it:8080/CollectionService/publicLogs/CWStestAPI.log

```
Parameters:
   CW665_7131
Result:
   void
...
Method: addModifyCollection
Parameters:
   aaa
   bbb
   ccc
   ddd
Result:
   void
...
Method: deleteCollection
Parameters:
   aaa
Result:
   void
...
```

- **Test summary:** All methods tested worked according to specification for all sets of input parameters. No errors occurred during the test run.

CWS API test (UNIDUE)

- **Tester:** Gudrun Fischer (Gudrun.Fischer@uni-duisburg.de)
- **Test date:** 4 March 2003
- **Scope of test:** Testing CWS API methods called from AS and SBS.
- **Test environment:** Test client was a Java[8] application running on Debian Linux, the XML-RPC implementation of Apache XML Project.
- **Test plan:**

  Each method was called first with valid parameters, then with invalid parameters of the right type, then with a wrong number of parameters, and finally, with a wrong parameter type.

  With the valid method calls, the semantic result was checked:

  – *getUserInfo* did return the correct user data
  – *getCollections* returned exactly those collections contained in the folder
  – *getQueries* returned exactly those queries contained in the folder
  – *saveQuery* resulted in the saved queries appearing in the folder
  – *saveResults* resulted in the saved records appearing in the folder

  With the intentionally invalid calls, it was checked whether the method returned the appropriate error code and message.

- **Test log:** An excerpt of the last test log is shown below.

[8]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)

```
Mar 4, 2003 9:41:14 AM: Reading from tests_for_partners/cw/CW_CompleteRun.properties
Mar 4, 2003 9:41:14 AM: ServerURL=http://cyclades.gmd.de/cgi-bin/cyc_cws.cgi
Mar 4, 2003 9:41:14 AM: ServerName=service

===========================================
Mar 4, 2003 9:41:14 AM: Method.1=getUserInfo
Mar 4, 2003 9:41:14 AM: comment.1=valid parameters

Mar 4, 2003 9:41:14 AM: param.1.1.type=Vector
Mar 4, 2003 9:41:14 AM: param.1.1.1.type=String
Mar 4, 2003 9:41:14 AM: param.1.1.1.value=CW665_7853
Mar 4, 2003 9:41:15 AM: TestService: getUserInfo returned
[{userId=CW665_7853, fullName=gf, mailAddress=fischer@ls6.cs.uni-dortmund.de}]
Mar 4, 2003 9:41:15 AM: TestService: spent 595 millis for request

===========================================
Mar 4, 2003 9:41:15 AM: Method.2=getUserInfo
Mar 4, 2003 9:41:15 AM: comment.2=list of user IDs contains one invalid ID

Mar 4, 2003 9:41:15 AM: param.2.1.type=Vector
Mar 4, 2003 9:41:15 AM: param.2.1.1.type=String
Mar 4, 2003 9:41:15 AM: param.2.1.1.value=CW665_7853
Mar 4, 2003 9:41:15 AM: param.2.1.2.type=String
Mar 4, 2003 9:41:15 AM: param.2.1.2.value=bla
Mar 4, 2003 9:41:18 AM: TestService: XML-RPC Fault #10102 -
Bad user id(s): ['bla']
Mar 4, 2003 9:41:18 AM: TestService: spent 3409 millis for request

===========================================
Mar 4, 2003 9:41:18 AM: Method.3=getUserInfo
Mar 4, 2003 9:41:18 AM: comment.3=missing parameter

Mar 4, 2003 9:41:22 AM: TestService: XML-RPC Fault #10001 -
Bad number of parameters in getUserInfo: expected 1, got 0
Mar 4, 2003 9:41:22 AM: TestService: spent 3158 millis for request

===========================================
Mar 4, 2003 9:41:22 AM: Method.4=getUserInfo
Mar 4, 2003 9:41:22 AM: comment.4=wrong parameter type

Mar 4, 2003 9:41:22 AM: param.4.1.type=Integer
Mar 4, 2003 9:41:22 AM: param.4.1.value=10
Mar 4, 2003 9:41:25 AM: TestService: XML-RPC Fault #10002 -
Bad parameter type in getUserInfo: expected array for userIds,
got <type 'int'>
Mar 4, 2003 9:41:25 AM: TestService: spent 3183 millis for request

...

===========================================
Mar 4, 2003 9:42:25 AM: Method.23=saveResults
Mar 4, 2003 9:42:25 AM: comment.23=wrong parameter type
(Integer instead of Vector)
```

```
Mar 4, 2003 9:42:25 AM: param.23.1.type=String
Mar 4, 2003 9:42:25 AM: param.23.1.value=CW665_7880
Mar 4, 2003 9:42:25 AM: param.23.2.type=String
Mar 4, 2003 9:42:25 AM: param.23.2.value=CW665_7853
Mar 4, 2003 9:42:25 AM: param.23.3.type=Integer
Mar 4, 2003 9:42:25 AM: param.23.3.value=10
Mar 4, 2003 9:42:28 AM: TestService: XML-RPC Fault #10002 -
Bad parameter type in saveResults: expected array for results,
got <type 'int'>
Mar 4, 2003 9:42:28 AM: TestService: spent 3183 millis for request
```

- **Test summary:** All methods called from the AS and the SBS were tested. All methods tested worked according to the specification for all sets of correct input parameters. Correct calls did not produce any errors, incorrect calls produced the appropriate error codes and messages.

3.3 CWS GUI test

3.3.1 CWS GUI description

The graphical user interface of the CWS presents the folders to which a registered user of the Cyclades system has access and allows for navigation among these folders and for adding, moving, deleting, reading or editing artifacts within those folders. The user interface shows the contents of one folder at a time in a folder listing. The functionality may be invoked by using menus and buttons that are part of a folder listing.

The folder listing

The CWS user interface presents the contents of a folder in a tabular representation with a header containing menus, navigation buttons and action shortcut buttons. As an example figure 3.1 shows a folder listing with an open File menu for creating new objects in the folder.

Several classes of artifacts can be created and shared in a folder: various kinds of folders (private, project, community), discussion forums containing notes, and annotations of artifacts which also consist of notes. In project folders, additional kinds of artifacts may be created and shared: documents and URLs. Records and queries are not created via user operations, but transferred from the Search and Browse Service; they may also be shared in folders.

Each artifact contained in a folder is represented by an entry in the folder listing. An entry consists of an information button, a checkbox (that a user has to "tick" to select the artifact for some multi-action), the name of the artifact, some icons, additional data and an action menu button.

Most prominent in each entry is the name of the artifact. Obviously, a user should choose names describing the content or purpose of each individual artifact. The icon immediately left of the artifact name denotes the artifact type. To the right of the artifact name the CWS user interface displays:

- Zero, one or more of the following icons:
  - *shared icon* indicates that a folder is shared;
  - *lock icon* indicates that someone has set a lock for this artifact;
  - *note icon* indicates that a note has been added to the artifact;
  - *rate icon* indicates that the document has been rated by one or more folder members;

- the user name of its owner;
- date and time of the most recent modification;
- zero, one or more icons indicating that some of the following events have occurred:
  - *new icon* indicates a new artifact;
  - *change icon* indicates changes to the artifact;
  - *read icon* indicates that someone has read the artifact;
  - *modification icon* indicates recent modifications in a sub-artifact;
- a menu button showing the actions involving the artifact.

Note that different actions are possible for different types of artifacts. Depending on the individual access rights, the number and type of actions permitted may vary.

Most icons in an artifact entry are "clickable", i. e. one gets more information on a community, a lock, a note, a rating etc. when one clicks on it.

The instant access navigation buttons

The upper right hand corner of the interface provides an icon bar showing buttons applicable to instantly access certain artifacts.

- *home folder icon* represents a user's home folder.
- *communities icon* represents a the existing communities.
- *clipboard icon* recpresents a user's clipboard, which serves as an intermediate store.
- *waste icon* represents a user's waste, which helps to prevent unauthorized or unintentional deletion of artifacts: In the CWS, an artifact can be irrevocably destroyed only from the waste-basket of its owner.
- *address book icon* represents a user's address book, to be used primarily to invite new members to the user's folder.
- *search & browse icon* represents the access to the Search & Browse Service.

The menu bar

At the upper left hand corner of the interface there is a menu bar with five menus and action shortcut buttons for the most frequent actions.

New artifacts are added to the current folder by selecting one of the *File* menu options:

File → *New* plus a sub-option from the list *Discussion, Private Folder, Community Folder, Project Folder* in order to create an artifact of the specified type directly on the CWS server. For project folders, also documents, and URLs may be created. Examples are:

- Select *File* → *New* → *Private Folder* in order to create a private folder within the current private or home folder.
- Select *File* → *New* → *Community Folder* in order to create a community folder within the current community or home folder.
- Select *File* → *New* → *Project Folder* in order to create a project folder within the current project or home folder.

- Select *File* → *New* → *Document* to upload a file from the user's local computer system to the current project folder.

If the user wants to create a new private, community or project folder, the CWS user interface asks for *name, description, collections* to be associated to that folder; finally the user is asked to specify whether recommendations are requested for *records, users, collections, or communities*, (i. e. tick the appropriate box). For communities one has to also specify in the creation dialogue whether the community is open to be joined by external users.

When a new document is identified to be uploaded to the CWS server, the user is asked via an additional dialogue to specify the document's local URL, name, description, MIME type and encoding.

If the user wants to rate the new artifact, she may choose one of the following options: *no* (for no rating available), *very poor, poor, fair, good, or excellent*, by ticking the respective radio button.

The *Edit* menu is used to transfer existing artifacts to/from from the clipboard by the following procedure. Select the *Paste* option in the Edit menu to add artifacts that arrived at the Clipboard from somewhere in the user's folders as a result of the most recent *Copy* or *Cut* action to the current Folder.

Via the *Options* menu, the user may edit *Preferences,* personal user *Details,* user *Communication* specifics, or user *Default Events.*

To navigate among the particular entities one can also use the *GoTo* menu.

- Select *GoTo* → *Home* to get to the user's home folder.
- Select *GoTo* → *Communities* to get to the existing communities.
- Select *GoTo* → *Clipboard* to get to the user's clipboard.
- Select *GoTo* → *Waste* to get to the user's waste.
- Select *GoTo* → *Address Book* to get to the user's address book.
- Select *GoTo* → *Search & Browse* to get to the Search & Browse Service.
- Select *GoTo* → *User Info* to receive the specifics of the user information stored in the system.

System administrators may access the existing collections and services via the following options of the *GoTo* menu.

- Select *GoTo* → *Collections*
- Select *GoTo* → *Services*

The communities listing

The communities listing shows the existing communities in the style of a folder listing. The communities are represented by their root folders. Community objects in the communities listing allow the following functions via the action menus (cf. below):

- *Join Community* to join the community, if it is open to subscription,
- *Mail Community Mgrs* to send an electronic letter to the manager(s) of a particular community (e. g. to get invited to a community).

The collections and services listing

The collections and services listings are intended mainly for administrators and show the existing collections and services in the style of a folder listing.

The collections folder allows the function

- *Update Collections* to update the list of collections from the Collection Service

via the action menu (cf. below). The collection objects have no specific functions other than *Info*.

The service objects in the services listing indicate the status of the service via the service icon and allow the functions

- *Ping* to test the availability of the service,
- *Process Queue* to process queued calls for this service (administrators only)

via the action menu. The services folder allows the function

- *Update Services* to update the list of services from the Mediator Service

also via its action menu.

The recommendations folder

Every folder may have up to one recommendations folder that contains the recommendations of records, collections, users or communities intended for this folder. The title of the recommendations folder is always *Recommendations*, it may not be moved. A recommendations folder may only contain recommended objects inserted by the system when received from the Filtering and Recommendation Service. Recommended objects allow the following functions via the action menu (cf. below):

- recommended records may be moved or copied to other folders,
- recommended collections may be associated to the containing folder,
- recommended users may be invited to the containing folder (managers only) or may be contacted by electronic mail,
- recommended communities may be joined if open for subscription or their managers may be contacted by electronic mail.

Recommendations may of course also be deleted. All these functions are invoked via the action menu of the respective entry.

The multi-action buttons

Preceding the list of artifacts, the CWS user interface provides a number of buttons for actions to be applied to several selected artifacts.

In general, an artifact is selected by marking the checkbox to the left of its name. The *Select all* and *Select none* buttons are shortcuts for selecting or de-selecting all artifacts within the current folder.

Hitting, e. g., the *copy* button invokes copying, and hitting the *cut* button triggers transfer of the selected artifacts to the user's clipboard; hitting the *delete* button invokes transfer of the selected artifacts to the waste. Certain buttons such as *send* and *rate* trigger actions that can be applied only to artifacts of specific types.

Note that artifacts transferred from a folder to the clipboard or to the waste are no longer visible to the members of the folder.

The action menus

At the right end of the entry containing the artifact name in the folder listing, the CWS user interface provides for every entry an action menu for operations to be applied only to that particular artifact. Here one gets all the actions that are applicable to the artifact, including generic ones like *Open, Catch up, History, Info*, or actions that depend on the nature of the artifact, e. g. *Rate, Attach note* for records or *Add Collections, Remove Collections* for CYCLADES folders.

Here in the single action menu the actions more specific to CWS are to be found that do not appear in the menu bar or the multi-actions:

- *Add Collections* to associate additional collections to a CYCLADES folder,
- *Remove Collections* to remove collections from the set of collections associated to a CYCLADES folder,
- *Edit Reco Prefs* to modify the preferences for the recommendations sought for,
- *Allow Subscription* to allow subscription to a community,
- *Deny Subscription* to refuse subscription to a community,
- *Update Folder Profile* to request an immediate update of the folder profile in the Filtering and Recommendation Service,
- *Join Community* to join the selected community,
- *Mail Community Mgrs* to send an electronic letter to the manager(s) of a particular community (e. g. to get permission to join a community).

The configuration of the Action menu depends on the type of artifact—for example, different actions are appropriate for a URL artifact, a folder or a document.

What actions are applicable to which artifacts? Access right management in the CWS is based on roles, whereby a role defines the set of artifacts and actions a user may apply for a specific task. As a consequence, the CWS will not display the single action menu entries for actions that a user may not perform on the specific artifact w. r. t. to the role assigned to the particular user. On the other hand, one may invite, for example, new community members assigning roles to them. Moreover, roles define access profiles which can be attached to any artifact in the CWS. A set of pre-defined roles serves as a starting point: *manager*, *owner* (originally the creator; also accountable for the disk space used), *member* and *restricted member* (read-only access). Role assignments are inherited along the folder hierarchy and can be modified at any time. In the pre-defined role definition, member management is reserved to managers.

Context Sensitive Help

The context sensitive help feature provides assistance to navigate the system. In a number of application environments one may click the *question mark* button to get an explanation of the action to be launched.

3.3.2 CWS GUI test plan

The CWS is based on BSCW. The following test plan lists test cases for functionality that has been specifically implemented for the CWS; it also lists test cases for functionality that is standard in BSCW, but only for cases where CWS specific objects are concerned.

The CWS GUI test plan is based on the list of 20 use cases that have been identified for the CWS in Deliverable D3.0.1. When a use case is too coarse to allow a proper test of functionality, it is split into a number of appropriate subcases, e.g. "Join community" is split into two subcases:

- Execute 'Join community' on a recommended community.
- Execute 'Join community' on a community shown in the Communities listing

The interaction sequence for the test cases may involve none, one or two intermediary forms where to enter parameters that are necessary for the completion of the test case. For the test plan, we list only the parameters, not a description of the interaction.

Create folder

When creating folders we distinguish two cases: the user creates the root of a new folder hierarchy in her home folder, or the user creates a subfolder within in an existing folder hierarchy. In the first case, a number of preferences have to be set which in the second case are inherited from the parent folder.

- **Create root folder**
  Creation of a root folder of a private domain, a project, or a community is split into three subcases for each kind of folder since different object types and input parameters are involved.
  - **Create private root folder**
    Action: Menu bar File→New→Private Folder in the home folder
    Parameters:
    - folder name
    - folder description
    - collections to be associated to the folder (from the personal set of collections when defined, from the set of all collections when no personal set has been defined)
    - recommendation preferences (which of the four varieties of recommendations (records, users, communities, collections) are welcome to the folder)
  - **Create project root folder**
    Action: Menu bar File→New→Project Folder in the home folder
    Parameters:
    - folder name
    - folder description
    - collections to be associated to the folder
    - recommendation preferences
  - **Create community root folder**
    Action: Menu bar File→New→Community in the home folder
    Parameters:
    - folder name
    - folder description
    - collections to be associated to the folder
    - recommendation preferences

        * subscription policy preference (open to subscriptions from outside or not).

    Check:

        * After creation of a community root folder the respective community should show up in the communities folder (cf. 'View communities' below)

- **Create subfolder**
  Again, the creation of a subfolder in a private domain, a project, or a community is split into three subcases for each kind of folder since different object types and input parameters are involved.
    - **Create private subfolder**
      Action: Menu bar File→New→Private Folder in a private folder
      Parameters:
        * folder name
        * folder description
        * recommendation preferences
    - **Create project subfolder**
      Action: Menu bar File→New→Project Folder in a project folder
      Parameters:
        * folder name
        * folder description
        * recommendation preferences
    - **Create community subfolder**
      Action: Menu bar File→New→Community Folder in a community folder
      Parameters:
        * folder name
        * folder description
        * recommendation preferences

Move and copy

Moving and copying objects within the CWS from one folder to another are generic BSCW operations that are also available for the CWS specific objects: the various kinds of folders, records, and queries. The mechanism works with a container called clipboard that every user has apart from her home folder. More than one object of different type may be moved or copied together.

Check:

- Moving and/or copying of certain objects should not be possible: Recommendations and recommendations folders, communities folder and its communities.

Delete, undelete and destroy

Again, deleting, undeleting and destroying objects are generic operations within the CWS available for folders, records, queries, discussion forums, and any other type of documents. The mechanism works with a container called waste that every user has apart from her home folder and clipboard. Several objects of different type may be deleted, undeleted or destroyed together.

Check:

- Deleting means destroying for some objects: recommendations and recommendations folders.
- Some objects cannot be deleted: communities folder and communities
- The FRS should be notified of deleting and undeleting of records.

Edit folder attributes

Folder attributes that may be edited by the user are

- name
- description
- associated collections
- recommendation preferences
- subscription policy preference (community root folders only)

All in all, we have seven subcases.

- **Edit name**
  Action: action menu 'Rename' of a folder entry
  Parameters: new folder name
- **Edit description**
  Action: action menu 'Description' of a folder entry
  Parameters: new folder description
- **Add associated collections**
  Parameters: additional collections Action: action menu 'Add Collections' of a folder entry
- **Remove associated collections**
  Action: action menu 'Remove Collections' of a folder entry
  Parameters: collections to be removed
- **Edit recommendation preferences**
  Action: action menu 'Edit Reco Prefs' of a folder entry
  Parameters: the new recommendation preferences for
  - records
  - collections
  - users
  - communities
- **Allow subscription** (community root folders only)
  Action: action menu 'Allow Subscription' of a folder entry
- **Deny subscription** (community root folder only)
  Action: action menu 'Deny Subscription' of a folder entry

Check:

- 'Allow Subscription' and 'Deny Subscription' should toggle.
- Changing the name of a community root folder should also change the name of the corresponding community in the communities folder.

Rate records (and other artifacts)

Rating is a generic operation within the CWS. Apart from the notification of the RMS and FRS in the case of records, the operation is equal for queries, recommendations, or other documents and was not tested separately. Also, mixed types of objects may be rated together in one operation. In this case, only record ratings are forwarded to the RMS and FRS.

- **Rate record**
  Action:
  - action menu 'Rate' of a record entry
  - multi-action 'rate' for all checked entries in a listing

  Parameters: the rating(s)

Annotate records (and other artifacts)

Annotations of records take the form of threaded discussions (like the discussion forums treated below). The first annotation opens the discussion, subsequent annotations are added to the discussion.

Annotating is a generic operation within the CWS. The operation is equal for queries, recommendations and other documents, and was not tested separately.

- **Annotate record**
  Action: action menu 'Attach Note' of a record entry
  Parameters:
  - the character of the annotation (Note, Pro, Con, ...)
  - the subject of the annotation
  - the text body of the annotation

Invite new members to a folder

The right to invite new members is restricted to managers. Invitation to communities is only possible in the community root folder and holds for all folders of the community. With project folders, new members may also be invited to subfolders of a project and then have no access to the parent folder of the folder they were invited to.

Invitation of new members may invoke a two-stage interaction when members are invited who are not already in the invitor's address book. This will unvariably the case, when emebers are to be invited who are not already registered users. In this case, the mail addresses (or login names) of these new members have first to be added to the invitor's address book and may then be selected for invitation.

- **Invite new members**
  Action:
  - menu bar File→Share→Invite Member
  - short-cut 'Members Icon' button

  Parameters:
  - the role in which new members are invited (Member, Restricted Member, Manager)
  - the login names of the new members

  - the mail adresses of the new members to be entered in the the subfunction 'Add Member to Address Book'

Check:

- Invitation should only be possible in community root folders and project folders.
- Only managers should be able to invite.
- Invitation to a root folder should be valid for all subfolders.

Remove members from a folder

The right to remove members is restricted to managers. Removal from communities is only possible in the community root folder and holds for all folders of the community. With project folders, members may also be removed from subfolders of a project and then have no access to the subfolders of this folder.

Removal is done in the member listing of a folder. Members shown only with their email address are invited members who have not yet registered. The member listing of a folder may be viewed by hitting on the 'Members Icon' directly preceding the folder name in the 'Your location' part of the CWS user interface.

- **Remove members**
  Action:
  - action menu 'Remove' of an entry in the member listing
  - multi-action 'remove' for all checked entries of the member listing

Check:

- Member removal should only be possible in community root folders and project folders.
- Only managers should be able to remove members.
- Member removal in a root folder should be valid for all subfolders.

Assign a member as manager

The right to assign members with the manager role is restricted to managers. Managers have also the right to assign managers with the member role. Role assignment may be done for all members of a folder.

- **Assign as manager**
  Action: action menu 'Assign Role' of folder entry
  Parameters: for each member
  - the new role assignment (Manager, Member, Restricted Member)

Check:

- Role assignment should only be possible in community root folders and project folders.
- Only managers should be able to assign roles to members.
- Role assignment in a community root folder should be valid for the whole community, i.e. all its folders.

Leave a community or project

Leaving a community or project is done by destroying the respective root folder from the user's home folder. With projects, one can also leave parts of the project, i.e. part hierarchies of the project folder hierarchy. Destroying a folder is done like destroying any other artifact and was treated in a test case above.

Check:

- Leaving a community by one member should leave the community available for the other members.
- Leaving the community by the last member should remove this community also from the communities folder (cf. View communities below).

View communities

This operation allows users to become aware of the existing communities.

- **View communities**
  Action:
  - menu bar Goto→Communities
  - navigation bar 'Communities Icon' button

Check:

- Consistency between the existing community root folders and the communities shown.

