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

Global System Architecture Report
D2.2.1

Delivery Type: R
Number: D2.2.1
Contractual Date of Delivery: month 7
Actual Date of Delivery: July, 31st 2001
Task: WP2

Name of Responsibles: Gudrun Fischer, Norbert Fuhr

University of Dortmund
Computer Science VI
D-44221 Dortmund
Germany

E-Mail: {fischer,fuhr}@ls6.cs.uni-dortmund.de

Contributors:

CNR-IEI:	Donatella Castelli, Pasquale Pagano, Elena M. Renda, Fabrizio Sebastiani, Umberto Straccia
University of Dortmund:	Gudrun Fischer, Norbert Fuhr
FORTH:	Nick Papadopoulos, Dimitris Plexousakis
GMD-FIT:	Tom Gross, Thomas Kreifelts

Abstract: This report defines the core functionality of the CYCLADES system as well as an architecture suitable for supporting such a system functionality. The functional specification is based on the use case model.

Contents

1 Cyclades **6**

1.1 Purpose of the Cyclades system . . . 6
1.2 Terminology . . . 6
1.3 Users . . . 8
1.4 Use cases for the single user . . . 8
1.4.1 Joining the Cyclades environment . . . 9
1.4.2 Folder management . . . 9
1.4.3 Gathering records . . . 9
1.4.4 Record management . . . 9
1.4.5 Exploring and joining communities . . . 9
1.5 Use cases for a community . . . 9
1.5.1 Setting up a new community . . . 10
1.5.2 Joining and leaving a community . . . 10
1.5.3 Additional record handling in community folders . . . 10
1.5.4 Awareness . . . 10
1.5.5 Communicate with other community members . . . 10
1.6 Use cases for a project . . . 10
1.6.1 Setting up a new project environment . . . 10
1.6.2 Joining and leaving a project . . . 11
1.6.3 Additional folder content handling in project folders . . . 11
1.6.4 Awareness . . . 11
1.6.5 Communicate with other project members . . . 11
1.7 Use cases for collection management . . . 11
1.8 Use cases for archive management . . . 11
1.9 Use cases for user archive/collection registration rights . . . 12

2 Use cases in detail **13**

2.1 Joining the Cyclades environment . . . 13
2.1.1 Register as a new user (SU 1) . . . 13
2.1.2 Login (SU 2) . . . 14
2.1.3 View and edit user information (SU 3) . . . 15

2.2 Folder management . . . 15
2.2.1 Create folders and subfolders (SU 4) . . . 15
2.2.2 Manage folders (SU 5) . . . 16
2.2.3 Edit folder content description (SU 6) . . . 18
2.3 Gathering records . . . 18
2.3.1 Update folder profile (SU 7) . . . 18
2.3.2 Searching and browsing without personalization (SU 8) . . . 18
2.3.3 Personalized searching and browsing of collections (SU 9) . . . 19
2.3.4 Receive recommendations from the system (SU 10) . . . 20
2.4 Record management . . . 21
2.4.1 Move or copy records from one folder to another (SU 11) . . . 21
2.4.2 Delete records (SU 12) . . . 21
2.5 Exploring and joining communities . . . 21
2.5.1 View list of communities and subscribe to a community (SU 13) . . . 22
2.5.2 Join a community via invitation of an administrator (SU 14) . . . 22
2.6 Setting up a new community . . . 24
2.6.1 Create a new community (CO 1) . . . 24
2.6.2 Manage community membership and access rights (CO 2) . . . 24
2.7 Joining and leaving a community . . . 25
2.7.1 Register after invitation/subscribe (CO 3) . . . 26
2.7.2 Leave/Delete a community (CO 4) . . . 26
2.8 Additional record handling in community folders . . . 26
2.8.1 Rate a record (CO 5) . . . 26
2.8.2 Annotate a record (CO 6) . . . 27
2.9 Awareness . . . 28
2.9.1 Edit event notification preferences (CO 7) . . . 28
2.9.2 Catch up (CO 8) . . . 29
2.10 Communicate with other community members . . . 29
2.10.1 Create and add a note to a discussion forum (CO 9) . . . 29
2.10.2