Join a community

This is an operation that is only possible for community objects as listed in the communities folder and for community recommendations as listed in the recommendations folder of some folder. In both cases, joining is only possible if the community's policy is open to subscriptions from outside, and if the user is not already member.

- **Join a community in the communities listing**
  Action: action menu 'Join Community' of a community entry
- **Join a recommended community**
  Action: action menu 'Join Community' of a community recommendation

Check:

- The above conditions for applicability of the operation.

Mail community managers

This is an operation that is only possible for community objects as listed in the communities folder and for community recommendations as listed in the recommendations folder of some folder. This operation is only possible for users not being managers of the community.

- **Mail managers of a community in the communities listing**
  Action: action menu 'Mail Managers' of a community entry
  Parameters: the subject and text of the message

- **Mail managers of a recommended community**
  Action: action menu 'Mail Managers' of a community recommendation Parameters: the subject and text of the message

Check:

- The above conditions for applicability of the operation.

Create discussion forums

Creation of discussion forums is a BSCW generic operation and was not specifically tested.

Add notes to a discussion forum

Adding notes to a discussion forums is a BSCW generic operation and was not specifically tested.

Edit event notification preferences

Editing event notification preferences is a BSCW generic operation and was not specifically tested.

Catching up on events

Catching up on events is a BSCW generic operation and was not specifically tested.

Edit personal preferences

Personal preferences have to do with email formats, known editors, user profile, user interface language etc. For the CWS, an attribute has been added to the personal preferences that states whether the user allows recommendation of herself to other users.

- **Edit personal preferences**
  Action: menu bar Options→Preferences
  Parameters: check or uncheck 'Allow recommendation' box

Check:

- The user may not be shown in recommendations folders even if recommended to others by the FRS.

Processing recommendations

Recommendations are received from the FRS and put into the recommendations subfolder of the folder for which the recommendations are meant.

Processing recommendations includes all operations that dispose of received recommendations. The type of the operations possible depends on the type of the recommendations (records, users, communities, collections). Deleting is always possible (and means destroying for user, community and collection recommendations). Moving and copying is only possible for recommended records. Joining communities and mailing community managers is only possible for recommended communities (and has already been treated as test cases above). For user and collection recommendations, there are the operations invite and associate that refer to the parent folder of the recommendations folder.

- **Invite a recommended user to the folder**
  Action: action menu 'Invite' of a user recommendation entry
- **Associate a recommended collection to the folder**
  Action: action menu 'Associate' of a collection recommendation
- **Move/copy recommended records to some other folder**
  Action: 'Cut'/'Copy' and 'Paste' actions
  - in menu bar 'Edit → '
  - in action menu

Check:

- Conditions of applicability for join and mail operations.
- User recommendations should only be received for users not already members of the parent folder.
- Collection recommendations should not be received for collections already associated to the parent folder.
- The FRS should not be notified of records being put into, or removed from, a recommendations folder.

Update folder profile

This operation is useful before searching and browsing when folder contents have been changed a great deal. With the present policy of direct FRS notification of record movements, it has become less necessary.

- **Update folder profile**
  Action: action menu 'Update Folder Profile' of a folder entry

3.3.3 CWS GUI test results

- **Tester:** S. Caizzi (caizzi@aliceinchains.com)
- **Test date:** 28 August, 2002 14:50 GMT
- **Scope of test:** Complete GUI of CWS v0.1 (http://cyclades.gmd.de/cws/cws.cgi) was tested according to the above test plan.
- **Test environment:** Test client was a Internet Explorer running on Windows 2000.
- **Test plan:** The test plan is contained in the preceding section.
- **Test log:** During the test of the use cases, the parameters chosen (if any) were documented along with results. Consistency tests that are possible within an isolated service test are also documented. An excerpt of the test log is shown below.

```
Test log, 28-08-2002 14:45
  ...
Case: create private root folder
  Parameters:
    Name: Instant favorites
    Description: What I am fond  of ...
```

```
      Associated Collections: Antiquities, Late Byzantine Mosaics,
        South Pole geographic documentation
      Recommendation preferences: Users
    Result:
      OK
    Checked: ‘Instant favorites’ is shown in the folder listing

  Case: create project root folder
    Parameters:
      Name: CWS testing
      Description: Metadata  somehow ...
      Associated Collections: None
      Recommendation preferences: Records
    Result:
      OK
    Checked: ‘CWS testing’ is shown in the folder listing

  Case: create community root folder
      Name: Music community
      Description: Information  about tour dates, LP releases ...
      Associated Collections: None
      Recommendation preferences: Collections, Communities
      Community  may be joined  from outside
    Result:
      OK
    Checked:
      ‘Music community’ is shown in the communities listing

  Case: Get information about ‘Music community’
    Result:
      OK

  Case: create private root folder
    Parameters:
      Name: Favorites
      Description: None
      Associated Collections: None
      Recommendation preferences: None
    Result:
      OK
    Checked: ‘Favorites’ is shown in the folder listing

  Case: create project folder in project folder ‘CWS testing’
    Parameters:
      Name: GUI testing
      Description: None
      Associated Collections: None
      Recommendation preferences: Records
    Result:
      OK
    Checked: ‘GUI testing’ is shown in the ‘CWS testing’ folder listing

  Case: create community folder in community folder ‘Music community’
    Parameters:
      Name: Folk Music
      Description: None
      Associated Collections: None
      Recommendation preferences: Records, Communities
    Result:
      OK
    Checked: ‘Folk Music’ is shown in the communities listing

  Case: Cut folder ‘Instant favorites’
    Result:
      OK
    Checked: ‘Instant favorites’ is shown in the clipboard listing

  Case: Paste folder ‘Instant favorites’ in private folder ‘Favorites’
    Result:
      OK
    Checked: ‘Instant favorites’ is shown in the ‘Favorites’ listing

  Case: Cut document ‘Test plan.doc’ from private folder ‘First folder’
    Result:
      OK
    Checked: ‘Test plan.doc’ is shown in the clipboard listing

  Case: Paste document ‘Test plan.doc’ in project folder ‘CWS testing’
    Result:
      OK
    Checked: ‘Test plan.doc’ is shown in the ‘CWS testing’ listing

  Case: Delete folder ‘First folder’
    Result:
      OK
    Checked: ‘First folder’ is shown in the waste listing

  Case: Undelete folder ‘First folder’
    Result:
      OK
    Checked: ‘First folder’ is shown in the folder listing

  Case: Delete document ‘Fosse la mia Principessa ...’ from folder ‘First folder’
    Result:
      OK
    Checked: ‘Fosse la mia Principessa ...’ is shown in the waste listing

  Case: Destroy document ‘Fosse la mia Principessa ...’ from waste
    Result:
      OK
    Checked: ‘Fosse la mia Principessa ...’ disappeared in the waste listing

  Case: Rename project folder ‘First folder’
    Parameters:
      New Name: ‘Dear all ...’
    Result:
      OK
    Checked: ‘Dear all ...’ is shown in the folder listing
```

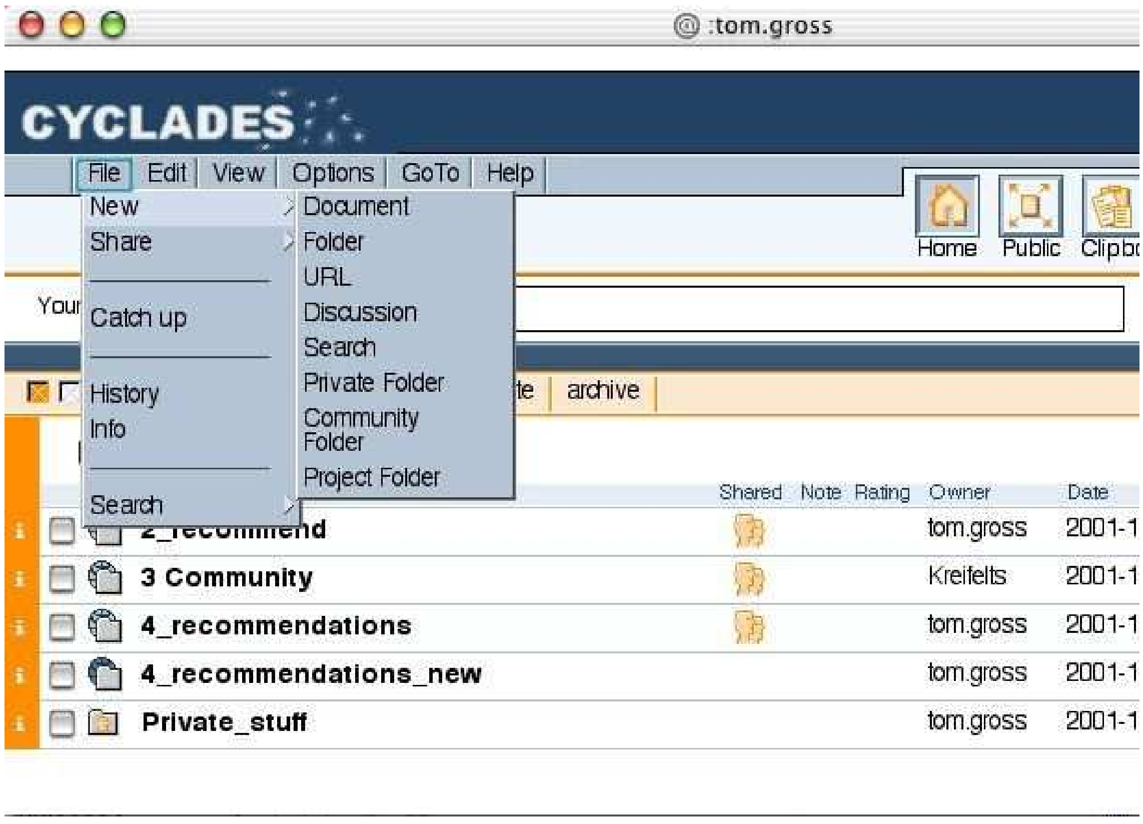
```
Case: Edit description of project folder ‘CWS testing’
  Parameters:
    Description: Test reports and so on
  Result:
    OK
  Checked: ‘Test reports and so on’ appears under ‘CWS testing’
    in the folder listing

Case: Associate collections to private folder ‘Favorites’
  Parameters:
    Associated Collections: Antiquities, Late Byzantine Mosaics,
    South Pole geographic documentation
  Result:
    OK
  Checked:

Case: Remove collections from private folder ‘Favorites’
  Parameters:
    Collections to be removed: South Pole geographic documentation
  Result:
    OK
  Checked:

Case: Edit Reco Prefs for private folder ‘Instant Favorites’ in
    private folder ‘Favorites’
  Parameters:
    Recommendation preferences: Users, Collections
  Result:
    OK
  Checked:

Case: Update ‘Music community’ folder profile
  Result:
    OK
  Checked:

Case: Rate  document ‘Test plan.doc’ in project folder ‘CWS testing’
  Parameters:
    Rating: Poor
  Result:
    OK
  Checked: a rating icon appeats right to ‘Test plan.doc’

Case: Annotate document ‘Test plan.doc’ in project folder ‘CWS testing’
  Parameters:
    Type: Note
    Subject: Warning
    Message: Should be written as soon as possible ...
  Result:
    OK
  Checked: an annotation icon appeats right to ‘Test plan.doc’

Case: Annotate document ‘Test plan.doc’ in project folder ‘CWS testing’
```

```
  Parameters:
    Type: Note
    Subject: Second Warning
    Message: Hurry up !!!
  Result:
    OK
  Checked:

Case: Invite a new member to the ‘CWS testing’ folder
  Parameters:
    Type of member: Restricted
    New member e-mail: wirsam@web.de
    Invitation language: en
    Invitation text: For testing purposes only ...
  Result:
    OK
  Checked:

Case: Remove a member from the ‘CWS testing’ folder
  Result:
    OK
  Checked:

Case: Assign a member as manager
  Parameters:
    Managers: ordosoft@yahoo.com, scaizzi
  Result:
    OK
  Checked:

Case: Leave the TOMs_test_community2 community
  Parameters: Delete TOMs_test_community2 from the root folder, click on
    the Waste icon and destroy TOMs_test_community2
  Result:
    OK
  Checked: the TOMs_test_community2 community folder disappeared from
    the folder listing

Case: View communities
  Parameters: Click on the Comms icon
  Result:
    OK
  Checked: community folders are shown in the folder listing

Case: Join the TOMs_test_community2 community in the Communities folder
  Parameters: Click on the action arrow and select Join community
  Result:
    OK
  Checked: a TOMs_test_community2 community folder is shown in
    the folder listing

Case: Mail community managers
  Parameters: Click on the action arrow and select Mail community mgrs
  Result:
    OK
  Checked:

Case: Create a discussion forum in the ‘Music community’ community folder
  Parameters:
    Name: Your opinion on metal bands ...
    Type: Idea
    Subject: Great or has-been ?
    Message: These days, mediocrity seems to rule the scene: in  fact,
      Metallica is dead, Pantera disappointing, SOAD is great but no metal anyway,
      Limp Bizkit is pure sh** as well as Korn and ... what the ...
      teens have no taste at all and prefer RNB cuz its cool man ... Disgusting ...
  Result:
    OK
  Checked:

Case: Add note to the ‘Your opinion on metal bands ...’ discussion forum
    in the ‘Music community’ community folder
  Parameters:
    Name: Your opinion on metal bands ...
    Type: Pro
    Subject: Some bands are still great ...
    Message: Come on dude, Ya oughta be optimistic: bands like Mushroomhead, Godsmack,
      Disturbed are kinda promising and take also a look at Panteras
      latest hostility that is Metal Edge: I swear Ya wont be disappointed ...
  Result:
    OK
  Checked:

Case: Edit event notification preferences
  Parameters:
    Notification preferences: Create Events, Change Events
  Result:
    OK
  Checked:

Case: Edit default event notification preferences
  Parameters:
    Event Service activated
    Notification preferences: Read Events, Create Events, Move Events, Change Events
  Result:
    OK
  Checked:

Case: Catch up on events in root folder
  Parameters: Click on the action arrow right to :scaizzi and select Catch up
  Result:
    OK
  Checked:

Case: Edit personal preferences
  Parameters:
    Format of email messages: plain text
    BCC of sent mail
```

```
      one selfs recommendation to others not allowed
      Show descriptions by default
      No file upload Helper
      No ActiveX
      User Profile: Expert
      Known editors: All of them
      Javascript
      Language English
      Supported languages: Deutsch, English
    Result:
      OK
    Checked:

  ....
```

- **Test summary:** All test case functions worked according to the use case descriptions and produced no errors or inconsistencies.
  The test, however, detected three irregularities in the CWS GUI:
  - The function *Change Password* was provided by the CWS, but should rather be provided by the Mediator Service only to avoid fatal inconsistencies when logging into the system.
  - The function *Invite Members* provided an input attribute that was subsequently ignored (language of the invitation message).
  - The function *Archive* for archiving the contents of a folder produced an error when the *compress* option was chosen.

  These cases were attended to by removing the function *Change Password* from the CWS GUI and by removing the language attribute from the input form of the *Invite Members* function. The *compress* option of the *Archive* function needs the operating system dependent installation of a compress routine which is outside the scope of the CWS; if need arises, such a routine will be installed on the target host.



	Shared	Note	Rating	Owner	Date
2_recommend	●			tom.gross	2001-1
3 Community	●			Kreifelts	2001-1
4_recommendations	●			tom.gross	2001-1
4_recommendations_new				tom.gross	2001-1
Private_stuff				tom.gross	2001-1

Figure 3.1: CWS user interface

Chapter 4

SBS Component Test

4.1 Introduction

The Search and Browse Service (SBS) provides the search and browse functionality of the CYCLADES system. The user can select a set of collections to search, choose between the schemas available for these collections, and formulate query conditions which may be mandatory (the condition must be fulfilled), optional (the condition may be fulfilled), or negative (the condition must not be fulfilled).

4.2 SBS API test

4.2.1 SBS API specification

The Search and Browse Service provides an API to the other CYCLADES services, as well as to its own GUI. The methods of this API may be called using the inter-service communication protocol XML-RPC. The only public method is getId. All other methods may be called only by the SBS GUI.

public:

- **Method:** getId
  Signature: String getId()
  Description: this method returns the ID of the service
  Parameters:
  Output: Search And Browse Service ID
  Calling services: any

service internal:

- **Method:** getSessions
  Signature: SearchBrowseSession* getSessions()
  Description: this method returns a list of the current S&B sessions
  Parameters:
  Output: a list of SearchBrowseSession objects

- **Method:** initiateSearch
  Signature: SearchBrowseSession initiateSearch(id,folderId,userId)

Description: this method can be called to initiate a search and browse session for the given *folderId* and *userId*, the session id *id* is generated by the caller (the GUI)
Parameters:

Input:	id:	the id of the new session
	userId:	the id of user who started the search
	folderId:	the id of the folder the user started the search from

Output: a new SearchBrowseSession object

- **Method:** deleteSession
  Signature: void deleteSession(sessionId)
  Description: this method can be called to explicitly remove the temporary data stored for a session from the SBS memory
  Parameters:

Input:	sessionId:	the id of the session

- **Method:** validateSession
  Signature: boolean validateSession(sessionId,flderId,userId)
  Description: check if there is a session with the specified ID for the specified *userId* and the specified *folderId*
  Parameters:

Input:	sessionId:	the id of the session
	folderId:	the id of the folder
	userId:	the id of the user

  Output: true, if session ID, user ID, and folder ID are consistent, false otherwise

- **Method:** getNewForFolder
  Signature: Record* getNewForFolder(sessionId)
  Description: determines the list of new records that are relevant to the current folder topic
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to

  Output: a list of records

- **Method:** search
  Signature: Record* search(sessionId, query)
  Description: returns a list of records that are deemed relevant with respect to the specified query
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
	query:	the query to be evaluated (string)

  Output: a list of records

- **Method:** filteredSearch
  Signature: Record* filteredSearch(sessionId, query)
  Description: returns a list of records that are deemed relevant with respect to the specified query, filtered with respect to the current folder topic
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
	query:	the query to be evaluated (string)

  Output: a list of records

- **Method:** saveResults
  Signature: void saveResults(sessionId, recordId*)
  Description: saves the records specified by recordId* to the user's folder
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
	list of recordId:	the list of record ids of the records that are to be stored to the user's folder

- **Method:** saveQuery
  Signature: void Query(sessionId, Query)
  Description: saves the specified query to the user's folder
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
	Query:	an SBS Query object

- **Method:** getSchemas
  Signature: Schema* getSchemas(sessionId)
  Description: returns the list of schemas available in a search and browse session
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
Output:	a list of Schema objects	

- **Method:** getCollections
  Signature: (Collection*, Schema*) getCollections(sessionId)
  Description: returns the list of collections and the corresponding schemas available in a search and browse session
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
Output:	list of Collection:	list of Collection objects
	list of Schema:	list of Schema objects

- **Method:** getPersonalCollections
  Signature: (Collection*, Schema*) getCollections(sessionId)
  Description: returns the list of personal collections for the user together with the corresponding schemas
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
Output:	list of Collection:	list of Collection objects
	list of Schema:	list of Schema objects

- **Method:** getFolderCollections
  Signature: (Collection*, Schema*) getCollections(sessionId)
  Description: returns the list of collections associated to the current folder together with the corresponding schemas
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
Output:	list of Collection:	list of Collection objects
	list of Schema:	list of Schema objects

- **Method:** getAttributeValues
  Signature: value* getAttributeValues(sessionId, schemaName, attributeName, maxNo)
  Description: returns the list of values for the specified attribute and schema available in a search and browse session
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
	schemaName:	the name of the schema
	attributeName:	the name of the attribute
	maxNo:	the maximum number of values to be returned

Output: a list of attribute values, ordered by the default order according to the attribute's type

- **Method:** getRecord
  Signature: Record getRecord(sessionId,recordId)
  Description: get the metadata record specified by *recordId*
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
	recordId:	the ID of the record

  Output: a Record object

- **Method:** getResultHistory
  Signature: (queryId,Record*)* getResultHistory(sessionId)
  Description: get the all results of all queries that were submitted during this session
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to

  Output: a list of query IDs, each with a list of resulting Records associated

- **Method:** getQuery
  Signature: Query getQuery(sessionId,queryId)
  Description: get the query specified by *queryId*
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to
	queryId:	the ID of the query

  Output: a Query object

- **Method:** getQueryHistory
  Signature: Query* getQueryHistory(sessionId)
  Description: get the queries that were submitted or saved during this session
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to

  Output: a list of Query objects

- **Method:** getFolderQueries
  Signature: Query* getFolderQueries(sessionId)
  Description: get the queries stored in the current folder
  Parameters:

Input:	sessionId:	the id of the SearchBrowseSession object this method refers to

  Output: a list of Query objects

The definition of the class SearchBrowseSession, that is used in some API signatures and that translates to XML-RPC `<struct>`s is as follows:

- **SearchBrowseSession**
  - **id:** the unique ID of this session (string)

  - **expirationTime:** the time this session will be expired and its temporary data deleted (string)
  - **userId:** the unique ID of the user who initiated this session (string)
  - **folderId:** the unique ID of the folder from which this session was started (string)
  - **currentQuery:** the current SB query in XML (string)

The classes Query and Record are described in the CWS chapter (3).

4.2.2 SBS API test

SBS API test (UNIDO/UNIDUE)

- **Tester:** Sascha Kriewel (kriewel@is.informatik.uni-duisburg.de)
- **Test date:** 7 March 2003
- **Scope of test:**

  Testing SBS API methods
- **Test environment:**

  Test client was a Java[1] application running on Debian Linux, the XML-RPC implementation of Apache XML Project.
- **Test plan:**

  Each method was called first with valid parameters, then with invalid parameters.

  With the valid method calls, the semantic result was checked:

  - *getId* did return the service ID
  - *initiateSearch* returned a new SearchBrowseSession object with the given folderId and userId
  - *getSessions* returned a list of SearchBrowseSession objects including the one for the search and browse session initiated during the step before
  - *validateSession* resulted in true for the existing session and false otherwise
  - *search* returned a list of records for the query to be evaluated
  - *saveResults* verifiably saved the records specified to the intended folder (using the web interface to verify this)
  - *saveQuery* saved the query specified to the intended folder, which could be checked by use of
  - *getFolderQueries*
  - *getFolderQueries* returned the query objects that had been saved to the current folder
  - *getCollections* returned a list of Collection and Schema objects
  - *getPersonalCollections* returned a list of Collection and Schema objects
  - *getFolderCollections* returned a list of Collection and Schema objects
  - *getAttributeValues* returned a list of values for the specified attribute and schema
  - *getRecord* returned a metadata record with the correct identifier
  - *getQuery* returned a query object with the correct identifier
  - *getQueryHistory* returned a list of query objects

[1]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)

- *getSchema* returned a list of available schemas
- *getNewForFolder* returned a list of records
- *getResultHistory* returned a list of query ids and lists of records for each id

With the intentionally invalid calls, it was checked whether the method returned the appropriate error code and message.

- **Test log:**

  An excerpt of the last test log is shown below.

```
...

Mar 5, 2003 4:24:46 PM: Reading from tests_sbs/SBSComponentTest.properties
Mar 5, 2003 4:24:46 PM: ServerURL=http://woodstock.is.informatik.uni-duisburg.de:15220
Mar 5, 2003 4:24:46 PM: ServerName=service
Mar 5, 2003 4:24:46 PM: proxyHost=
Mar 5, 2003 4:24:46 PM: proxyPort=

===========================================
Mar 5, 2003 4:24:46 PM: Method.1=getId
Mar 5, 2003 4:24:46 PM: comment.1=get SBS ID, no parameters
Mar 5, 2003 4:24:46 PM: flags.1=

Mar 5, 2003 4:24:47 PM: TestService: getId returned SB438
Mar 5, 2003 4:24:47 PM: TestService: spent 146 millis for request

===========================================
Mar 5, 2003 4:24:47 PM: Method.2=initiateSearch
Mar 5, 2003 4:24:47 PM: comment.2=start a new search and browse session
Mar 5, 2003 4:24:47 PM: flags.2=

Mar 5, 2003 4:24:47 PM: param.2.1.type=String
Mar 5, 2003 4:24:47 PM: param.2.1.value=id14556
Mar 5, 2003 4:24:47 PM: param.2.2.type=String
Mar 5, 2003 4:24:47 PM: param.2.2.value=CW665_20186
Mar 5, 2003 4:24:47 PM: param.2.3.type=String
Mar 5, 2003 4:24:47 PM: param.2.3.value=CW665_20164
Mar 5, 2003 4:24:47 PM: TestService: initiateSearch returned
    [activeFolderId=CW665_20186,
     userId=CW665_20164,
     currentQuery=,
     expirationTime=Wed Mar 05 17:10:21 CET 2003,
     id=id14556]
Mar 5, 2003 4:24:47 PM: TestService: spent 15 millis for request

===========================================
Mar 5, 2003 4:24:47 PM: Method.3=getSessions
Mar 5, 2003 4:24:47 PM: comment.3=test if new session is there, no parameters
Mar 5, 2003 4:24:47 PM: flags.3=

Mar 5, 2003 4:24:47 PM: TestService: getSessions returned [[...],
  [activeFolderId=CW665_20186,
   userId=CW665_20164,
   currentQuery=,
   expirationTime=Wed Mar 05 17:10:21 CET 2003,
   id=id14556]]
Mar 5, 2003 4:24:47 PM: TestService: spent 28 millis for request

===========================================
Mar 5, 2003 4:24:47 PM: Method.4=validateSession
Mar 5, 2003 4:24:47 PM: comment.4=validate the existing session
Mar 5, 2003 4:24:47 PM: flags.4=

Mar 5, 2003 4:24:47 PM: param.4.1.type=String
Mar 5, 2003 4:24:47 PM: param.4.1.value=id14556
Mar 5, 2003 4:24:47 PM: param.4.2.type=String
Mar 5, 2003 4:24:47 PM: param.4.2.value=CW665_20186
Mar 5, 2003 4:24:47 PM: param.4.3.type=String
Mar 5, 2003 4:24:47 PM: param.4.3.value=CW665_20164
Mar 5, 2003 4:24:47 PM: TestService: validateSession returned
   [unknown type: true]
Mar 5, 2003 4:24:47 PM: TestService: spent 49 millis for request

===========================================

...

===========================================
Mar 5, 2003 4:24:47 PM: Method.8=search
Mar 5, 2003 4:24:47 PM: comment.8=
Mar 5, 2003 4:24:47 PM: flags.8=-ids-only

Mar 5, 2003 4:24:47 PM: param.8.1.type=String
Mar 5, 2003 4:24:47 PM: param.8.1.value=id14557
Mar 5, 2003 4:24:47 PM: param.8.2.type=String
Mar 5, 2003 4:24:47 PM: param.8.2.value=
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sbquery SYSTEM
    "http://woodstock.is.informatik.uni-duisburg.de:15210/ac/query.dtd">
<sbquery schema="dc">
  <condition weight="+" field="format">
    <field-condition predicate="$cw$" value="video"/>
  </condition>
  <collection id="SB438_CO_all"/>
</sbquery>
Mar 5, 2003 4:27:43 PM: TestService: search returned
    [id=AC143_oai_dc_oai:VTETD:etd-5414132139711101]
Mar 5, 2003 4:27:43 PM: TestService: spent 176372 millis for request

===========================================

...


===========================================
Mar 5, 2003 4:27:48 PM: Method.16=getAttributeValues
Mar 5, 2003 4:27:48 PM: comment.16=
Mar 5, 2003 4:27:48 PM: flags.16=
Mar 5, 2003 4:27:48 PM: param.16.1.type=String
Mar 5, 2003 4:27:48 PM: param.16.1.value=id14557
Mar 5, 2003 4:27:48 PM: param.16.2.type=String
Mar 5, 2003 4:27:48 PM: param.16.2.value=oai_dc
Mar 5, 2003 4:27:48 PM: param.16.3.type=String
Mar 5, 2003 4:27:48 PM: param.16.3.value=subject
Mar 5, 2003 4:27:48 PM: param.16.4.type=Integer
Mar 5, 2003 4:27:48 PM: param.16.4.value=10
Mar 5, 2003 4:27:51 PM: TestService: getAttributeValues returned
   ['AUHELAWA LANGUAGE,*Fang,*PB,*ProtoBantu,01 |
    AERONAUTICS,02 | AERODYNAMICS,03 | AIR TRANSPORTATION AND
    SAFETY,04 | AIRCRAFT COMMUNICATIONS AND NAVIGATIONS,05 |
    AIRCRAFT DESIGN, TESTING AND PERFORMANCE,06 | AVIONICS AND
    AIRCRAFT INSTRUMENTATION]
Mar 5, 2003 4:27:51 PM: TestService: spent 3910 millis for request

===========================================
Mar 5, 2003 4:27:51 PM: Method.17=getRecord
Mar 5, 2003 4:27:51 PM: comment.17=
Mar 5, 2003 4:27:51 PM: flags.17=

Mar 5, 2003 4:27:51 PM: param.17.1.type=String
Mar 5, 2003 4:27:51 PM: param.17.1.value=id14557
Mar 5, 2003 4:27:51 PM: param.17.2.type=String
Mar 5, 2003 4:27:51 PM: param.17.2.value=
                   AC143_oai_dc_oai:VTETD:etd-5414132139711101
Mar 5, 2003 4:27:54 PM: TestService: getRecord returned [name=
Intelligent Navigation of Autonomous Vehicles in an Automated
Highway System: Learning Methods and Interacting Vehicles Approach,
metadata=

...,
id=AC143_oai_dc_oai:VTETD:etd-5414132139711101]
Mar 5, 2003 4:27:54 PM: TestService: spent 2170 millis for request

===========================================
Mar 5, 2003 4:27:54 PM: Method.18=getQuery
Mar 5, 2003 4:27:54 PM: comment.18=try to get query with queryId
                                  that no longer exists
Mar 5, 2003 4:27:54 PM: flags.18=

Mar 5, 2003 4:27:54 PM: param.18.1.type=String
Mar 5, 2003 4:27:54 PM: param.18.1.value=id14557
Mar 5, 2003 4:27:54 PM: param.18.2.type=String
Mar 5, 2003 4:27:54 PM: param.18.2.value=SB438_id14557_1046873411430_1
Mar 5, 2003 4:27:54 PM: TestService: XML-RPC Fault #12500 -
   org.apache.xmlrpc.XmlRpcException: Unexpected resultgetQuery:
   got null for session id14557, query id SB438_id14557_1046873411430_1
Mar 5, 2003 4:27:54 PM: TestService: spent 463 millis for request

===========================================

...
```

```
==========================================
Mar 5, 2003 4:28:01 PM: Method.22=getFolderQueries
Mar 5, 2003 4:28:01 PM: comment.22=test if stored query is there
Mar 5, 2003 4:28:01 PM: flags.22=

Mar 5, 2003 4:28:01 PM: param.22.1.type=String
Mar 5, 2003 4:28:01 PM: param.22.1.value=id14557
Mar 5, 2003 4:28:05 PM: TestService: getFolderQueries returned
[[name=Query: test query 1,id=test query 1,queryString=
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sbquery SYSTEM
    "http://woodstock.is.informatik.uni-duisburg.de:15210/ac/query.dtd">
<sbquery schema="dc">
  <condition weight="+" field="format">
    <field-condition predicate="$cw$" value="video"/>
  </condition> <collection id="SB438_CO_all"/>
</sbquery>]]
Mar 5, 2003 4:28:05 PM: TestService: spent 3459 millis for request

...
```

- **Test summary:**

  All methods specified above from the SBS API were tested. All methods tested worked according to the specification for all sets of correct input parameters. Correct calls did not produce any errors, incorrect calls produced appropriate error codes and messages.

4.3 SBS GUI test

4.3.1 SBS GUI description

The SB GUI is used for searching and browsing for metadata records. The first page that is shown is the query formulation dialogue (figure 4.1).

A query consists of a (possibly empty) list of collections to be searched, and a list of conditions. This is reflected by the two list areas in the query formulation dialogue.

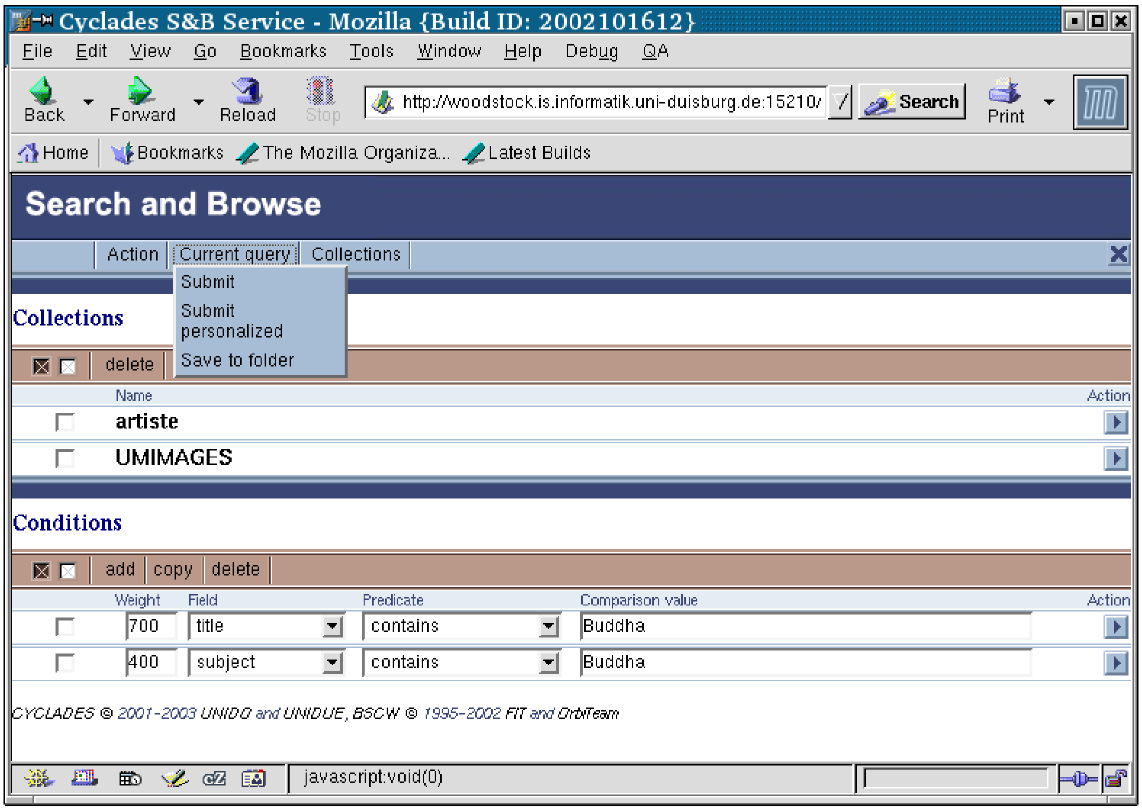
The Collection List

This list in the upper part of the query formulation dialogue contains those collections that the user wants to be searched in the current query. If the list does not contain any specific collection, a hint is displayed that all collections in the system will be searched.

In the menu bar above the collection list, the user can select all collections, or none. Individial collections can be selected via the checkbox beside the collection name. The user can then delete the selected collections from the query by pressing the *Delete* button in this bar.

Conditions

The list of conditions is displayed in the lower part of the query formulation dialogue. Each condition consists of an optional weight (which can be ”+”, ”-”, or a positive number between 1


Cyclades S&B Service - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
Back Forward Reload Stop
http://woodstock.is.informatik.uni-duisburg.de:15210/
Search
Print
Home Bookmarks The Mozilla Organiza... Latest Builds
Search and Browse
Action Current query Collections
Submit
Submit personalized
Save to folder
delete
add copy delete
javascript:void(0)

Collections

Name	Action
artiste	
UMIMAGES	

Conditions

Weight	Field	Predicate	Comparison value	Action
700	title	contains	Buddha	
400	subject	contains	Buddha	

CYCLADES © 2001-2003 UNIDO and UNIDUE, BSCW © 1995-2002 FIT and OrbiTeam

Figure 4.1: SB Query Formulation

and 1000), an attribute (according to the current metadata schema), a predicate, and a comparison value.

In the menu bar above the list of conditions, the user can select all, or none, for further actions. Each condition can also be selected or deselected individually via the checkbox on the left. The *Delete* button in the menu bar causes all selected conditions to be removed from the query, the *Copy* button duplicates each selected condition. *Add* ignores selections and simply adds an empty condition to the query.

Main menu

The main menu bar at the top of the query formulation dialogue contains the following entries:

- *Action*
- *Current query*
- *Collections*

The *Action* menu contains actions that replace the current query without saving it:

- *New query* - Forget the current query and start with an empty one
- *Get new records for folder* - Forget the current query and get new records for the current folder instead
- *Select query from history* - Select another query from the session history

- *Select query from folder* - Select another query from the current folder

The *Current query* menu contains actions that refer to the current query:

- *Submit* - Submit the query as is
- *Submit personalized* - Submit the query and filter the results (calls the Filtering and Recommendation Service)
- *Save to folder* - Save the current query to the folder

The *Collections* menu contains actions to add collections to the current query:

- *Add from folder* - Add collections that are associated to the current folder
- *Add from personal set* - Add collections from the set of personal collections
- *Add from system* - Add collections from the set of all collections in the system
- *Use all* - Clear the list of collections in the current query, thus causing all collections in the system to be searched

Collection Selection

Each of the first three actions in the *Collections* menu opens a dialogue with a list of collections. The user can select all, or none, of those collections via the multi-object menu above the list, or individual collections via the checkboxes beside each collection name, and then add the selected collections to the current query by pressing the *Add* button in the menu bar. This will take the user back to the query formulation dialogue, augmented by the additional collections.

Schema Selection

This functionality is still missing from the current Search and Browse GUI. It will be implemented as a *Schema Selection* dialogue later.

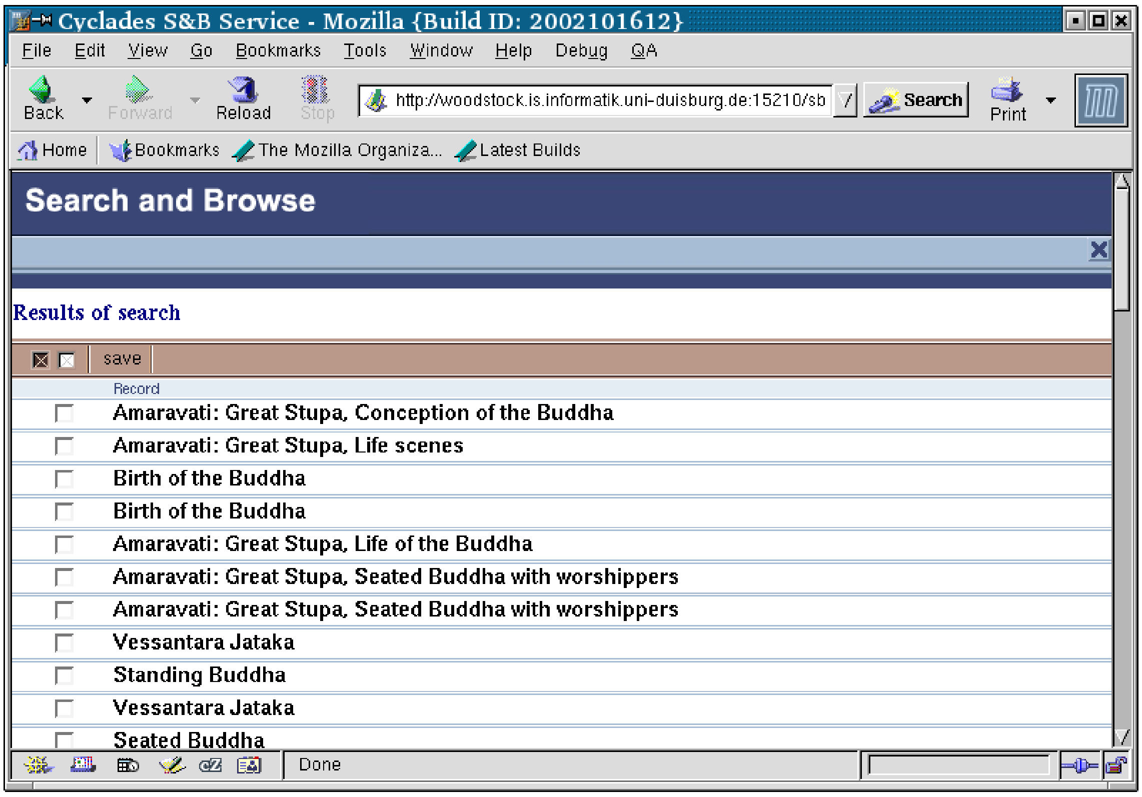
Query Selection

The *Query Selection* dialogue is shown when the user chooses to select a query either from the query history of the current session, or from the list of queries stored in the current folder. Each query is displayed with its name, if it has one, else with an automatically generated ID. By clicking on a query, the user returns to the query formulation dialogue, where the selected query is displayed.

Result List

A *Result List* (figure 4.2) is shown whenever a query was submitted, or the user requested to look for new records for the current folder. It consists of a list of records, and a multi-object menu above the list. Each record is displayed wither with its title, or, if there is no title, with its record id. A click on a record opens a new window which displays the complete metadata record as it was harvested from its archive.

On the left-hand side of each record list entry, there is a checkbox for selecting the record. The user can also select all records, or none, via the multi-object menu. Selected records can be saved to the current folder by pressing the *Save* button in the multi-object menu.


Cyclades S&B Service - Mozilla {Build ID: 2002101612}
File Edit View Go Bookmarks Tools Window Help Debug QA
Back Forward Reload Stop
http://woodstock.is.informatik.uni-duisburg.de:15210/sb
Search Print
Home Bookmarks The Mozilla Organiza... Latest Builds
Search and Browse
save
Done

Results of search

Record
- Amaravati: Great Stupa, Conception of the Buddha
- Amaravati: Great Stupa, Life scenes
- Birth of the Buddha
- Birth of the Buddha
- Amaravati: Great Stupa, Life of the Buddha
- Amaravati: Great Stupa, Seated Buddha with worshippers
- Amaravati: Great Stupa, Seated Buddha with worshippers
- Vessantara Jataka
- Standing Buddha
- Vessantara Jataka
- Seated Buddha

Figure 4.2: SB Result List

4.3.2 SBS GUI test plan

The search and browse interface should provide support for the use cases described in Deliverable D3.0.1. This test plan is based on these use cases, and tests the functionality of the user interface by performing the necessary interactions. It will be tested if each interaction with the User Interface results in the expected behaviour and has no side effects.

The search and browse GUI should provide dialogs for the indivual steps of the search and browse process:

- Select Collections
  - Select Collection
    Within the collection selection dialog the user should be able to select from a list of collections those that shall be queried. The list presented is either
    - a list of all personal collections,
    - a list of collections associated with the user's current folder, or
    - a list of all collections.

    In order to support the user, the interface should allow for the following actions:
    - adding of one or more collections at a time
    - removing of one or more collections at a time
    - searching the whole system (no collection restriction)

    Check:
    - adding new collections should leave the previously chosen ones intact

    - adding or removing collections should not change query conditions already specified
    - adding a collection twice mustn't result in a double entry for that collection
  - Browse Search Schemas

    The metadata browsing should present a list of available metadata schemas for a collection. For one of these schemas at a time the possible attributes can be shown, from which one can be selected to show specific information about it, e.g. its subfields or type.

    Three main actions should be possible in this dialog:

    - browsing the metadata schema
    - selecting one schema for the active query
    - aborting the browsing without changing the query
  - Browse Attribute Values

    The values for a specific attribute or field should be provided in a manner that enables browsing through the entire list in a user friendly way. Three actions should be possible in this dialog:

    - browsing through the value list
    - selecting one value to be used in a condition of the current query
    - aborting the browsing without changing the query

- Edit a Query
  - Edit Query

    Within the query formulation dialog the user should be able to perform the following actions in order to create a satisfactory query:

    - adding a new condition
    - removing a given condition
    - editing a condition by chosing from fields and predicates
    - editing a condition by entering condition weights and values
    - resetting the query (clearing everything)
    - submitting the query

    Check:

    - the conditions that are not manipulated should remain unaffected
    - already chosen collections should remain unaffected
    - invalid entries in the free text fields should be caught and result in appropriate warning
  - Save Query

    The save query dialog allows for the saving of the current query in the user's active folder. There needs to be a dialog for giving the query a human readable name.

    Check:

    - after the dialog ends the query should remain unchanged
    - if the dialog is canceled the folder should not be changed
  - Select Previous Queries

    Within the query selection dialog the user can select from a list of previously submitted queries, if any. At most one query can be selected from this list and then be further modified or submitted again as is.

    - Browse Queries from Folder
    - Browse Queries from History

- Submit Query
  - Submit Query without Personalization
    After the query has been constructed and collections selected, this dialog submits the query to the system, and provides the user with a view of the result.
  - Submit Query with Personalization
    As above this submits the query, but uses personalization that depends on the current user and the active folder. After the query finishes, this dialog provides the user with a view of the result.
  - Browse and Save Results
    This dialog consists mainly of a list of result records. Within this list it should be possible to
    - choose from a list of available attributes a number that will be shown for each record
    - select one or more records
    - save the current selecting to the active folder
- Get New Records
  - Get New Records For Folder
    Ignoring the current query, this dialog allows to submit a search request based on the profile of the current folder. As with query submissing, after the request has finished, a view of the result records should be presented.
  - Browse and Save Results
    This dialog consists mainly of a list of result records. Within this list it should be possible to
    - choose from a list of available attributes a number that will be shown for each record
    - select one or more records
    - save the current selecting to the active folder

4.3.3 SBS GUI test results

- **Tester:** Sascha Kriewel (sascha.kriewel@uni-duisburg.de)
- **Test date:** March 20th, 2003
- **Scope of test:** Complete GUI of SBS was tested according to the described test plan.
- **Test environment:** The test client was Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.2b) Gecko/20021016, running under Linux
- **Test log:** An excerpt of the test log is presented below. A complete version of the log is available at ...

```
Test 1.1.1

  Description: Adding one collection to query
  Action:
      Menu: Collections -> Add from system
      checked checkbox for collection "caltechcstr"
      Button: Add
  Result:
      OK
  Check:
      - query conditions remained unchanged

Test 1.1.2

  Description: Adding multiple collections to query
  Action:
      Menu: Collections -> Add from system
      checked checkboxes for collections
        "ibiblio", "IR", "CYCLADES"
      Button: Add
  Result:
      OK
  Check:
      - previously selected collections were kept
      - query conditions remained unchanged

Test 1.1.3

  Description: Adding all collections associated with the
               current folder to query
  Action:
      Menu: Collections -> Add from folder
      Button: Select All [x]
         ("caltechcstr", "CCSDarchiveSIC")
      Button: Add
  Result:
      OK
  Check:
      - previously selected collections were kept
      - query conditions remained unchanged
      - no doubling of collection "caltechcstr"

Test 1.1.4

  Description: Removing a collection from query
  Action:
      Object "caltechcstr": Menu: Delete
  Result:
      OK
  Check:
      - other collections weren't affected
      - query conditions remained unchanged
  Comment:
      - same result can be achieved by the actions
        described in 1.2.5

Test 1.1.5

  Description: Removing multiple collections from query
  Action:
      checked checkbox for collections
         "ibiblio", "IR"
      Button: Delete
  Result:
      OK
  Check:
      - other collections weren't affected
      - query conditions remained unchanged

Test 1.1.6

  Description: Removing all collections from query
  Action:
      Button: Select All [x] from Collections view
      Button: Delete
  Result:
      OK
  Check:
      - query conditions remained unchanged

Test 1.1.7

  Description: Using all collections from query
  Action:
      Menu: Collections -> Use all
  Result:
      OK
  Check:
      - query conditions remained unchanged
  Comment:
      - same result can be achieved by the actions
        described in 1.2.6

Test 1.2

  Description: Browsing search schema
  not available at time of testing

Test 1.3

  Description: Browsing attribute values
  not available at time of testing

Test 2.1.1

  Description: Adding a new condition to query
  Action:
      Button: Add from the Conditions view
  Result:
      OK
  Check:
      - previously edited conditions remained the same

Test 2.1.2

  Description: Editing a query
  Action:
      entered weight of "+2.0" in textbox for "weight"
      chose field "title" from selection list
      chose predicate "contains" from selection list
      entered value "retrieval" in textbox for "comparison value"
  Result:
      OK

Test 2.1.3

  Description: Copying a condition of the query
  Action:
      Object "condition 1": Menu: Copy
  Result:
      OK
  Check:
      - all other conditions remained the same

Test 2.1.4

  Description: Removing a condition from query
  Action:
      Object "condition 2": Menu: Delete
  Result:
      OK
  Check:
      - all other conditions remained the same

Test 2.1.5

  Description: Remove multiple conditions from query
  Action:
      checked checkboxes for conditions 3 and 4
      Buttom: Delete from the Conditions view
  Result:
      OK
  Check:
      - all other conditions remained the same

Test 2.1.6

  Description: Resetting the query
  Action:
      Menu: Actions->New query
  Result:
      OK

Test 2.2.1

  Description: Saving the query to folder
  Action:
      Menu: Current query->Save to folder
      in new screen entered "search for 'retrieval'"
         as a title for this query
      Buttom: OK
  Result:
      OK

Test 2.2.2

  Description: Aborting saving the query to folder
  Action:
      Menu: Current query->Save to folder
      in new screen entered "search for 'retrieval'"
         as a title for this query
      Buttom: Cancel
  Result:
      OK
  Check:
      - the query remained unchanged

Test 2.3.1

  Description: Browsing and selecting queries from folder
  Action:
      Menu: Action->Select query from folder
      in new screen clicked on hyperlink named
         "search for 'retrieval'"
  Result:
      OK
  Check:

Test 2.3.2

  Description: Browsing and selecting queries from history
  Action:
      Menu: Action->Select query from history
      in new screen clicked on hyperlink for most recent query
  Result:
      OK
  Check:

Test 3.1

  Description: Submitting a query without personalization
  Action:
      Menu: Current query->Submit
  Result:
      OK

Test 3.2

  Description: Submitting a query with personalization
  Action:
      Menu: Current query->Submit personalized
  Result:
      OK

Test 3.3
```

```
  Description: Browsing the results
  Action:
      clicking on the hyperlink of the first result
  Result:
      OK

Test 3.4

  Description: Saving the results
  Action:
      Buttom: Select All [x]
      Buttom: Save
  Result:
      OK

Test 4.1

  Description: Getting new records for folder
  Action:
      Menu: Actions->Get new records from folder
  Result:
      OK

Test 4.2

  Description: Browsing the results
  Action:
      clicking on the hyperlink for the last shown result
  Result:
      OK

Test 4.3

  Description: Saving the results
  Action:
      checking the checkboxes on several results
      Buttom: Save
  Result:
      OK
```

- **Test summary:** All test case functions worked according to the use case descriptions and produced no errors or inconsistencies.

Chapter 5

FRS Component Test

5.1 Introduction

The Filtering and Recommendation Service (FRS) provide the user with highly flexible and personalized interaction (*content-based*). User's behavior are used by the service to "understand" user's preferences, which is based on *automatically* generation a "profile" of the user and of her interest.

In CYCLADES personalization is archived by implementing two basic mechanisms, filtering and recommendation.

- *Filtering* refers to an activity of personalizing the interaction between user and system based on feedback information provided by the user herself.
- *Recommendation* refers instead to an activity of personalizing the interaction between user and system based on feedback information provided by other users that the system considers "similar" to this user.

The FRS API offers two kind of methods[1]. (a) *Filtering methods*, which based on folder "profile" allow to do both a personalized filtering and get new records. (b) *Administrative methods* which allow update a folder "profile", create/delete folders, add/delete records from a folder, set folder recommendation status, etc.

5.2 FRS API test

5.2.1 FRS API specification

The Filtering and Recommendation Service (FRS) API to be used by CYCLADES services is describer bellow. This API uses the XML-RPC protocol. Methods description contains: method name, signature, description, input/output parameters and calling service.

- **Method:** filteredSearch
  Signature: (record*) filteredSearch (query, maxRecordNo, folderID)
  Description: this method may be invoked in order to filter records, retrieved according to a query, with respect to the profile learned from the folder.
  Parameters:

[1] *Recommendation* is done by FRS in a period basis, i.e. witout an explicit call from CYCLADES system.

Input:	query:	the query according to the syntax specified by the access service. XML-RPC data type: <string>.
	maxRecordNo:	maximal number of records to be retrieved. XML-RPC data type: <int>.
	folderID:	the folder ID with respect to which filtering should be performed. XML-RPC data type: <string>.
Output:	record*:	list of records. XML-RPC data type: <array> of <struct> (Record from CWS specification).

Calling service: SBS

- **Method:** getNewRecords
  Signature: (record*) getNewRecords (query, maxRecordNo, folderID, userID)
  Description: this method may be invoked in order to get new records, retrieved with respect to the profile learned from the folder.
  Parameters:

Input:	query:	the query according to the syntax specified by the access service. XML-RPC data type: <string>.
	maxRecordNo:	maximal number of records to be retrieved. XML-RPC data type: <int>.
	folderID:	the folder ID with respect to which filtering should be performed. XML-RPC data type: <string>.
	userID:	the user ID with respect to which filtering should be performed. XML-RPC data type: <string>.
Output:	record*:	list of records. XML-RPC data type: <array> of <struct> (Record from CWS specification).

Calling service: SBS

- **Method:** updateFolderProfile
  Signature: void updateFolderProfile (folderID)
  Description: this method may be invoked in order to update the folder profile, i.e. to learn the folder profile.
  Parameters:

Input:	folderID:	the folder ID with respect to which filtering should be performed. XML-RPC data type: <string>.

Calling service: CWS

- **Method:** createFolder
  Signature: void createFolder (folderID, userID, recommendationValue)
  Description: this method may be invoked in order to store a new created folder in the FRS rating DB.
  Parameters:

Input:	folderID:	a folder identifier. XML-RPC data type: <string>.
	userID:	a user identifier. XML-RPC data type: <string>.
	recommendationValue:	the value contains the bit encoded recommendation preferences of the folder as describer below in setRecommendationYesNo method. XML-RPC data type: <int>.

Calling service: CWS

- **Method:** destroyFolder
  Signature: void destroyFolder (folderID)
  Description: this method destroy FRS folder information (profile, timestamps, etc.).

Parameters:
Input: folderID: a folder identifier. XML-RPC data type: <string>.
Calling service: CWS

- **Method:** deleteUser
  Signature: void deleteUser (userID)
  Description: this method delete FRS user information.
  Parameters:
  Input: userID: a user identifier. XML-RPC data type: <string>.
  Calling service: MS

- **Method:** addRecord
  Signature: void addRecord (recordID, folderID, userID)
  Description: this method may be invoked in order to update FRS folder information.
  Parameters:
  Input: recordID: a record identifier. XML-RPC data type: <string>.
  folderID: a folder identifier. XML-RPC data type: <string>.
  userID: a user identifier. XML-RPC data type: <string>.
  Calling service: CWS

- **Method:** deleteRecord
  Signature: void deleteRecord (recordID, folderID, userID)
  Description: this method delete a record from a folder and which belong to a user.
  Parameters:
  Input: recordID: a record identifier. XML-RPC data type: <string>.
  folderID: a folder identifier. XML-RPC data type: <string>.
  userID: a user identifier. XML-RPC data type: <string>.
  Calling service: CWS

- **Method:** setRecommendationYesNo
  Signature: void setRecommendationYesNo (folderID, value)
  Description: this method may be invoked in order to activate/disactivate the production of recommendations with respect to a folder.
  Parameters:
  Input: folderID: the folder ID with respect to which filtering should be performed. XML-RPC data type: <string>.
  value: the value contains the bit encoded recommendation preferences of the folder:
  - a zero value stands for no recommendations (default),
  - bit 0 stands for record recommendations,
  - bit 1 stands for user recommendations,
  - bit 2 stands for collection recommendations, and
  - bit 3 stands for community recommendations.
  XML-RPC data type: <int>.
  Calling service: CWS

- **Method:** addRating
  Signature: void addRating (recordID, folderID, userID, ratingValue)
  Description: this method may be invoked in order to store a rating in the FRS rating DB. The rating is specified as in the RMS service.
  Parameters:

Input:		
	recordID:	a record identifier. XML-RPC data type: <string>.
	folderID:	a folder identifier. XML-RPC data type: <string>.
	userID:	a user identifier. XML-RPC data type: <string>.
	ratingValue:	the assigned rating value. XML-RPC data type: <int>.

Calling service: RMS

5.2.2 FRS API test

In the following we list the API methods by calling service indicating test responsibilities for the methods listed.

- **CWS** (FIT)
  - updateFolderProfile
  - createFolder
  - destroyFolder
  - addRecord
  - deleteRecord
  - setRecommendationYesNo
- **MS** (FORTH)
  - deleteUser
- **RMS** (FIT)
  - addRating
- **SBS** (UNIDO)
  - filteredSearch
  - getNewRecords

In the following we summarize the results of the FRS API (v0.5) tests as conducted by CNR and the partners responsible for services calling the FRS API.

FRS API test (CNR)

- **Tester:** Henri Avancini (avancini@iei.pi.cnr.it)
- **Test date:** 18 February, 2003 19:22 GMT
- **Scope of test:** Complete API of FRS v0.5.
- **Test environment:** Test client was a Java[2] application running on Linux (2.4.18-19.8.0), the XML-RPC implementation of Apache XML Project[3].
- **Test plan:** All FRS API methods were tested according to the flow diagram presented below. Basically it is composed by a general loop, which randomly[4] decide methods to be called in each iteration. Potentially all methods could be called on each iteration.

[2]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)
[3]http://xml.apache.org/xmlrpc
[4]http://java.sun.com/j2se/1.4/docs/api/java/util/Random.html

```
\STRUCT{init}{Test FRS API}{%
  \WHILE{continue}{%
    \IF{new\\folder?}%
    \THEN{%
      \ACTION{createFolder}%
    }%
    \ELSE{%
    }%
    \ENDIF%
    \IF{add \&\\set new?}%
    \THEN{%
      \ACTION{addRecord\\addRating\\setRecom.}%
    }%
    \ELSE{%
      \ACTION{deleteRecord}%
    }%
    \ENDIF%
    \ACTION{updateF.Prof.\\filteredSearch\\getNewRecs.}%
    \IF{del\\folder \&\\user?}%
    \THEN{%
      \ACTION{destroyFolder\\deleteUser}%
    }%
    \ELSE{%
    }%
    \ENDIF%
  }%
  \ENDWHILE%
}%
\normalsize
```

Data type identifiers (folder, user, record) are generated sequentially, eg. Folder_01, Folder_02, User_01, Record_01, Record_02, etc. An iteration in which no new folder (user, record) is generated a randomly existing one is selected to call methods. Query is loaded from a file. Both "rating" value and "recommendation" value are randomly generated for each method call as well as the "maximum number of records" to be retrieved.

The main FRS API methods are tested on each iteration ("updateFolderProfile", "filteredSearch" and "getNewRecords"). The other methods are called randomly ("createFolder", "addRecord", "addRating", "setRecommendationYesNo", "deleteFolder", "destroyFolder" and "deleteUser").

- **Test run:** The FRS API test was performed on the server 'http://pc-avancini.iei.pi.cnr.it' which runs the same version of Cyclades FRS as the 'http://project.iei.pi.cnr.it' server. The 'http://pc-avancini.iei.pi.cnr.it' server is not connected to the other services so the FRS API functionality can be tested with no side effects on the other services. The test was performed using adhoc designed Cyclades services (AS, CWS, MS and RMS), which simulate on line versions.

  Calls and results were logged. Log files are availables on line[5]. The file 'frsAPITest_detailed.log' contains for each iteration the iteration counter, specifications of methods called and counter of total FRS API calls. For example:

```
Method called: frsAPITest
```

[5]http://project.iei.pi.cnr.it:8080/FRS/publicLogs/

```
Loop: 350
New Folder created: Folder_260. New User created: User_260.
Recommendation Value: 7
Add record & rating. Record: Record_350. Rating: 4
Update folder profile.-
Filtered search with max.Rec.N.: 12
Get new records with max.Rec.N.: 19
FRS API calls sumatory:2318
```

Correspond to loop n.350 in which a new folder was created with identifier: “Folder_260” ans well as a new user was added to the FRS data bases with identifier: “User_260”. The initial recommendation value was setted to 7 (means 0111 binary, i.e. record, collection and user recommendations allowed). After that a record with identifier: “Record_350” was added and rating was setted. Later on, update folder profile, filtered search and get new records method calls were performed. No record/folder/user was deleted on this loop. After this loop 2318 FRS API calls were successfully completed.

’FRSServer.log’ contains FRS server side log, which correspond to the test period. The file ’frsAPITest.log’ contains console outputs from the client side and error messages. During the complete test run no errors have occurred. The file ’getFolderID.log’ keeps a trace of FRS folders before and after test was performed. Lastly, ’startfrs.sh.log’ file contains console outputs from the server side and error messages.

- **Test summary:** All available FRS API methods were executed and worked according to specification for all sets of input parameters. A total of 3323 FRS API calls have been made. No Errors occured during the test run. The method calls and the results produced by the FRS system were logged and made available online.

FRS API test (FIT)

```
FRS API Test (FIT)
Tester: Wido Wirsam (wido.wirsam@fit.fraunhofer.de)
Date: 25.2.03
Time: 10.38


Method tested: updateFolderProfile
Argument(s)  : "CW665_18629"
Return Value : none

Method tested: updateFolderProfile
Argument(s)  : "CW665_18630"  (Folder-ID does not exist)
Return Value : <Fault 0: 'java.lang.Exception: java.lang.NumberFormatException: '>

Method tested: createFolder
Argument(s)  : "CW665_18630", "CW665_7225", 15
Return Value : none

Method tested: destroyFolder
Argument(s)  : "CW665_18630"
Return Value : none

Method tested: addRecord
Argument(s)  : "Catalog of Apollo Experiment Operations", "CW665_18629", "CWCW665_7225"
```

```
Return Value : none

Method tested: deleteRecord
Argument(s)  : "Catalog of Apollo Experiment Operations", "CW665_18629", "CWCW665_7225"
Return Value : none

Method tested: setRecommendationYesNo
Argument(s)  : "CW665_18629", 15
Return Value : none

Method tested: setRecommendationYesNo
Argument(s)  : "CW665_18629", 0
Return Value : none
```

FRS API test (UNIDO/UNIDUE)

- **Tester:** Sascha Kriewel (kriewel@is.informatik.uni-duisburg.de)
- **Test date:** 12 March 2003
- **Scope of test:**
  Testing FRS API methods
- **Test environment:**
  Test client was a Java[6] application running on Debian Linux, the XML-RPC implementation of Apache XML Project.
- **Test plan:**
  Each method was called first with valid parameters, then with invalid parameters, missing parameters, and wrong parameter types.

  With the valid method calls, the semantic result was checked:
  - *filteredSearch* returned a list of appropriate records
  - *initiateSearch* returned the list of new records for the specified folder

  With the intentionally invalid calls, it was checked whether the method returned the appropriate error code and message.
- **Test log:**
  An excerpt of the last test log is shown below.

```
...

Mar 12, 2003 3:55:11 PM: ServerURL=http://project.iei.pi.cnr.it:4413
Mar 12, 2003 3:55:11 PM: ServerName=service

...

Mar 12, 2003 3:55:11 PM: Method.1=filteredSearch
Mar 12, 2003 3:55:11 PM: comment.1=valid parameters
Mar 12, 2003 3:55:11 PM: flags.1=
```

[6]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)

```
Mar 12, 2003 3:55:11 PM: param.1.1.type=String
Mar 12, 2003 3:55:11 PM: param.1.1.value=
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query SYSTEM
"http://woodstock.is.informatik.uni-duisburg.de:15210/ac/query.dtd">
<query schema="dc">
  <collection-query>
     <condition weight="+" field="description">
        <field-condition predicate="$cw$" value="video"/>
     </condition>
  </collection-query>
</query>
Mar 12, 2003 3:55:11 PM: param.1.2.type=Integer
Mar 12, 2003 3:55:11 PM: param.1.2.value=10
Mar 12, 2003 3:55:11 PM: param.1.3.type=String
Mar 12, 2003 3:55:11 PM: param.1.3.value=CW665_20186
Mar 12, 2003 3:56:09 PM: TestService: filteredSearch returned [
[name=A Tutorial on Authorware,
metadata=<?xml version="1.0" encoding="UTF-8"?>
<GetRecord xmlns="http://www.openarchives.org/OAI/1.1/OAI_GetRecord"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openarchives.org/OAI/1.1/OAI_GetRecord
http://www.openarchives.org/OAI/1.1/OAI_GetRecord.xsd">

<responseDate>2002-06-19T11:48:36-05:00</responseDate>
<requestURL>http://scholar.lib.vt.edu:80/theses/OAI/cgi-bin/index.pl?
verb=GetRecord&amp;identifier=oai:VTETD:etd-18409759651581&amp;
metadataPrefix=oai_dc</requestURL>

<record>
<header>
<identifier>oai:VTETD:etd-18409759651581</identifier>
<datestamp>1996-04-25</datestamp>
</header>
<metadata>
...
</metadata>
</record>
</GetRecord>
,classifierLabel=,id=AC143_oai_dc_oai:LTRS:NASA-97-81agard-awb]]

Mar 12, 2003 3:56:09 PM: TestService: spent 58047 millis for request

===========================================
Mar 12, 2003 3:56:10 PM: Method.2=filteredSearch
Mar 12, 2003 3:56:10 PM: comment.2=invalid folder ID
Mar 12, 2003 3:56:10 PM: flags.2=

Mar 12, 2003 3:56:10 PM: param.2.1.type=String
Mar 12, 2003 3:56:10 PM: param.2.1.value=
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query SYSTEM
"http://woodstock.is.informatik.uni-duisburg.de:15210/ac/query.dtd">
<query schema="dc">
```

```
  <collection-query>
     <condition weight="+" field="description">
        <field-condition predicate="$cw$" value="video"/>
     </condition>
  </collection-query>
</query>
Mar 12, 2003 3:56:10 PM: param.2.2.type=Integer
Mar 12, 2003 3:56:10 PM: param.2.2.value=10
Mar 12, 2003 3:56:10 PM: param.2.3.type=String
Mar 12, 2003 3:56:10 PM: param.2.3.value=bla
Mar 12, 2003 3:57:05 PM: TestService: filteredSearch returned []
Mar 12, 2003 3:57:05 PM: TestService: spent 55743 millis for request

===========================================
Mar 12, 2003 3:57:05 PM: Method.3=filteredSearch
Mar 12, 2003 3:57:05 PM: comment.3=missing parameter
Mar 12, 2003 3:57:05 PM: flags.3=

Mar 12, 2003 3:57:05 PM: TestService: XML-RPC Fault #0 -
             java.lang.NoSuchMethodException: filteredSearch
Mar 12, 2003 3:57:05 PM: TestService: spent 118 millis for request

...

Mar 12, 2003 3:57:05 PM: Method.5=getNewRecords
Mar 12, 2003 3:57:05 PM: comment.5=getNewRecords with valid parameters
Mar 12, 2003 3:57:05 PM: flags.5=

Mar 12, 2003 3:57:05 PM: param.5.1.type=String
Mar 12, 2003 3:57:05 PM: param.5.1.value=
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE query SYSTEM
"http://woodstock.is.informatik.uni-duisburg.de:15210/ac/query.dtd">
<query schema="dc">
  <collection-query>
     <condition weight="+" field="description">
        <field-condition predicate="$cw$" value="video"/>
     </condition>
  </collection-query>
</query>
Mar 12, 2003 3:57:05 PM: param.5.2.type=Integer
Mar 12, 2003 3:57:05 PM: param.5.2.value=10
Mar 12, 2003 3:57:05 PM: param.5.3.type=String
Mar 12, 2003 3:57:05 PM: param.5.3.value=CW665_20186
Mar 12, 2003 3:57:05 PM: param.5.4.type=String
Mar 12, 2003 3:57:05 PM: param.5.4.value=CW665_20164
Mar 12, 2003 3:58:10 PM: TestService: getNewRecords returned [
[name=A Tutorial on Authorware,metadata=
<?xml version="1.0" encoding="UTF-8"?>
<GetRecord xmlns="http://www.openarchives.org/OAI/1.1/OAI_GetRecord"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openarchives.org/OAI/1.1/OAI_GetRecord
http://www.openarchives.org/OAI/1.1/OAI_GetRecord.xsd">
<responseDate>2002-06-19T11:48:36-05:00</responseDate>
<requestURL>http://scholar.lib.vt.edu:80/theses/OAI/cgi-bin/index.pl?
verb=GetRecord&amp;identifier=oai:VTETD:etd-18409759651581&amp;
metadataPrefix=oai_dc</requestURL>

<record>
<header>
<identifier>oai:VTETD:etd-18409759651581</identifier>
<datestamp>1996-04-25</datestamp>
</header>
<metadata>
...
</metadata>
</record>
</GetRecord>
,classifierLabel=,id=AC143_oai_dc_oai:LTRS:NASA-97-81agard-awb]]
Mar 12, 2003 3:58:10 PM: TestService: spent 64336 millis for request

...

Mar 12, 2003 3:58:19 PM: Method.9=getNewRecords
Mar 12, 2003 3:58:19 PM: comment.9=wrong parameter type
Mar 12, 2003 3:58:19 PM: flags.9=

Mar 12, 2003 3:58:19 PM: param.9.1.type=Integer
Mar 12, 2003 3:58:19 PM: param.9.1.value=10
Mar 12, 2003 3:58:19 PM: param.9.2.type=Integer
Mar 12, 2003 3:58:19 PM: param.9.2.value=10
Mar 12, 2003 3:58:19 PM: param.9.3.type=String
Mar 12, 2003 3:58:19 PM: param.9.3.value=CW665_7880
Mar 12, 2003 3:58:19 PM: param.9.4.type=String
Mar 12, 2003 3:58:19 PM: param.9.4.value=CW665_7853
Mar 12, 2003 3:58:19 PM: TestService: XML-RPC Fault #0 -
       java.lang.NoSuchMethodException: getNewRecords
Mar 12, 2003 3:58:19 PM: TestService: spent 98 millis for request
```

- **Test summary:**

  All methods specified above from the FR API were tested. All methods worked for valid parameters. Invalid parameters returned empty results. Missing parameters or wrong parameter types resulted in Java exceptions.

Chapter 6

CS Component Test

6.1 Introduction

The Collection Service (CS) manages and stores collections and it's responsible for the dissemination of information about them.

Collection is a mechanisms for dynamically structuring the overall information space into meaningful (from some community's perspective) collections.

The CS has an API supplying 13 methods as well as a graphical user interface (GUI) with a set of functionality about collections management described in Deliverable D3.0.1.

6.2 CS API test

6.2.1 CS API specification

The Collection Service provides an API to other CYCLADES services. The methods of this API may be called using the inter-service communication protocol XML-RPC.

Invoking the Collection Service's methods such default exception can be thrown:

10000	no such method	If the method invoked isn't defined.
10001	bad number of parameters	If you invoke a method with a wrong number of parameters.
10002	bad parameter type	If you invoke a method with a wrong parameter type.
10010	internal error	In all cases where an *undefined* exception exists.

- **Method:** addCollection
  Signature: collectionId addCollection()
  Description: this method creates a new collection identifier which can be assigned to a collection which will be soon created.
  Parameters:
  Output: collectionId integer the identifier of the new collection that will be created.

- **Method:** initializeCollection
  Signature: collectionId initializeCollection(collectionId, collectionName, collectionDescription, membershipCondition, userId)
  Description: this method creates a collection, which parent collection is the *Cyclades* collection , if the membership condition is legal.
  Parameters:

Input:	collectionId	string	the identifier of the new collection.
	collectionName	string	the printable name of the collection (max 50 chars).
	collectionDescription	string	textual description of the collection.
	membershipCondition	string	the condition to be verified by all the members of the collection coded in XML (see 6.2.2).
	userId	string	the identifier of the user which sends the request.
Output:	collectionId	string	the identifier of the collection that has been initialized.

Exception:

10003	missing or null parameter value	if `collectionName` or `collectionDescription` are "".
14112	User doesn't exists	
10200	no permission	operation not allowed, user isn't enable to do it
14113	Identifier isn't valid	`collectionId` isn't valid.
14107	Bad XML file	error parsing `membershipCondition`.
14115	Out of bounds	if `collectionName` is out of bounds.

- **Method:** initializeCollection
  Signature: collectionId initializeCollection(collectionId, collectionName, collectionDescription, membershipCondition, userId, parentCollection)
  Description: this method creates a collection whose parent collection is parentCollection, if the membership condition is legal.
  Parameters:

Input:	collectionId	string	the identifier of the new collection.
	collectionName	string	the printable name of the collection (max 50 chars).
	collectionDescription	string	textual description of the collection.
	membershipCondition	string	the condition to be verified by all the members of the collection coded in XML (see 6.2.2).
	userId	string	the identifier of the user which sends the request.
	parentCollection	string	the identifier of the parent collection in the collection hierarchy.
Output:	collectionId	string	the identifier of the collection that has been initialized.

Exception:

10003	missing or null parameter value	if `collectionName` or `collectionDescription` are "".
14112	User doesn't exists	
10200	no permission	operation not allowed, user isn't enable to do it
14113	Identifier isn't valid	`collectionId` isn't valid.
14105	Collection doesn't exists	`parentCollection` doesn't exists.
14107	Bad XML file	error parsing `membershipCondition`.
14115	Out of bounds	if `collectionName` is out of bounds.

- **Method:** deleteCollection
  Signature: void deleteCollection(collectionId, userId)
  Description: this method removes a collection from the set of existing collections if: a) the user is authorized to do it and b) the specified collection exists.
  Parameters:

Input:	collectionId	string	the identifier of the new collection.
	userId	string	the identifier of the user which sends the request.

Exception:

10200	no permission	operation not allowed, user isn't enable to do it
14105	Collection doesn't exists	`collectionId` doesn't exists.

- **Method:** addSearchBrowseFormat
  Signature: void addSearchBrowseFormat(collectionId, subschema, userId)
  Description: this method adds a new search/browse format if: a) the user is authorized to do it, b) the subschema is legal.

Parameters:

Input:	collectionId	string	the identifier of the collection.
	subschema	string	the specification of the subschema (coded in XML – see 6.2.2 –) used for querying, browsing and displaying results.
	userId	string	the identifier of the user who sends the request.

Exception:

14107	Bad XML file	error parsing `subschema`.
14109	Search&Browse Format already exists.	
10200	no permission	operation not allowed, user isn't enable to do it
14110	Schema doesn't exists.	
14108	Attribute doesn't exists.	
14105	Collection doesn't exists	`collectionId` doesn't exists.

- **Method:** removeSearchBrowseFormat
  Signature: void removeSearchBrowseFormat(collectionId, subschemaName, userId)
  Description: this method removes a search/browse format if: a) the user is authorized to do it, b) the name of the subschema identifies an existing format.
  Parameters:

Input:	collectionId	string	the identifier of the collection from which the search/browse format has to be removed.
	subschemaName	string	the name of the search/browse format to remove.
	userId	string	the identifier of the user who sends the request.

  Exception:

14106	Search&Browse Format doesn't exists.	
10200	no permission	operation not allowed, user isn't enable to do it
14105	Collection doesn't exists	`collectionId` doesn't exists.

- **Method:** listCollections
  Signature: (collectionId, collectionName, collectionDescription, parentCollection)* listCollections(userId)
  Description: this method returns the list of existing collections whose owner is userId.
  Parameters:
  Input: userId string the identifier of the user who sends the request
  Output: a list of (collectionId, collectionName, collectionDescription, parentCollection) where:

collectionId	string	the identifier of the collection.
collectionName	string	the name of the collection.
collectionDescription	string	the description of the collection.
parentCollection	string	the identifier of the parent. collection .

  Exception:
  14112 User doesn't exists

- **Method:** listCollections
  Signature: (collectionId, collectionName, collectionDescription, parentCollection)* listCollections()
  Description: this method returns the list of existing collections.
  Parameters:
  Output: a list of (collectionId, collectionName, collectionDescription, parentCollection) where:

collectionId	string	the identifier of the collection.
collectionName	string	the name of the collection.
collectionDescription	string	the description of the collection.
parentCollection	string	the identifier of the parent collection.

- **Method:** editCollection
  Signature: void editCollection(collectionMetadata,userId)
  Description: Update collection metadata description.
  Parameters:

Input:

collectionMetadata	string	new collection metadata coded in XML (see 6.2.2).
userId	string	the identifier of the user who sends the request.

Exception:

14107	Bad XML file	error parsing `collectionMetadata`.
10200	no permission	operation not allowed, user isn't enable to do it
14105	Collection doesn't exists	

- **Method:** getCollectionMetadata
  Signature: (collectionId, collectionMetadata)* getCollectionMetadata(collectionIds*)
  Description: for each specified collection identifier, this method returns the corresponding descriptive metadata.
  Parameters:
  Input: collectionIds string* a list of collection identifiers.
  Output: A list of pairs (collectionId,collectionMetadata) where:

collectionId	string	the identifier of the collection .
collectionMetadata	string	the collection metadata coded in XML (see 6.2.2).

- **Method:** getPersonalCollections
  Signature: (collectionId, collectionName, collectionDescription, parentCollection)* getPersonalCollections(userId)
  Description: this method returns the list of personal set of collections for user `userId`.
  Parameters:
  Input: userId string the identifier of the user who sends the request
  Output: a list of (collectionId, collectionName, collectionDescription, parentCollection) where:

collectionId	string	the identifier of the collection.
collectionName	string	the name of the collection.
collectionDescription	string	the description of the collection.
parentCollection	string	the identifier of the parent. collection .

  Exception:
  14112 User doesn't exists

- **Method:** deleteUser
  Signature: void deleteUser(userId)
  Description: Notify the Collection Service that user `userId` was removed.
  Parameters:
  Input: userId string the identifier of the user.
  Exception:
  14112 User doesn't exists

- **Method:** deleteArchive
  Signature: void deleteArchive(archiveId)
  Description: Notify the Collection Service that archive `archiveId` was removed.
  Parameters:
  Input: archiveId string the identifier of the archive.

6.2.2 XML objects: XML schemas

`Collection Metadata` **Schema**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
        targetNamespace="http://project.iei.pi.cnr.it:8080/CollectionService"
        xmlns:query="http://project.iei.pi.cnr.it:8080/CollectionService"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:cs="http://project.iei.pi.cnr.it:8080/CollectionService"
```

```
      elementFormDefault="unqualified" attributeFormDefault="unqualified">
  <xs:include schemaLocation="query.xsd"/>
  <xs:include schemaLocation="membership.xsd"/>
  <xs:element name="CollectionMetadata" type="cs:collectionMetadataType">
  </xs:element>
  <xs:complexType name="collectionMetadataType">
    <xs:sequence>
      <xs:element name="Id" type="xs:string"/>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Description" type="xs:string"/>
      <xs:element name="OwnerId" type="xs:string"/>
      <xs:element name="ParentCollection" type="xs:string"/>
      <xs:element name="MembershipCondition" type="cs:MCType"/>
      <xs:element name="FilteringCondition" type="query:FCType"/>
      <xs:element name="Schema" type="cs:schemaType" maxOccurs="unbounded"/>
      <xs:element name="Subschema" type="cs:subschemaType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="FCType">
    <xs:sequence>
      <xs:element name="query">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="collection-query" type="query:collection-queryType"
                minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="schema" type="xs:string" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="schemaType">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="URL" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="subschemaType">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="URL" type="xs:string"/>
      <xs:element name="Schema" type="xs:string"/>
      <xs:element name="Attribute" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

`Membership Condition` **Schema**

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema
        targetNamespace="http://project.iei.pi.cnr.it:8080/CollectionService"
        xmlns:cs="http://project.iei.pi.cnr.it:8080/CollectionService"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="unqualified" attributeFormDefault="unqualified">
  <xs:element name="MembershipCondition" type="cs:MCType"/>

  <xs:complexType name="MCType">
    <xs:sequence>
      <xs:element name="metadataFormat" type="xs:string"/>
      <xs:element name="condition" type="cs:conditionType" maxOccurs="unbounded"/>
      <xs:element name="archive" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="conditionType">
    <xs:attribute name="weight" type="xs:string"/>
    <xs:attribute name="field" type="xs:string" use="required" />
    <xs:attribute name="predicate" type="xs:string" use="required" />
    <xs:attribute name="value" type="xs:string" use="required" />
  </xs:complexType>

</xs:schema>
```

`Subschema` **Schema**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="subschema">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="url" type="xs:string"/>
        <xs:element name="schema" type="xs:string"/>
        <xs:element name="attribute">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="name" type="xs:string"/>
            <xs:element name="datatype" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

`Query` **Schema**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
        targetNamespace="http://project.iei.pi.cnr.it:8080/CollectionService"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:query="http://project.iei.pi.cnr.it:8080/CollectionService"
        elementFormDefault="unqualified" attributeFormDefault="unqualified">
  <xs:complexType name="archiveType">
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="collection-queryType">
    <xs:sequence>
      <xs:element name="condition" type="query:queryConditionType"
          minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="archive" type="query:archiveType"
          minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="queryConditionType">
    <xs:sequence>
      <xs:element name="field-condition" type="query:field-conditionType"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="weight" type="xs:string"/>
    <xs:attribute name="field" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="field-conditionType">
    <xs:attribute name="subfield" type="xs:string"/>
    <xs:attribute name="predicate" type="xs:string" use="required"/>
    <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:element name="query">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="collection-query" type="query:collection-queryType"
            minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="schema" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

6.2.3 CS API test

In the following we list the API methods by calling services indicating test responsabilities for the methods listed.

- **AS** (UNIDO)
  - addCollection
  - initializeCollection
  - deleteArchive
- **CWS** (FIT)
  - listCollection
  - getPersonalCollections
- **SBS** (UNIDO)

  - getCollectionMetadata

- **MS** (FORTH)
  - deleteUser

In the following we summarize the results of the CS API tests as conducted by CNR and the partners responsible for services calling the CS API.

CS API test (CNR)

- **Tester:** Leonardo Candela (`candela@iei.pi.cnr.it`)
- **Test date:** 10 February 2003
- **Scope of test:** API testing using a Java program which tested every method of the API with a number of parameter sets that were read from a test data file.
- **Test environment:** Test client was a Java 1.3.1 program running on Windows 2000 making use of Apache XML-RPC implementation[1] .
- **Test plan:** The first step necessary to test the CS API is to create some dummy collections[2] that populate the CYCLADES system. Some of this collections are created in order to "simulate" the creation of collection in the case of new archive registration. In order to create a collection the testing procedure calls the method `addCollection` and then the method `initializeCollection`. File `0_createCalls.txt` contains the set of calls (method+parameters) used in this phase. This file looks like this:

```
call_0 = initializeCollection
param_0_0 = collection name
param_0_1 = collection description
param_0_2 = membership condition defined as an XML file
param_0_3 = creator identifier
```

  The second step is to call a set of collection managing methods. This methods are `editCollection`, `getCollectionMetadata`, `addSearchBrowseFormat` and `removeSearchBrowseFormat`. Each method is called 10 times and the choice of the collection to use is made randomly among that created[3]. File `1_manageCalls.txt` contains the set of parameters used in this phase. This file looks like this:

```
call_0 = editCollection                 => collectionId random
...
call_10 = getCollectionMetadata
param_10_0 = all                        => all collections
...
call_11 = getCollectionMetadata         => collectionId random
...
call_21 = addSearchBrowseFormat         => collectionId random
param_21_0 = searchBrowse1
...
call_32 = removeSearchBrowseFormat      => collectionId random
```

[1]`http://ws.apache.org/xmlrpc/`
[2]File `http://project.iei.pi.cnr.it:8080/CollectionService/publicLogs/test.zip` contains all input/output files used in this test.
[3]Other data like `ownerId` are acquired from CS system.

```
param_32_0 = searchBrowse1
...
```

Moreover in this phase is tested the method `listCollections`.

The third step is related to testing method `getPersonalCollections`. In order to test this methods the tester have manually select the personal set of collections using the GUI for each user. Then the method is invoked for each user created for this test as reported in the file `2_listPersonalCalls.txt`.

The fourth step is related to testing methods `deleteUser`, `deleteArchive` and `delete-Collection`. As reported in files `3_clearArchsUsers.txt` and `3_clearCollections.txt` all archives, users and collections created for this test will be removed.

- **Test run:** The CS API test was performed on the server `http://pc-candela.iei.pi.cnr.it` which runs the same version of CYCLADES CS. The `http://pc-candela.iei.pi.cnr.it` server isn't really connected to other CYCLADES services, but simulate them.

  Tester have follows procedure steps reported in test plan. For each step a log-file and an error-log file are produced.

  This is an excerpt of the file `CS_testRun_0.log`(the log-file generated during the first step of this test).

```
CS run log (0_createCalls.txt) - created Mon Feb 10 12:23:44 CET 2003

CALL Method service.addCollection
RESP CO1_Coll8673066

CALL Method service.initializeCollection
   Param  CO1_Coll8673066
   Param  collectionTest Archive1
   Param  collectionTest Archive1 Description
   Param  <?xml version="1.0" encoding="iso-8859-1"?><MembershipCondi ...
   Param  TestUser1
RESP CO1_Coll8673066
```

- **Test summary:** All methods worked according to specification for all sets of input parameters. All together 114 API-calls have been made. The method calls and the results produced by the CS system were logged and made available online.

CS API test (FIT)

```
CS API Test (FIT)
Tester: Wido Wirsam (wido.wirsam@fit.fraunhofer.de)
Date: 25.2.03
Time: 10.51

Method tested: listCollections
Argument(s)  :
Return Value :

[['CO563_Coll10843', 'AIM25', 'Archive AIM25', 'CO_CollCYCLADES'],
['CO563_Coll1158', 'ArchiveLyon2', 'Archive ArchiveLyon2',
'CO_CollCYCLADES'],

['CO563_Coll23241', 'CCSDJeanNicod', 'Archive CCSDJeanNicod',
'CO_CollCYCLADES'],

['CO563_Coll29930', 'CCSDarchiveSIC', 'Archive CCSDarchiveSIC',
'CO_CollCYCLADES'],

['CO563_Coll84032', 'CCSDthesis', 'Archive CCSDthesis',
'CO_CollCYCLADES'],

['CO563_Coll23113', 'CDLCIAS', 'Archive CDLCIAS', 'CO_CollCYCLADES'],

['CO563_Coll72335', 'CPS', 'Archive CPS', 'CO_CollCYCLADES'],

['CO563_Coll8005', 'CS', 'Computer science archives',
'CO_CollCYCLADES'],

['CO563_Coll53270', 'CSTC', 'Archive CSTC', 'CO_CollCYCLADES'],

['CO_CollCYCLADES', 'CYCLADES', 'CYCLADES super-collection', ''],

['CO563_Coll65160', 'Clay Animation', 'the most analog way to create
animated Films', 'CO_CollCYCLADES'],

['CO563_Coll60371', 'DBMS papers', 'DBMS(DB Management Systems)
papers', 'CO_CollCYCLADES'],

['CO563_Coll45955', 'DLCommons', 'Archive DLCommons',
'CO_CollCYCLADES'],

['CO563_Coll23667', 'DUETT', 'Archive DUETT', 'CO_CollCYCLADES'],

['CO563_Coll23606', 'Digital Library papers', 'Digital Library
papers', 'CO_CollCYCLADES'],

['CO563_Coll49044', 'FullTex', 'full text collection',
'CO_CollCYCLADES'],

['CO563_Coll83703', 'Fusion', 'Collection about fusion',
'CO_CollCYCLADES'],

['CO563_Coll29823', 'GenericEPrints', 'Archive GenericEPrints',
'CO_CollCYCLADES'],

['CO563_Coll498', 'JTRS', 'Archive JTRS', 'CO_CollCYCLADES'],

['CO563_Coll86068', 'LSUETD', 'Archive LSUETD', 'CO_CollCYCLADES'],

['CO563_Coll82242', 'LTRS', 'Archive LTRS', 'CO_CollCYCLADES'],

['CO563_Coll16645', 'MathPreprints', 'Archive MathPreprints',
'CO_CollCYCLADES'],
['C0563_Coll7610', 'NACA', 'Archive NACA', 'C0_CollCYCLADES'],

['C0563_Coll11789', 'Papers about logic', 'A collection of papers
about logic', 'C0_CollCYCLADES'],

['C0563_Coll96646', 'Query optimization papers', 'Query optimization
papers', 'C0563_Coll60371'],

['C0563_Coll62217', 'RIACS', 'Archive RIACS', 'C0_CollCYCLADES'],

['C0563_Coll5579', 'StoriaCollection', 'collezione di record su
"storia"', 'C0_CollCYCLADES'],

['C0563_Coll74166', 'Test1', 'only test again', 'C0_CollCYCLADES'],

['C0563_Coll76527', 'Test2', 'only a test', 'C0563_Coll29930'],

['C0563_Coll97398', 'UMIMAGES', 'Archive UMIMAGES',
'C0_CollCYCLADES'],

['C0563_Coll38469', 'UniversityOfNottingham', 'Archive
UniversityOfNottingham', 'C0_CollCYCLADES'],

['C0563_Coll99485', 'VTETD', 'Archive VTETD', 'C0_CollCYCLADES'],

['C0563_Coll24379', 'ackarch', 'Archive ackarch', 'C0_CollCYCLADES'],

['C0563_Coll5617', 'anlc', 'Archive anlc', 'C0_CollCYCLADES'],

['C0563_Coll11014', 'ans', 'Archive ans', 'C0_CollCYCLADES'],

['C0563_Coll41676', 'anu', 'Archive anu', 'C0_CollCYCLADES'],

['C0563_Coll7907', 'artiste', 'Archive artiste', 'C0_CollCYCLADES'],

['C0563_Coll4594', 'bmc', 'Archive bmc', 'C0_CollCYCLADES'],

['C0563_Coll44215', 'caltechEERL', 'Archive caltechEERL',
'C0_CollCYCLADES'],

['C0563_Coll61013', 'caltechETD', 'Archive caltechETD',
'C0_CollCYCLADES'],

['C0563_Coll52651', 'caltechcstr', 'Archive caltechcstr',
'C0_CollCYCLADES'],

['C0563_Coll32500', 'cav2001', 'Archive cav2001', 'C0_CollCYCLADES'],

['C0563_Coll1421', 'cbold', 'Archive cbold', 'C0_CollCYCLADES'],

['C0563_Coll20887', 'cdlib1', 'Archive cdlib1', 'C0_CollCYCLADES'],

['C0563_Coll85995', 'celebration', 'Archive celebration',
'CO_CollCYCLADES'],

['CO563_Coll59959', 'cimi', 'Archive cimi', 'CO_CollCYCLADES'],

['CO563_Coll34271', 'cogprints', 'Archive cogprints',
'CO_CollCYCLADES'],

['CO563_Coll50088', 'conoze', 'Archive conoze', 'CO_CollCYCLADES'],

['CO563_Coll68048', 'cyclades_fuzzy_logic', 'fuzzy logic',
'CO563_Coll89489'],

['CO563_Coll89489', 'cyclades_logic', 'a collection of papers about
logic.', 'CO_CollCYCLADES'],

['CO563_Coll24509', 'dlpscoll', 'Archive dlpscoll',
'CO_CollCYCLADES'],

['CO563_Coll35942', 'eldorado', 'Archive eldorado',
'CO_CollCYCLADES'],

['CO563_Coll74074', 'epsilondiss', 'Archive epsilondiss',
'CO_CollCYCLADES'],

['CO563_Coll38677', 'epubWU', 'Archive epubWU', 'CO_CollCYCLADES'],

['CO563_Coll32597', 'ethnologue', 'Archive ethnologue',
'CO_CollCYCLADES'],

['CO563_Coll66090', 'formations', 'Archive formations',
'CO_CollCYCLADES'],

['CO563_Coll87517', 'hofprints', 'Archive hofprints',
'CO_CollCYCLADES'],

['CO563_Coll35707', 'hsss', 'Archive hsss', 'CO_CollCYCLADES'],

['CO563_Coll10882', 'ibiblio', 'Archive ibiblio', 'CO_CollCYCLADES'],

['CO563_Coll15553', 'ioffe', 'Archive ioffe', 'CO_CollCYCLADES'],

['CO563_Coll55048', 'lacito', 'Archive lacito', 'CO_CollCYCLADES'],

['CO563_Coll29421', 'lcoa1', 'Archive lcoa1', 'CO_CollCYCLADES'],

['CO563_Coll19644', 'ncstrlh', 'Archive ncstrlh', 'CO_CollCYCLADES'],

['CO563_Coll66644', 'pastel', 'Archive pastel', 'CO_CollCYCLADES'],

['CO563_Coll31033', 'perseus', 'Archive perseus', 'CO_CollCYCLADES'],

['CO563_Coll15673', 'sammelpunkt', 'Archive sammelpunkt',
'CO_CollCYCLADES'],
['CO563_Coll51675', 'test', 'test collection', 'CO_CollCYCLADES'],

['CO563_Coll86770', 'tkn', 'Archive tkn', 'CO_CollCYCLADES']]
```

```
Method tested: getPersonalCollections
Argument(s)  : "CW665_7225"
Return Value :

[['CO563_Coll32597', 'ethnologue', 'Archive ethnologue',
'CO_CollCYCLADES'],

['CO563_Coll60371', 'DBMS papers', 'DBMS(DB Management Systems)
papers', 'CO_CollCYCLADES'],

['CO563_Coll65160', 'Clay Animation', 'the most analog way to create
animated Films', 'CO_CollCYCLADES'],

['CO563_Coll84032', 'CCSDthesis', 'Archive CCSDthesis',
'CO_CollCYCLADES'],

['CO563_Coll96646', 'Query optimization papers', 'Query optimization
papers', 'CO563_Coll60371']]
```

CS API test (UNIDUE)

- **Tester:** Gudrun Fischer (Gudrun.Fischer@uni-duisburg.de)
- **Test date:** 4 March 2003
- **Scope of test:** Testing CS API methods called from AS and SBS.
- **Test environment:** Test client was a Java[4] application running on Debian Linux, the XML-RPC implementation of Apache XML Project.
- **Test plan:**

  Each method was called first with valid parameters, then with invalid parameters of the right type, then with a wrong number of parameters, and finally, with a wrong parameter type.

  With the valid method calls, the semantic result was checked:

  - *addCollection* did return IDs which could be used for creating new collections
  - *initializeCollection* did create the collections as specified
  - *getCollectionMetadata* returned the metadata for all valid collection ids
  - *getPersonalCollections* returned exactly the personal collections of the user
  - *deleteArchive* could not be called twice for the same archive id, and the archive's collection was no longer available afterwards

[4]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)

With the intentionally invalid calls, it was checked whether the method returned the appropriate error code and message.

- **Test log:** An excerpt of the last test log is shown below.

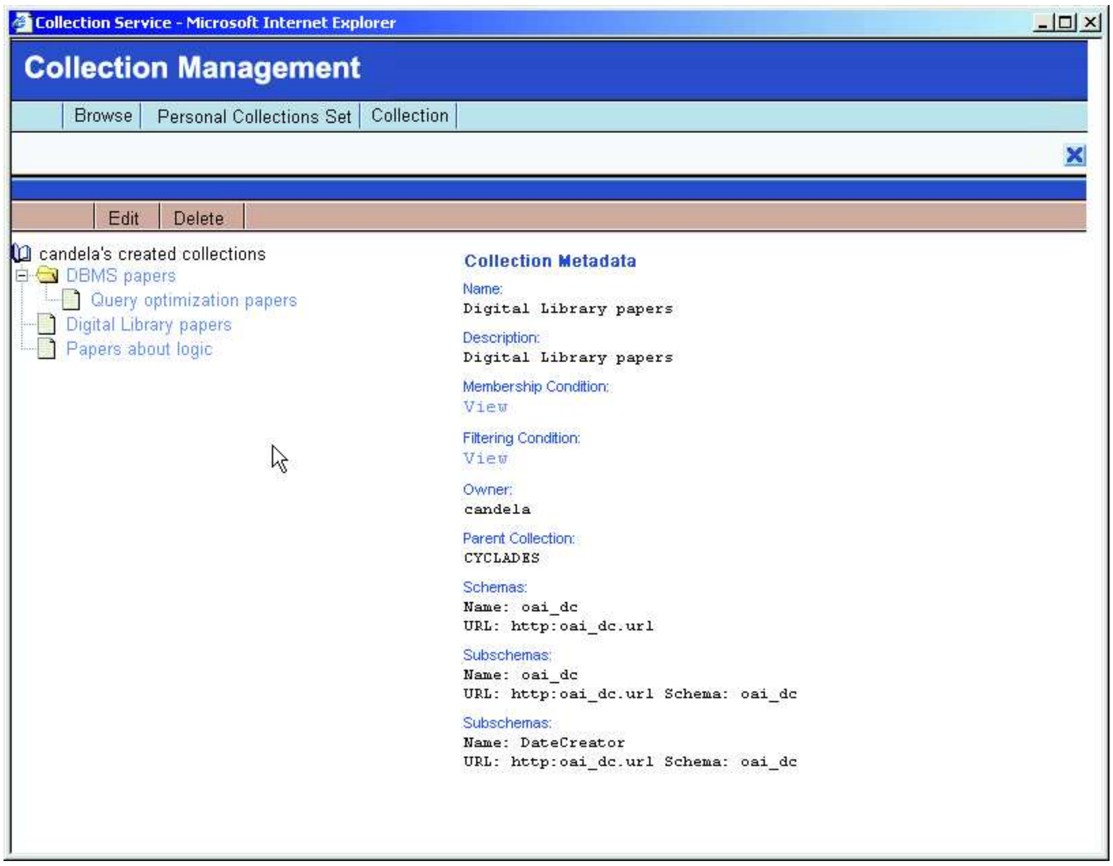
```
...

Mar 4, 2003 8:22:35 PM: Reading from
tests_for_partners/co/CO_initializeCollection.properties
Mar 4, 2003 8:22:35 PM: ServerURL=
http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server
Mar 4, 2003 8:22:35 PM: ServerName=service
Mar 4, 2003 8:22:35 PM: proxyHost=
Mar 4, 2003 8:22:35 PM: proxyPort=

===========================================
Mar 4, 2003 8:22:35 PM: Method.1=initializeCollection
Mar 4, 2003 8:22:35 PM: comment.1=initialize the first collection
Mar 4, 2003 8:22:35 PM: flags.1=

Mar 4, 2003 8:22:35 PM: param.1.1.type=String
Mar 4, 2003 8:22:35 PM: param.1.1.value=CO563_Coll98705
Mar 4, 2003 8:22:35 PM: param.1.2.type=String
Mar 4, 2003 8:22:35 PM: param.1.2.value=test collection 1
Mar 4, 2003 8:22:35 PM: param.1.3.type=String
Mar 4, 2003 8:22:35 PM: param.1.3.value=this is dummy collection no. 1
Mar 4, 2003 8:22:35 PM: param.1.4.type=String
Mar 4, 2003 8:22:35 PM: param.1.4.value=<MembershipCondition>
<metadataFormat>oai_dc</metadataFormat> <archive>celebration</archive>
</MembershipCondition>
Mar 4, 2003 8:22:35 PM: param.1.5.type=String
Mar 4, 2003 8:22:35 PM: param.1.5.value=CW665_7853
Mar 4, 2003 8:22:37 PM: TestService: initializeCollection returned
CO563_Coll98705
Mar 4, 2003 8:22:37 PM: TestService: spent 2155 millis for request

...

Mar 4, 2003 8:22:39 PM: Method.3=initializeCollection
Mar 4, 2003 8:22:39 PM: comment.3=initialize the 3rd collection,
child of itself
Mar 4, 2003 8:22:39 PM: flags.3=

Mar 4, 2003 8:22:39 PM: param.3.1.type=String
Mar 4, 2003 8:22:39 PM: param.3.1.value=CO563_Coll66524
Mar 4, 2003 8:22:39 PM: param.3.2.type=String
Mar 4, 2003 8:22:39 PM: param.3.2.value=test collection 3
Mar 4, 2003 8:22:39 PM: param.3.3.type=String
Mar 4, 2003 8:22:39 PM: param.3.3.value=this is dummy collection no. 3
Mar 4, 2003 8:22:39 PM: param.3.4.type=String
Mar 4, 2003 8:22:39 PM: param.3.4.value=<MembershipCondition>
<metadataFormat>oai_dc</metadataFormat> <archive>AIM25</archive>
</MembershipCondition>
Mar 4, 2003 8:22:39 PM: param.3.5.type=String
Mar 4, 2003 8:22:39 PM: param.3.5.value=CW665_7853
Mar 4, 2003 8:22:39 PM: param.3.6.type=String
Mar 4, 2003 8:22:39 PM: param.3.6.value=CO563_Coll66524
Mar 4, 2003 8:22:39 PM: TestService: XML-RPC Fault #14105 -
Collection doesn't exists
Mar 4, 2003 8:22:39 PM: TestService: spent 593 millis for request

...

Mar 4, 2003 8:36:25 PM: Reading from
tests_for_partners/co/CO_getPersonalCollections.properties
Mar 4, 2003 8:36:25 PM: ServerURL=
http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server
Mar 4, 2003 8:36:25 PM: ServerName=service
Mar 4, 2003 8:36:25 PM: proxyHost=
Mar 4, 2003 8:36:25 PM: proxyPort=

===========================================
Mar 4, 2003 8:36:25 PM: Method.1=getPersonalCollections
Mar 4, 2003 8:36:25 PM: comment.1=valid parameter
Mar 4, 2003 8:36:25 PM: flags.1=

Mar 4, 2003 8:36:25 PM: param.1.1.type=String
Mar 4, 2003 8:36:25 PM: param.1.1.value=CW665_7853
Mar 4, 2003 8:36:26 PM: TestService: getPersonalCollections returned
[[CO563_Coll10843,AIM25,Archive AIM25,CO_CollCYCLADES],
[CO563_Coll85995,celebration,Archive celebration,CO_CollCYCLADES]]
Mar 4, 2003 8:36:26 PM: TestService: spent 784 millis for request

...

Mar 4, 2003 8:46:10 PM: Reading from
tests_for_partners/co/CO_deleteArchive.properties
Mar 4, 2003 8:46:10 PM: ServerURL=
http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server
Mar 4, 2003 8:46:10 PM: ServerName=service
Mar 4, 2003 8:46:10 PM: proxyHost=
Mar 4, 2003 8:46:10 PM: proxyPort=

===========================================
Mar 4, 2003 8:46:10 PM: Method.1=deleteArchive
Mar 4, 2003 8:46:10 PM: comment.1=valid parameter
Mar 4, 2003 8:46:10 PM: flags.1=

Mar 4, 2003 8:46:10 PM: param.1.1.type=String
Mar 4, 2003 8:46:10 PM: param.1.1.value=celebration
Mar 4, 2003 8:46:28 PM: TestService: deleteArchive returned
Mar 4, 2003 8:46:28 PM: TestService: spent 18196 millis for request

...

Mar 4, 2003 8:46:28 PM: Method.4=deleteArchive
Mar 4, 2003 8:46:28 PM: comment.4=archive that has been deleted before
Mar 4, 2003 8:46:28 PM: flags.4=
Mar 4, 2003 8:46:28 PM: param.4.1.type=String
Mar 4, 2003 8:46:28 PM: param.4.1.value=celebration
Mar 4, 2003 8:46:29 PM: TestService: XML-RPC Fault #14116 -
Archive doesn't exists
Mar 4, 2003 8:46:29 PM: TestService: spent 426 millis for request

...
```

- **Test summary:** All methods called from the AS and the SBS were tested. All methods tested worked according to the specification for all sets of correct input parameters. Correct calls did not produce any errors, incorrect calls produced the appropriate error codes and messages.

6.3 CS GUI test

6.3.1 CS GUI description

The graphical user interface of the CS is organized into two areas, the *menu area* and the *working area* as shown in figure 6.1. Menu area contains a *menu bar* (at the upper) and an *action menu*. Working area contains a *collection hierarchy area* and a *collection data area*.



candela's created collections
DBMS papers
Query optimization papers
Digital Library papers
Papers about logic

Collection Metadata
Name:
Digital Library papers
Description:
Digital Library papers
Membership Condition:
View
Filtering Condition:
View
Owner:
candela
Parent Collection:
CYCLADES
Schemas:
Name: oai_dc
URL: http:oai_dc.url
Subschemas:
Name: oai_dc
URL: http:oai_dc.url Schema: oai_dc
Subschemas:
Name: DateCreator
URL: http:oai_dc.url Schema: oai_dc

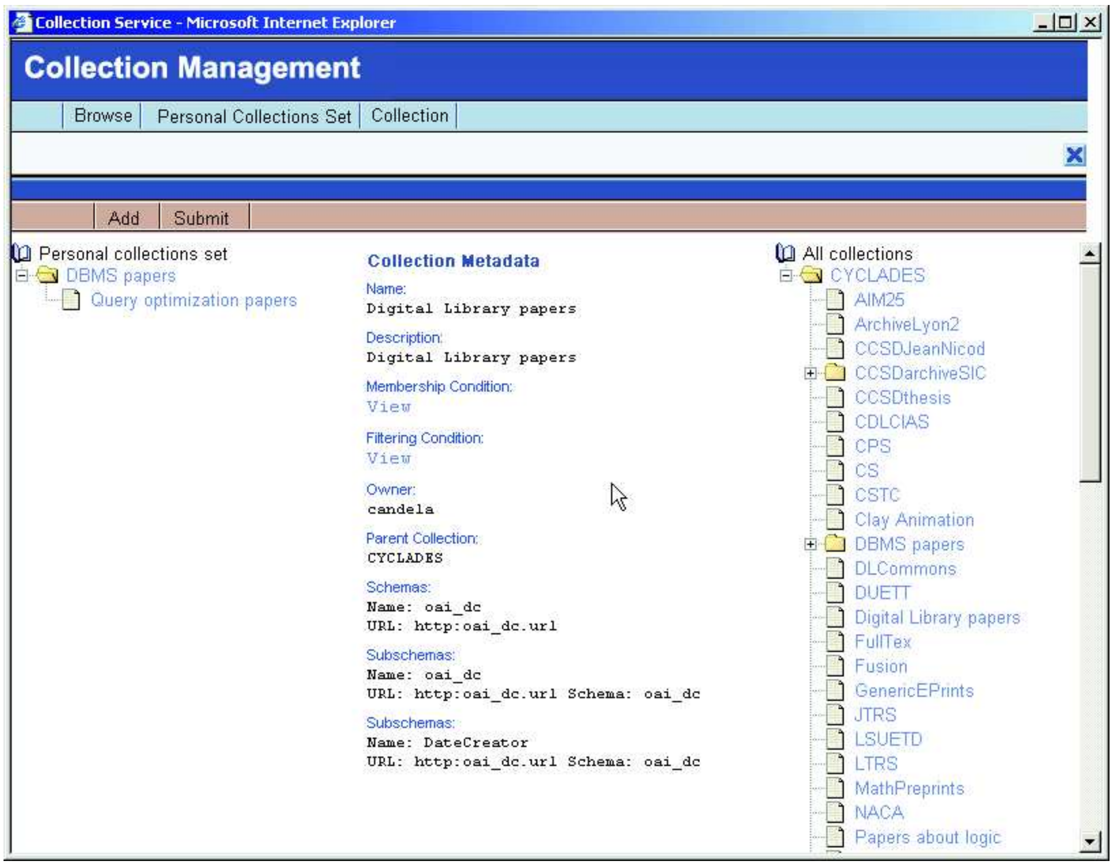
Figure 6.1: CS Graphical User Interface

The menu bar

At the upper of the interface (under the Collection Management title bar) there is a menu bar with three menus and/or action shortcut.

Via the *Browse* menu the user may choice the set of collections shown in the working area among own created collections and all CYCLADES collections.

Via the *Personal Collections Set* shortcut the user can browse/edit his "personal collection set". Figure 6.2 shows the GUI that allows user to manage his personal collection set. This GUI has a working area little bit different from the previous, there are two collections hierarchy areas, one (the left) for the "actual" personal collections set and the other (the right) for all collections. Clicking on a collection in the left area the user can remove this from the actual personal collections set, clicking on a collection in the right area the user can add this from the actual personal collections set. Collection data area in the middle shows collection data (e.g. name, description) for the selected collection.



Collection Metadata

Name:
`Digital Library papers`

Description:
`Digital Library papers`

Membership Condition:
View

Filtering Condition:
View

Owner:
`candela`

Parent Collection:
`CYCLADES`

Schemas:
```
Name: oai_dc
URL: http:oai_dc.url
```

Subschemas:
```
Name: oai_dc
URL: http:oai_dc.url Schema: oai_dc
```

Subschemas:
```
Name: DateCreator
URL: http:oai_dc.url Schema: oai_dc
```

Figure 6.2: CS Graphical User Interface for select the Personal Collections Set

Via *Collection→New* the user can create a new collection.

The action menus

Under the menu bar the CS interface provide an action menu. The items of this menu are related to the collection shown in the collection data area.

If the collection data area shows a collection created by the user than the action menu contains the item *Edit*, in order to edit this collection, and *Delete* in order to delete this collection.

The collections hierarchy area

On the left of the working area there is the collections hierarchy area. In this area there is a navigable hierarchical view of the set of collections actually in use (own created collections or all collections).

Clicking on a collection allows a user to see collection data in the collection data area and, if the user has the rights, to manage them (via the action menu).

The collection data area

On the right of the working area there is the collection data area. This area shows collection data (e.g. name, description) for the selected collection in the collections hierarchy area.

6.3.2 CS GUI test plan

The CS GUI test plan is based on use cases that have been identified for the CS in Deliverable 3.0.1.

The interaction sequence for the test cases may involve intermediary forms where to enter parameters that are necessary for the completion of the test case.

Create a collection

Action: Menu bar `Collection` → `New`
Parameter:

- Collection Name
- Collection Description
- Parent Collection
- Membership Condition

Choice of parent collection is made possible via a combo box that lists all collections in Cyclades. When a parent collection is selected the system update the membership condition with the membership condition of the selected parent collection.

In order to edit the Membership Condition the user has to select a set of archives, eventually none, and to edit a list of condition. The set of archives may be chose via a combo box that lists all archives in Cyclades system and a menu action. Selected archives will be shown on the left side of the combo box. The list of collections are in a tabular representation and can be manipulated via a menu action; adding, deleting and copying of selected conditions are possible. For each condition the user has to specify:

- weight
- field (via a combo box)
- predicate (via a combo box)
- value

Edit a collection

Action: Menu bar `Browse` → `user's created collections` → chose a collection in hierarchical view → Action menu `Edit`
Parameter: new collection description

Delete a collection

Action: Menu bar `Browse` → `user's created collections` → chose a collection in hierarchical view → Action menu `Delete`

Selecting personal collections set

Action: Menu bar `Personal Collections Set`

Browse own created collections

Action: Menu bar `Browse` → `user's created collections`

Browse all Cyclades collections

Action: Menu bar `Browse` → `All collections`

6.3.3 CS GUI test results

- **Tester:** Leonardo Candela (`candela@iei.pi.cnr.it`)
- **Test date:** 13 February 2003
- **Scope of test:** Complete GUI of CS v0.2[5] was tested.
- **Test environment:** Test client was a Internet Explorer 6 running on Windows 2000.
- **Test log:** An excerpt of test log is shown below. The complete test log is available on-line[6].

```
19:31:11 - listCollections - userId=CW665_7131
19:31:17 - addCollection
19:31:17 - listCollections
19:31:43 - listCollections
19:31:44 - initializeCollection
           Id=CO563_Coll3483
           Name=Collection GUI test 1
           Description=Collection GUI test 1 name
           userId=CW665_7131
           parent=CO_CollCYCLADES
19:31:48 - listCollections - userId=CW665_7131
19:31:52 - getCollectionMetadata - [CO563_Coll3483]
19:31:55 - addCollection
19:31:56 - listCollections
19:32:12 - listCollections
19:32:39 - listCollections
19:32:42 - listCollections
19:32:55 - listCollections
19:32:55 - initializeCollection
           Id=CO563_Coll1253
           Name=Collection GUI test 1.1
           Description=Collection GUI test 1.1 description
           userId=CW665_7131
```

[5]`http://project.iei.pi.cnr.it:8080/CollectionService/start?userid=CW665_7131`
[6]`http://project.iei.pi.cnr.it:8080/CollectionService/publicLogs/CSGUItest_2003_02_13.log`

```
          parent=CO563_Coll3483
...
19:58:01 - getCollectionMetadata - [CO563_Coll1065]
19:58:04 - deleteCollection  -  Id=CO563_Coll1065 - userId=CW665_7131
...
```

- **Test summary:** All test case functions worked according to the use case descriptions and produced no errors or inconsistencies.

Chapter 7

MS Component Test

7.1 Introduction

The Mediator Service (MS) integrates and enables the various services of the Cyclades to communicate with each other. The MS stores information about the services and the registered users in Cyclades. The MS is responsible for services' "registration" to Cyclades. Also, services in Cyclades refer to MS in order to get information about other services.

The MS partly handles the registration of the users in Cyclades and fully handles the loggin of the users to the system, via the graphical user interface. Also the MS, in interaction with the Collaborative Work Service, is involved in the cases of user's invitation.

Thus, the MS supports:

- Inter-System Communication,
- User Registration and Login,
- Service Registration.

The MS has an API supplying 13 methods as well as a graphical user interface (GUI) for supporting the users' registration and users' loggin to Cyclades.

7.2 MS API test

7.2.1 MS API specification

The Mediator Service provides an API to the other CYCLADES services. The methods of this API may be called using the inter-service communication protocol XML-RPC. For every method, the services that call this method, according to the service interaction as specified in Deliverable D3.0.1, are also listed.

- (servId, version, address, quality)* getService(type)
  Description: this method is used in order to get a list of services of particular type.

Input:	type:	string	a service type.
Output:			a list of tuples that describe a service (the ID, the version number, the address and the quality of a service).

  Calling service: all

- description getServiceDescription(servId)
  Description: this method is used in order to get the description of a service.

Input:	servId:	string	a service ID.
Output:	description	string	the (short) description of the particular service.

  Calling service: all

- errorLog getErrorLog(servId)
  Description: this method is used in order to get the error log file of a service.

Input:	servId:	string	a service ID.
Output:	errorLog	string	the error log file of the particular service.

  Calling service: all

- void reportError(servId, errorLog)
  Description: this method is used in order to report an error(s) for a service.

Input:	servId:	string	a service ID.
	errorLog:	string	an error to be added to the already existing error log file of the service.

  Calling service: all

- (serviceId, servicePassword) addService(version, address, type, description,URL)
  Description: this method is used in order to add a service to the system.

Input:	version:	string	the version of the service.
	address:	string	the machine address.
	type:	string	the type of the service.
	description:	string	a short description of the service.
	URL:	string	the URL for the GUI of the service.
Output:	a tuple containing		
	serviceId	string	a service ID
	servicePassword	string	a service password (for use in particular methods).

  Calling service: all

- void deleteService(servicePassword)
  Description: this method is used in order to delete/remove a service from the system.

Input:	servicePassword:	string	the service's password.

  Calling service: all

- void updateService(servicePassword, version, address, description, URL)
  Description: this method is used in order to update the information (version, machine address, description, URL) of a service.

Input:	servicePassword:	string	the service's password.
	version:	string	the (new) version of the service.
	address:	string	the (new) machine address.
	description:	string	a (new) short description of the service.
	URL:	string	the (new) URL for the GUI of the service.

  Calling service: all

- void resetErrorLog(servicePassword)
  Description: this method is used in order to reset the error log file of a service.

Input:	servicePassword:	string	the service's password.

  Calling service: all

- void addUser(id, username, password, mailAddr, folderId)
  Description: this method is used in order a registered user of Cyclades to be added to the MS database.

Input:	id:	string	the id of the user to be added (id is generated by CWS).
	username:	string	the username of the user to be added.
	password:	string	the password which the user will use.
	mailAddr:	string	the e-mail address of the user.
	folderId:	string	the id of the user's home folder (folder id is generated by CWS).

  Calling service: CWS

- void deleteUser(id)
  Description: this method is used in order for a registered user of Cyclades to be deleted from the MS only.

Input:	id:	string	the id of the user to be deleted.

  Calling service: all

- void inviteUser(mailAddr, inviteMsg)
  Description: this method is used in order for a registered user of Cyclades to be able to invite another unregister user to the system.

Input:	mailAddr:	string	e-mail address of the invitee.
	inviteMsg:	string	the invitation message to the inviter.

  Calling service: CWS

- UserId* getUserIds()
  Description: this methods is used in order to obtain the ids of all users.

Output:	UserIds:	a list containing the IDs of all users.

  Calling service: all

- (userId, username, mailAddress, homeFolder)* getUserInfo(userIds*)
  Description: this methods is used in order to obtain information about users.

Input:	userIds:	a list containing the IDs of the users we want to obtain information about.
Output:		a list containing information (userId, username, mailAddress, homeFolder) about some users.

  Calling service: all

- **Service**
  - **servId:** the unique ID of the service.
  - **password:** the password of the service.
  - **version:** the version of the service.
  - **address:** the machine address of the service, so as others can ommunicate with it.
  - **quality:** a number between 0-1, indicating the quality of the service.
  - **type:** the type of the service.
  - **dscription:** (short) textual description of the service.
  - **errorLog:** for descriping-holding the various errors that occur.

7.2.2 MS API test

In the following list the API methods are listed by calling service indicating test responsibilities for the methods listed.

- **CWS**
  - addUser
  - invitePasswd
- **all services**
  - getService
  - getServiceDescription
  - getErrorLog
  - reportError
  - addService
  - deleteService
  - updateService
  - resetErrorLog
  - getUserIds
  - getUserInfo

MS API test (FORTH)

MS API test (CNR)

- **Tester:** Leonardo Candela (candela@iei.pi.cnr.it)
- **Test date:** 14 February 2003
- **Scope of test:** Testing MS API v0.2[1]
  methods called from CS.
- **Test environment:** Test client was a Java program running on Windows 2000 making use of
  Apache XML-RPC v1.0 implementation[2].
- **Test plan:** Test plan can be divided in two phases: *acquire users informations*
  and *manage services.*
  The acquire users information phase consists in calling the method `getUserInfo`.
  This method require as parameters a list of valid user identifiers that can be acquired
  via the `getUserIds` method.
  The manage service phase consists in calling, and testing, methods `addService`,
  `updateService`, `getService` and `deleteService`. When a new service is added
  the MS create a service identifier and a service password, this password must be used to manage
  the service (`updateService` and `deleteService`). Moreover services has a
  parameter `type` (e.g. 'CO' for CS, 'CW' for CWS), in this test 'CO' is used.

[1] `http://calliope.ics.forth.gr:8888`

[2] `http://xml.apache.org/xmlrpc`

- **Test log:** An excerpt of test log is shown below. The complete test log is available on-line[3].

```
MS run log - created Fri Feb 14 12:06:14 CET 2003

Method: getUserIds

Parameters:

Result:

   [CW665_7225, CW665_13620, CW665_6971, CW665_7131, ...

...

Method: addService

Parameters:

   1.1

   http://project.iei.pi.cnr.it:8080/CollectionServi...

   CO

   Service created for test API 2

   http://project.iei.pi.cnr.it:8080/CollectionServi...

Result:

   [CO364, COrhz493jl]

...

Method: updateService

Parameters:

   COn0wba8mm

   1.0.1

   http://project.iei.pi.cnr.it:8080/CollectionServi...

   Service created for test API updated

   http://project.iei.pi.cnr.it:8080/CollectionServi...

Result:

   void
```

[3]http://project.iei.pi.cnr.it:8080/CollectionService/publicLogs/MStestAPI.log

```
    ...

    Method: deleteService

    Parameters:

       C0flovd6h8

    Result:

       void

    ...
```

- **Test summary:** All methods tested worked according to specification for all sets of input parameters.

  No errors occurred during the test run.

First test from **CNR**:

```
Report on a ME API v 0.2 Test
=============================

Date 13 September 2002
Author Leonardo Candela - ISTI-CNR - Italy

Test procedure
--------------
All methods were tested manually with a small number of parameter
combinations (see MediatorTestLog.txt) from a Java Client.

Results
-------
- getService             OK but
   getService('aaa') --> []
                         should reply 'bad/invalid service type'
- getServiceDescription  OK
- getErrorLog            OK
- reportError            OK
- getErrorLog            OK
- resetErrorLog          OK
- getUserIds             OK
- getUserInfo            OK
- inviteUser             OK but
   inviteUser('aaa') --> void
                         should reply 'bad/invalid email address'
- addService             OK
- updateService          OK
- deleteService          OK

Other comments
```

```
--------------
None.
```

Second test from **CNR**:

```
Logs on a ME API v 0.2 Test
===========================

Date 13 September 2002
Author Leonardo Candela - ISTI-CNR - Italy

=====================================================

Method: getService
-----------------------------------------------------
Parameters:
   CO
Result:
   [[CO002, 1.0, http://www.address.com, 0.79999995],
[CO176, 0.2, http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server, 1.0]]
-----------------------------------------------------
Parameters:
   ME
Result:
   [[ME000, 0.1, http://calliope.ics.forth.gr:8888, 0.9],
[ME038, 1.0.1, http://abc.xyz.com, 1.0], [ME335, 1.0.1, http://abc.xyz.com, 1.0]]
-----------------------------------------------------
Parameters:
   aaa
Result:
   []
-----------------------------------------------------
Parameters:
   aaa
   bbb
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(getService): got 2,expected 1
-----------------------------------------------------
Parameters:
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(getService): got 0,expected 1
-----------------------------------------------------
Parameters:
   1
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(getService(1)): got java.lang.Integer,expected String
=====================================================

Method: getServiceDescription
```

```
-----------------------------------------------------
Parameters:
   CO176
Result:
   Collection Service v.0.2
-----------------------------------------------------
Parameters:
   CO176
   CO002
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(getServiceDescription): got 2,expected 1
-----------------------------------------------------
Parameters:
   12
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(getServiceDescription(12)): got java.lang.Integer,expected String
=====================================================

Method: getErrorLog
-----------------------------------------------------
Parameters:
   aaa
Result:
   FaultCode   17101
   FaultString org.apache.xmlrpc.XmlRpcException: Bad service id (getErrorLog): aaa
-----------------------------------------------------
Parameters:
   1
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(getErrorLog(1)): got java.lang.Integer,expected String
-----------------------------------------------------
Parameters:
   CO176
Result:
   void
=====================================================

Method: reportError
-----------------------------------------------------
Parameters:
   CO176
   Error 1
Result:
   void
-----------------------------------------------------
Parameters:
   CO176
   Error 2
```

```
Result:
   void
-----------------------------------------------------
Parameters:
   CO176
   1
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(reportError(CO176,1)): got java.lang.Integer,expected String
-----------------------------------------------------
Parameters:
   1
   Error
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(reportError(1,Error)): got java.lang.Integer,expected String
-----------------------------------------------------
Parameters:
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(reportError): got 0,expected 2
-----------------------------------------------------
Parameters:
   a
   b
   c
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(reportError): got 3,expected 2
=====================================================

Method: getErrorLog
-----------------------------------------------------
Parameters:
   CO176
Result:
   Error 1
Error 2
-----------------------------------------------------
Parameters:
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(getErrorLog): got 0,expected 1
=====================================================

Method: resetErrorLog
-----------------------------------------------------
Parameters:
   1
```

```
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(resetErrorLog(1)): got java.lang.Integer,expected String
-----------------------------------------------------
Parameters:
   aaa
Result:
   FaultCode   17102
   FaultString org.apache.xmlrpc.XmlRpcException: Bad service password
(resetErrorLog): aaa
-----------------------------------------------------
Parameters:
   CO2dhk7ub0
Result:
   void

check error log for
getErrorLog('CO176')  --->  void
=====================================================

Method: getUserIds
-----------------------------------------------------
Parameters:
   a
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(getUserIds): got 1,expected 0
-----------------------------------------------------
Parameters:
Result:
   [CW111_5520, CW111_5541, CW111_5021, CW111_397, CW111_4812, CW111_457, CW111_4849,
CW111_4879, CW111_4767, CW111_5303, CW111_5479, CW111_5562, CW111_5583, CW111_5604,
CW111_5625, CW111_5646, CW111_5667, CW111_5688]
=====================================================

Method: getUserInfo
-----------------------------------------------------
Parameters:
   CW111_5520
   a
   CW111_5541
Result:
   FaultCode   17109
   FaultString org.apache.xmlrpc.XmlRpcException: Bad user id (getUserInfo): a
-----------------------------------------------------
Parameters:
   CW111_5520
   CW111_5541
Result:
   [[CW111_5520, wirsam2, wido.wirsam@fit.fhg.de, CW111_5523],
[CW111_5541, wirsam3, wirsam@web.de, CW111_5544]]
-----------------------------------------------------
```

```
Parameters:
   1
Result:
   FaultCode   17109
   FaultString org.apache.xmlrpc.XmlRpcException: Bad user id (getUserInfo): 1
=====================================================

Method: inviteUser
-----------------------------------------------------
Parameters:
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(inviteUser): got 0,expected 1
-----------------------------------------------------
Parameters:
   candela@iei.pi.cnr.it
Result:
   void

The email was generated!
-----------------------------------------------------
Parameters:
   ccc
Result:
   void
=====================================================

Method: addService
-----------------------------------------------------
Parameters:
   3
   http://address.com
   CO
   bad service version type
   http://address.com
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(addService(3,http://address.com,CO,bad service version type,http://address.com)):
got java.lang.Integer,expected String
-----------------------------------------------------
Parameters:
   '0.3'
   http://address.com
   CO
   service desc
   http://address.com
Result:
   [CO678, CO93cbfnuc]
=====================================================

Method: updateService
-----------------------------------------------------
```

```
Parameters:
   aaa
   '0.3'
   http://address.com
   CO
   service desc
   http://address.com
Result:
   FaultCode   17001
   FaultString org.apache.xmlrpc.XmlRpcException: Bad number of parameters
(updateService): got 6,expected 5
-----------------------------------------------------
Parameters:
   'aaa'
   0.3
   http://address.com
   service desc
   http://address.com
Result:
   FaultCode   17102
   FaultString org.apache.xmlrpc.XmlRpcException: Bad service password
(updateService): aaa
-----------------------------------------------------
Parameters:
   1
   0.3
   http://address.com
   service desc
   http://address.com
Result:
   FaultCode   17002
   FaultString org.apache.xmlrpc.XmlRpcException: Bad parameter type
(updateService(1,0.3,http://address.com,service desc,http://address.com)):
got java.lang.Integer,expected String
-----------------------------------------------------
Parameters:
   CO93cbfnuc
   0.4
   http://addressUpdated.com
   service desc updated
   http://addressUpdated.com
Result:
   void

check service data
getService('CO')--->[[CO002, 1.0, http://www.address.com, 0.79999995],
[CO678, 0.4, http://addressUpdated.com, 1.0],
[CO176, 0.2, http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server, 1.0]]
=====================================================

Method: deleteService
-----------------------------------------------------
Parameters:
   CO678
```

```
Result:
   FaultCode   17102
   FaultString org.apache.xmlrpc.XmlRpcException: Bad service password (deleteService): CO678
-----------------------------------------------------
Parameters:
   CO93cbfnuc
Result:
   void

check service
getService('CO')--->[[CO002, 1.0, http://www.address.com, 0.79999995],
[CO176, 0.2, http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server, 1.0]]
-----------------------------------------------------

=====================================================
Add, update and remove a service
=====================================================
Method: addService
-----------------------------------------------------
Parameters:
   0.3
   http://address.com
   CO
   service desc
   http://address.com
Result:
   [CO172, CO1iug8c9g]
-----------------------------------------------------
Method: getService
Parameters:
   CO
Result:
   [[CO002, 1.0, http://www.address.com, 0.79999995],
[CO176, 0.2, http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server, 1.0],
[CO172, 0.3, http://address.com, 1.0]]
-----------------------------------------------------
Method: getServiceDescription
Parameters:
   CO172
Result:
   service desc
-----------------------------------------------------
Method: updateService
Parameters:
   CO1iug8c9g
   0.4
   http://addressUpdated.com
   service desc updated
   http://addressUpdated.com
Result:
   void
-----------------------------------------------------
Method: getService
Parameters:
   CO
Result:
   [[CO002, 1.0, http://www.address.com, 0.79999995],
[CO176, 0.2, http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server, 1.0],
[CO172, 0.4, http://addressUpdated.com, 1.0]]
-----------------------------------------------------
Method: getServiceDescription
Parameters:
   CO172
Result:
   service desc updated
-----------------------------------------------------
Method: deleteService
Parameters:
   CO1iug8c9g
Result:
   void
-----------------------------------------------------
Method: getService
Parameters:
   CO
Result:
   [[CO002, 1.0, http://www.address.com, 0.79999995],
[CO176, 0.2, http://project.iei.pi.cnr.it:8080/CollectionService/CS_Server, 1.0]]
=====================================================
```

MS API test (FIT)

```
MS API Test (FIT)
Tester: Wido Wirsam (wido.wirsam@fit.fraunhofer.de)
Date: 25.2.03
Time: 11.13


Method tested: addUser
Argument(s)  : "CW111_7225", "wirxam", "cyclades", "wido.wirxam@fit.fraunhofer.de", "CW111_7228"
Return Value : none

Method tested: inviteUser
Argument(s)  : "wirsam@web.de", "Join the Team"
Return Value : none
Side Effect: the following Email was sent to 'wirsam@web.de':

"
Mail generated by CYCLADES System.

You have been invited to the CYCLADES System with the following message:

----------------------------------------------------------------------------------------
Join the Team
----------------------------------------------------------------------------------------
```

```
In order for this invitation to take effect you must visit the CYCLADES registration page.
You can do so by following the link:

http://calliope.ics.forth.gr:7007/Cyclades/RegistrationForm.html

Invitation will only take effect if the e-mail address under which you received
that mail is used.
"


Method tested: getService
Argument(s)  : "CW"
Return Value : [['CW665', '0.2', 'http://cyclades.gmd.de/cgi-bin/cyc_cws.cgi',
                  0.79999995000000002]]

Method tested: getServiceDescription
Argument(s)  : "CW665"
Return Value : "The Collaborative Work Service manages and organizes metadata
                records in folders and lets users share their folders"

Method tested: getErrorLog
Argument(s)  : "CW665"
Return Value : 13/1/2003 14:58:48 test-error1
               13/1/2003 14:59:28 test-error2

Method tested: reportError
Argument(s)  : "CW665", "25.02.2003 11.26.00 CWS test error report"
Return Value :
Side Effect: Error message has been added to the error log.

Method tested: addService
Argument(s)  : "1.0", "http://addressOfAPI", "CW", "Service created for testing",
               "http://addressOfGUI"
Return Value : ['CW455', 'CW5m985zbk']

Method tested: deleteService
Argument(s)  : "CW5m985zbk"
Return Value :

Method tested: updateService
Argument(s)  : "CWaflve4bx", "1.1", "http://addressOfAPI",
               "Service created for testing", "http://addressOfGUI"
Return Value :

Method tested: resetErrorLog
Argument(s)  : "CWaflve4bx"
Return Value :

Method tested: getUserIds
Argument(s)  :
Return Value :
['CW665_7225', 'CW665_13620', 'CW665_6971', 'CW665_7131', 'CW665_7471', 'CW665_7512',
'CW665_7558', 'CW665_7594', 'CW665_7853', 'CW665_8042', 'CW665_30', 'CW665_8102',
'CW665_10244', 'CW665_10770', 'CW665_11159', 'CW665_11497', 'CW665_11577',
'CW665_11610', 'CW665_11643', 'CW665_12520', 'CW665_12550', 'CW665_12580',
'CW665_12610', 'CW665_7441', '-', 'CW665_15325', 'CW665_18867', 'CW665_19135',
'CW665_19938', 'CW665_19564', 'CW111_7225']

Method tested: getUserInfo
Argument(s)  : ["CW665_7225"]
Return Value : [['CW665_7225', 'wirsam', 'wirsam@fit.fraunhofer.de', 'CW665_7228']]
```

7.3 MS GUI test

7.3.1 MS GUI description

The MS implements the Graphical User Interface of CYCLADES for users' registration and login. Through this the user is able to register and login into CYCLADES. Also she is able to access the CYCLADES services (and their GUIs, when provided) and thus navigate through them.

When a user is visitting the CYCLADES page she might choose whether she wants to login (in case she has already registered) or she can go to the registration page for registrtation.

After succesfull login user is presented with an interface where both her home folder (CWS GUI) and buttons for accessing CYCLADES services are shown. A user can either navigate through her folders or choose to activate a service. If she chooses to activate another service, then she is presented with that service's user interface.

Among the buttons that the user is presented with, at the MS GUI, there is a button for administration purposes. If the user chooses to press that button then a form asking for administration password will be shown. There the user in order to continue must fill in the administrator's password or, in the case that she has been granted with the administration privilege, she must fill in her name and her pasword. In the MS administration area someone can see all the registered users in CYCLADES and can apply various actions on them.

The actions an administrator can apply onto the users (for the moment) are:

- remove a user
- delete a user
- check whether a user has administrator's privileges or not
- set the administrator privilege to a user
- un-set the administrator privilege to a user

(Of course the set of actions can be extended.)

7.3.2 MS GUI test plan

The MS GUI test plan is based on the list of use cases that have been identified for the MS in the Deliverable D3.0.1. These use cases are:

- User registration
- User login

- User navigation through services

Also the MS GUI test plan presents the administration section at MS level where administrator can apply a set of actions on users. For the moment this set includes the followin actions:

- remove a user
- delete a user
- get the status of a user (whether she is an administrator or not)
- set a user to be an administrator
- 'un-set' a user of being an administrator

Also the MS GUI test plan presents the options that a user has, which for the moment are:

- change password
- un-register from Cyclades

7.3.3 MS GUI test results

- **Tester:** Papadopoulos Nikos (npap@ics.forth.gr)
- **Test date:** 22 November, 2002
- **Scope of test:** Complete GUI of MS was tested according to the above test plan.
- **Test environment:** Test client was an Internet Explorer running on Windows 2000.

```
Case: User registration
username: test
password: test
e-mail address: npap@ics.forth.gr
Result: OK
Checked: registration succeeded and after confirmation user was presented with the main
user interface where both her home folder and the buttons for activating the various
services are shown.

Case: User login
username: test
password: test
Result: OK
Checked: login succeeded and user was presented with the main user interface where both
her home folder and the buttons for activating the various services are shown.

Case: User navigation through services
Result: OK
Checked: User chosed to activate Collection Service and she was (successfully) presented
with the GUI of CS.

Case: administration actions
Result: OK
Checked: after entering a valid administration username and password, user was presented
with all the registered users and the administration actions.
```

Case: administration actions: remove a user
Result: OK
Checked: User successfully removed from the CYCLADES. Removed from each service and from the MS.

Case: administration actions: delete a user
Result: OK
Checked: User successfully deleted only from the MS Database only and not from the CYCLADES.

Case: administration actions: set a user as administration - get status - 'un-set' that user of being an administrator
Result: OK
Checked: Administrator user sets a user as administration and confirms that by applying 'get status' action on that user. Now that user is shown as an administrator. Then, the administrator user applys 'un-set' action on the same user and removes administration right. Now that user (with 'get status' action) is not shown as an administrator.

Case: User options: change password
Result: OK
Checked: User after selecting to change her password, she re-logined into CYCLADES and now the new password required.

Case: User options: un-register CYCLADES
Result: OK
Checked: User was successfully removed from the CYCLADES, meaning that she was removed from all the services and the MS databse too. User had to re-register in order to be able to use CYCLADES and its services.

Chapter 8

RMS Component Test

8.1 Introduction

The Rating Management Service (RMS) stores record ratings that take place within the CWS and makes them available to other services. The RMS stores the ratings in form of a table where every row corresponds to a rating with entries of the rated record, the user that rated, the folder in whose context the rating was made and finally date and time of rating and the rating value itself.

The RMS has an API with 5 methods, but no graphical user interface.

8.2 RMS API test

8.2.1 RMS API specification

The RMS provides an API to the other CYCLADES services. The methods of this API may be called using the inter-service communication protocol XML-RPC. For every method also the services are listed that call this method according to the service interaction as specified in Deliverable D3.0.1.

- **Method:** saveRating
  Signature: void saveRating(recordId, folderId, userId, ratingValue)
  Description: this method is invoked in order to store a rating in the RMS rating table. The rating is specified by the identifier of the record that has been rated, the identifier of the folder that contained the record when it was rated, the identifier of the user that rated, and the rating value that was assigned by the user to the record. The rating timestamp is generated within the RMS.
  Parameters:

Input:	recordId:	a record identifier.
	folderId:	a folder identifier.
	userId:	a user identifier.
	ratingValue:	an integer representing the rating value.

  Calling service: CWS

- **Method:**
  Signature: (recordId, userId, value)* getFolderRatings(folderId, timestamp)
  Description: this method is invoked in order to get all ratings that the records contained in a given folder received since a given point in time.
  Parameters:

Input:	folderId:	a folder identifier.
	timestamp:	a point in time coded as an ISO 8601 date/time in UTC.
Output:		a list of triples containing each:
	recordId:	a record identifier,
	userId:	a user identifier,
	value:	an integer representing a rating value.

Calling service: FRS

- **Method:**
  Signature: (folderId, userId, value)* getRecordRatings(recordId, timestamp)
  Description: this method is invoked in order to get all ratings that a given record has received since a given point in time.
  Parameters:

Input:	recordId:	a record identifier,
	timestamp:	a point in time coded as an ISO 8601 date/time in UTC.
Output:		a list of triples containing each:
	folderId:	a folder identifier.
	userId:	a user identifier,
	value:	an integer representing a rating value.

  Calling service: FRS

- **Method:**
  Signature: (recordId, folderId, value)* getUserRatings(userId, timestamp)
  Description: this method is invoked in order to get all ratings that a given user has made since a given point in time.
  Parameters:

Input:	userId:	a user identifier,
	timestamp:	a point in time coded as an ISO 8601 date/time in UTC.
Output:		a list of triples containing each:
	recordId:	a record identifier,
	folderId:	a folder identifier.
	value:	an integer representing a rating value.

  Calling service: FRS

- **Method:**
  Signature: void anonymizeUserRatings(userId)
  Description: this method is invoked in order to anonymize all ratings of a given user; the user identifier in the ratings table is replaced by the string 'Anonymous'.
  Parameters:
  Input: userId: a user identifier,
  Calling service: CWS

8.2.2 RMS API test

In the following we list the API methods by calling service indicating test responsibilities for the methods listed.

- **CWS** (FIT)
  - saveRating
  - anonymizeUserRatings

- **FRS** (CNR)
  - getFolderRatings
  - getRecordRatings
  - getUserRatings

In the following we summarize the results of the RMS API tests as conducted by FIT and the partners responsible for services calling the RMS API.

RMS API test (FIT)

- **Tester:** Wido Wirsam (wido.wirsam@fit.fraunhofer.de)
- **Test date:** 6 November, 2002 10:43 GMT
- **Scope of test:** Complete API of RMS v0.2[1] was tested via several Python scripts which tested every method of the API with a number of parameter sets that were read from a test data file.
- **Test environment:** Test client was a Python 2.2 script running on Windows XP making use of xmlrpclib module v. 0.9.9, the XML-RPC implementation of Secret Labs AB[2].
- **Test plan:** The RMS XML-RPC test platform provides a tool to automatically create CWS-users. All API methods that are provided by the RMS-service are performed on these users and their folders.

  The platform consists of the following python programs:

  - definitions.py This configuration file defines a set of values necessary for the creation of the test-users and the behaviour of the testrun.
  - TestFieldGenerator.py This script performs the generation of test-users. All relevant information about the generated users is stored in a log file.
  - logfile.py This class provides access to all information about the users created by the TestFieldGenerator.
  - main.py This script executes all XML-RPC calls provided by the RMS-services on the users generated by the TestFieldGenerator.

  To test the RMS API the same dummy users are used, that have been created to test the CWS API methods. Then all RMS API methods have been executed. Most of the methods require a user-Id or a folder-Id as parameters. All methods of the RMS API that require a user-Id and no folder-Id as parameter are called once for every user, with that specific user-Id as parameter. All methods that require a folder-Id as parameter are called once per folder of every user. Each call of every method and its results are logged the 'Access.txt' file.

  The RMS API test was performed on the server 'http://cosidetti.gmd.de' which runs the same version of CYCLADES CWS as the 'http://cyclades.gmd.de' server. The 'http://cosidetti.gmd.de' server is not connected to the other services so the CWS API functionality can be tested with no side effects on the other services. The test was performed with 20 computer generated users on November 6th 2002. Every user was generated by the CWS API method:

  ```
  createUser(name, password, emailAddress)
  ```

  The username and password are automatically generated. For the emailAddress argument a valid eMail address is used. This is the resulting logfile 'TFGLog83.txt' :

[1] http://cosidetti.gmd.de/rpc2/cyc_cws.cgi
[2] http://www.pythonware.com/products/xmlrpc/

ID	userName	userPwd	userMail	userID	userFolder
0	Karlitpqr	zxtbqo	wirsam@web.de	CW111_11179	CW111_11182
1	Karlmdx	kjooqh	wirsam@web.de	CW111_11222	CW111_11225
2	Karlthbe	qvuxyun	wirsam@web.de	CW111_11265	CW111_11268
3	Karllvpzij	cbahu	wirsam@web.de	CW111_11308	CW111_11311
4	Karlasg	ozjhlt	wirsam@web.de	CW111_11351	CW111_11354
5	Karlhkrvqf	bttfgc	wirsam@web.de	CW111_11394	CW111_11397
6	Karlcjfb	vkgvvjn	wirsam@web.de	CW111_11437	CW111_11440
7	Karluaje	ttkgna	wirsam@web.de	CW111_11480	CW111_11483
8	Karlselx	bbfsovw	wirsam@web.de	CW111_11523	CW111_11526
9	Karlngj	dfxdr	wirsam@web.de	CW111_11566	CW111_11569
10	Karlcrmi	gskib	wirsam@web.de	CW111_11609	CW111_11612
11	Karlmxdtii	eouvhmn	wirsam@web.de	CW111_11652	CW111_11655
12	Karlbavk	mccmoqa	wirsam@web.de	CW111_11695	CW111_11698
13	Karlxno	nwjlvbw	wirsam@web.de	CW111_11738	CW111_11741
14	Karlcldq	fkiri	wirsam@web.de	CW111_11781	CW111_11784
15	Karlfqk	izlkeo	wirsam@web.de	CW111_11824	CW111_11827
16	Karlbkw	acabwh	wirsam@web.de	CW111_11867	CW111_11870
17	Karlwxrf	yqxjn	wirsam@web.de	CW111_11910	CW111_11913
18	Karlwueige	hyyaodv	wirsam@web.de	CW111_11953	CW111_11956
19	Karloed	vrsqffj	wirsam@web.de	CW111_11996	CW111_11999

The file 'TFGLog83.txt' is available online[3]. For each user four folders were produced: one private folder, one project folder, one community root folder and one community folder. This was done by calling the CWS API method:

```
createFolder(userID, folderID, folderType)
```

In the test run all RMS API methods were called. The calls and the results were stored in a logfile called 'Access.txt'. In the logfile 'Errors.txt' error messages would have been logged if any had occured. Most methods take either a userID or a folderID as argument. Every API method that needs a userID is called once for every user. The method 'getFolders(userID)' returns a list of folders that are owned by the user. In our testrun the method returned the four folders that previously were created. The results of these calls were stored during the testrun. Thereafter every method of the RMS API that needs a folderID as argument was called with each of the returned folderIDs for every user. In detail these were the following methods:

called once at all:

```
getRecordRatings()
```

called once per user:

```
getUserRatings()
```

called once per folder:

```
saveRating()
getFolderRatings()
```

[3]http://cyclades.gmd.de/publicLogs/RMS/TFGLog83.txt

All together 181 API-Calls have been made.

This is an excerpt of the 'Access.txt' logfile. The complete file is available online[4].

```
AccessLogfile created Wed Nov 06 10:43:03 2002

result = s.service.getUserRatings('CW111_11179', xmlrpclib.DateTime(time.time()
-30000000))
[['AC_00001', 'CW111_11200', 1], ['AC_00001', 'CW111_11204', 1], ['AC_00001',
'CW111_11211', 1], ['AC_00001', 'CW111_11218', 1]]

result = s.service.saveRating(testRecords[0], 'CW111_11200', 'CW111_11179', 1)


result = s.service.getFolderRatings('CW111_11200', xmlrpclib.DateTime(time.time
()-30000000))
[['AC_00001', 'CW111_11179', 1]]

result = s.service.saveRating(testRecords[0], 'CW111_11204', 'CW111_11179', 1)


result = s.service.getFolderRatings('CW111_11204', xmlrpclib.DateTime(time.time
()-30000000))
[['AC_00001', 'CW111_11179', 1]]

result = s.service.saveRating(testRecords[0], 'CW111_11211', 'CW111_11179', 1)


result = s.service.getFolderRatings('CW111_11211', xmlrpclib.DateTime(time.time
()-30000000))
[['AC_00001', 'CW111_11179', 1]]
```

During the complete testrun no errors have occured. This is the contens of the error logfile 'Errors.txt':

```
ErrorLogfile created Wed Nov 06 10:43:03 2002

ErrorLogfile closed Wed Nov 06 10:45:27 2002
```

- **Test summary:** All methods worked according to specification for all sets of input parameters. 20 computer generated users were created via the CWS API method 'createUser()'. On these users all available RMS API methods were executed. No Errors occured during the testrun. The method calls and the results produced by the RMS system were logged and made available online.

RMS API test (CNR)

- **Tester:** Henri Avancini (avancini@iei.pi.cnr.it)
- **Test date:** 24 February 2003
- **Scope of test:** Testing RMS API[5] methods called from FRS.

[4]http://cyclades.gmd.de/publicLogs/RMS/Access.txt

[5]`http://cyclades.gmd.de/cgi-bin/cyc_cws.cyc`

- **Test environment:** Test client was a Java[6] application running on Linux (2.4.18-19.8.0), the XML-RPC implementation of Apache XML Project[7].
- **Test plan:** Test is divided in two stages. First, all user and folder identifiers are gathering from MS and CWS respectively (using *MS getUserIds* method and *CWS getFolders* method). Second, a user defined number of iterations are executed testing RMS methods.

  Each iteration consist of:

  - Select both a random[8] valid user and folder identifier.
  - Call used RMS method: *getFolderRating*. Write down returned values. "Timestamp" is setted to an arbitrary initial value.
- **Test log:** An excerpt of test log is show bellow (file: frsCalledAPITest_detailed.log). The complete test log is available on-line[9].

```
Method called: frsCalledAPITest
Mon Feb 24 12:21:05 CET 2003
Initialize {AS, CWS, MS, RMS}Clients.

Method called: msclient.getUserIds ()
[CW665_7225, CW665_13620, CW665_6971, CW665_7131, ...]

Method called: cwsclient.getFolders (userID)
Retrieving user folders. User: CW665_7225
Folders: [CW665_7103, CW665_7111, CW665_7495, CW665_7499, ...]

Loop: 8
Selected user: CW665_7131
Selected folder: CW665_19406

rmsclient.getFolderRatings:
[[AC832_oai_dc_oai:UMIMAGES:1000505, CW665_7131, 2]]

{AS, CWS, MS, RMS} API calls sumatory: 101
```

  'FRSServer.log' contains FRS server side log, which correspond to the test period. The file 'frsCalledAPITest.log' contains console outputs from the client side and error messages. Lastly, 'startfrs.sh.log' file contains console outputs from the server side and error messages.
- **Test summary:** All methods called from FRS were tested. All methods tested worked according to specification for all set of correct input parameters. A total of 120 API calls were executed[10]. No errors occurred during the test run.

[6]Java version 1.4.0_02, Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)
[7]http://xml.apache.org/xmlrpc
[8]http://java.sun.com/j2se/1.4/docs/api/java/util/Random.html
[9]http://project.iei.pi.cnr.it:8080/FRS/publicLogs/
[10]Total number of calls from FRS. As AS, CWS, MS, RMS test were made together, because of his dependencies, this number correspond to the sumatory of calls made from FRS to other CYCLADES services.

Chapter 9

Integrated System Test

9.1 Introduction

Integrated system testing is to ensure that the single CYCLADES components communicate properly to provide the intended system functionality. The components, i. e. the single CYCLADES services, communicate using the inter-service communication protocol XML-RPC making use of the diverse service APIs. The components reside on the developing partner's hosts during the test.

Integrated system testing uses as a general test plan the use cases as described in Deliverable D3.0.1 focussing specifically on use cases that involve inter-service communication (the other use cases have been tested during component testing). When integrated system functionality is to be tested that involved more than one user, the existing use cases are extended, or two or more use cases are put together to form a multi-user test case. The interaction diagrams depicted in the "Service interaction diagrams" sections of Deliverable D3.0.1 served as a check-list for the integrated systems testing because they typically represent system functionality that involves inter-service communication.

9.2 Integrated system test plan

The test plan for the integrated system test of the CYCLADES system comprises all use cases identified in D3.0.1 that involve more than one service. The test cases are executed from the initiating service. This will normally be done by a user from the service GUI; with some use cases, however, the execution is triggered by a service internal schedule, e. g. folder profile update at scheduled time. The effects of the inter-service communication are checked with the services involved to confirm a successful test, e. g. registering with the Mediator Service should also create a user and home folder object in the Collaborative Work Service.

Some of the service GUIs are started by pushing buttons in other service GUIs. This inter-service communication does not involve the service APIs and is tested separately, i. e. test cases for the Collection Service start from the CS GUI and not from the Mediator Service GUI that is used to start up the CS GUI window. Following is the list of services that start up other service GUIs:

- MS: Start up Mediator Service and Collaborative Work Service after login
- MS: Start up the Archive Service GUI
- MS: Start up the Collection Service GUI
- CWS: Start up the Search and Browse Service GUI

The rest of the test plan is organized by the services from whose GUI the test case starts out. The name of the test case is normally the name of the use case that is the initial step of the test case. The steps to be executed by other services in the course of the test case are listed.

- AS: register archive
  should also create an archive collection in the CS by adding and initializing the archive collection in the CS which in turn asks the AS for the archive description and samples the new archive
- AS: delete archive
  should also notify the CS of the deletion
- CWS: create folder
  should notify FRS along with recommendation preferences
- CWS: destroy folder
  should notify FRS
- CWS: add record
  should notify FRS
- CWS: delete/cut record
  should notify FRS
- CWS: edit folder preferences
  should notify FRS of changed recommendation preferences
- CWS: rate records
  should notify FRS and RMS
- CWS: invite members by e-mail addresses
  should ask MS to actually invite the new members to register; the invitees should receive an e-mail asking them to register; after registration the invitees should be members of the folder to which they were invited
- CWS: update folder profile (on demand)
  should ask FRS to immediately update the folder profile; the FRS in turn should receive records put into the folder since the last recommendation of records from the CWS; after computation of new record recommendations the CWS should receive them if there are any
- CWS: update collections (scheduled and on demand)
  should receive the up-to-date list of collections from CS
- CWS: update services (scheduled and on demand)
  should receive the up-to-date list of services from MS
- SBS: browse system collections
  should receive all collections in the system from CS and receive these collections' metadata from the CS
- SBS: browse folder collections
  should receive the associated collections of current folder from CWS and receive these collections' metadata from the CS
- SBS: browse personal collections
  should receive the personal collection for the user set from CS and receive these collections' metadata from the CS
- SBS: browse attribute values
  should receive the attribute values from AS

- SBS: request new records for folder
  should receive the associated collections of current folder from CWS and receive for each collection the 'new' records from FRS which in turn should retrieve these records from the AS based on collection and folder profile
- SBS: save results
  should store the selected results in the CWS
- SBS: save query
  should store the current query in the CWS
- SBS: submit query without personalization
  should receive the matching records from the AS
- SBS: submit query with personalization
  should receive the matching records from the FRS which in turn should receive these records from the AS based on the query and the folder profile
- FRS: update folder profiles (scheduled)
  should receive all folder ids of all users from the CWS; should then for each folder receive the records put into the folder since the last profile update from the CWS, receive the indexed terms and weights of these records from the AS, receive the ratings of records in the folder from the RMS if there are any, and then compute the new folder profile
- FRS: recommend records for folder (scheduled)
  should receive records that were put into similar folders since last recommendation from CWS, compute recommendations and recommend the top scorers to the CWS folder
- FRS: recommend collections for folder (scheduled)
  should receive collections associated to similar folders from CWS, compute recommendations and recommend the top scorers to the CWS folder
- FRS: recommend users for folder (scheduled)
  should receive members of similar folders from CWS, compute recommendations and recommend the top scorers to the CWS folder
- FRS: recommend communities for folder (scheduled)
  should receive the communities that similar folders belong to from CWS, compute recommendations and recommend the top scorers to the CWS folder
- CS: create collection
  should notify CWS of the new collection
- CS: delete collection
  should notify CWS of the deletion
- CS: select personal collection set
  should notify the CWS of the new personal collection set
- MS: register
  should have the CWS create a user object with password and mail address given and receive user identifier and home folder identifier back
- MS: change password
  should change the password also in the CWS
- MS: delete user
  should delete the user object in the CWS which in turn anonymizes all ratings of this user in the RMS
- MS: become collection administrator

9.3 Integrated system test results

The system version tested was 1.0 The test reports on the diverse portions of the test plan are included below.

MS test cases (FIT)

- **Tester:** Thomas Kreifelts (kreifelts@fit.fraunhofer.de)
- **Test date:** February 25 14:19:23 MET 2003
- **Scope of test:** All test cases of Mediator Service were tested.
- **Test environment:** Test client was a Netscape 4.7 browser on a SUN workstation running Solaris.
- **Test plan:** The MS test cases as listed above were tested for a new user.
- **Test log:** The CWS access log file for the test cases is appended below.

```
...
```

- **Test summary:** All test cases worked according to specification. However, a button or link to start the interaction to become a collection administrator was not found in the MS GUI.

AS test cases (UniDuE)

- **Tester:** Gudrun Fischer (Gudrun.Fischer@uni-duisburg.de)
- **Test date:** March 25 16:40:02 MET 2003
- **Scope of test:** All test cases of the Access Service were tested.
- **Test environment:** Test client was Mozilla 1.2b under Debian Linux.
- **Test plan:** The AS test cases as listed above were tested for the user `CW665_7853 - gf`.
- **Test log:** The AS access log file for the test cases is appended below.

```
Case Register archive
 Methods invoked:
  - addCollection
  - initializeCollection
         Id=CO563_Coll70457
         Name=mundus.ac.uk
         Description=Cyclades archive mundus.ac.uk
         userId=CW665_7853
  - getUserInfo
 Result:
  - OK
 Checked:
  - user information was returned by the CWS, the user gf received and
 e-mail message
  - new collection was sampled by the Collection Service
  - new collection was available for search

Case Delete archive
 Methods invoked:
  - deleteArchive - Id=TalkBank
 Result:
  - OK
Checked:
  - the collection was not available any more afterwards
```

- **Test summary:** Both test case worked according to specification.

SBS test cases (UniDuE)

- **Tester:** Gudrun Fischer (Gudrun.Fischer@uni-duisburg.de)
- **Test date:** March 25 16:57:00 MET 2003
- **Scope of test:** All test cases of the Search and Browse Service were tested.
- **Test environment:** Test client was Mozilla 1.2b under Debian Linux.
- **Test plan:** The SBS test cases as listed above were tested for the user `CW665_7853 - gf` and folder `CW665_7880`.
- **Test log:** The SBS access log file for the test cases is appended below.

```
Case Browse system collections
 Methods invoked:
  - listCollections
  - getCollectionMetadata
 Result:
  - OK
 Checked:
  - all system collection IDs were received from the CS
  - for all IDs specified, the metadata was returned by the CS

Case Browse folder collections
 Methods invoked:
  - getCollections (CWS)
  - getCollectionMetadata (CS)
 Result:
  - OK
 Checked:
  - all folder collection IDs were received from the CWS
  - for all IDs specified, the metadata was returned by the CS

Case Browse personal collections
 Methods invoked:
  - getPersonalCollections
  - getCollectionMetadata
 Result:
  - OK
 Checked:
  - all personal collection IDs for the user were received from the CS
  - for all IDs specified, the metadata was returned by the CS

Case Browse attribute values
(Tested by calling the getAttributeValues API method at the SB, as
this functionality is still missing in the SB GUI.)
 Methods invoked:
  - getAttributeValues (AS)
 Result:
  - OK
 Checked:
  - the values for the given attributes and schemas were returned by
the AS

Case Get new records for folder
 Methods invoked:
  - getCollections (CWS)
  - getNewRecords (FRS)
 Result:
  - OK
 Checked:
  - the ids of the collections associated to the given folder were
 returned by the CWS
  - the query was generated correctly by the SBS and passed to the FRS
  - the FRS returned a list of records as expected

Case Save results
 Methods invoked:
  - saveResults
 Result:
  - OK
 Checked:
  - after refreshing the folder frame, the saved records were shown in
 the user's folder

Case Save query
 Methods invoked:
  - saveQuery
 Result:
  - OK
 Checked:
  - after refreshing the folder frame, the query was shown in
 the user's folder

Case Submit query without personalization
 Methods invoked:
  - search (AS)
 Result:
  - OK
 Checked:
  - a correct query was generated by the SBS and submitted to the AS
  - the AS returned records from the specified collections, which were
 matching the query conditions
```

```
Case Submit query with personalization
 Methods invoked:
  - filteredSearch
 Result:
  - OK
 Checked:
  - a correct query was generated by the SBS and submitted to the FRS
  - the FRS submitted the query to the AS
  - the AS returned records from the specified collections, which were
 matching the query conditions
  - the FRS returned the list of records from the AS, filtered
 according to the folder profile
```

- **Test summary:** All test cases worked according to the specifications.

CS test cases (CNR)

- **Tester:** Leonardo Candela (candela@iei.pi.cnr.it)
- **Test date:** February 27 17:04:02 MET 2003
- **Scope of test:** All test cases of Collection Service v0.2[1] were tested.
- **Test environment:** Test client was a Internet Explorer 6 browser running on Windows 2000.
- **Test plan:** The CS test cases as listed above were tested for the user `CW665_7131 - candela`.
- **Test log:** The CS access log file for the test cases is appended below.

```
Case Create collection
 Methods invocked:
  - addCollection
  - initializeCollection
          Id=CO563_Coll74282
          Name=IntegratedTestCollection
          Description=IntegratedTestCollection description
          userId=CW665_7131
          parent=CO_CollCYCLADES
 Result:
  - OK
 Checked:
  - new collection is shown in the candela's created collection
    tree hierarchy
  - new collection is notified to CWS (trying to create a new
    project folder I can see the list of all collection - if
    no personal collection are selected -)

Case Delete collection
 Methods invocked:
  - deleteCollection - Id=CO563_Coll74282 - userId=CW665_7131
 Result:
```

[1]http://project.iei.pi.cnr.it:8080/CollectionService/start?userid=CW665_7131

```
   - OK
  Checked:
   - collection CO563_Coll74282 was removed from candela's created
     collection tree hierarchy
   - deletion of collection CO563_Coll74282 is notified to CWS
     (trying to create a new project folder I can see the list of
     all collection - if no personal collection are selected -)

 Case Select personal collection set
  Methods invocked:
   - listCollections - userId=CW665_7131
   - listCollections
   - getCollectionMetadata - [CO563_Coll49044]
   - listCollections
   - getCollectionMetadata - [CO563_Coll23606]
   - listCollections
   - listCollections - userId=CW665_7131
  Result:
   - OK
  Checked:
   - candela's personal collection set contains collections
     CO563_Coll49044 and CO563_Coll23606
   - new candela's personal collection set is notified to CWS
     (trying to create a new project folder I can see now the
     list of my personal selected collection)
```

- **Test summary:** All test cases worked according to specification.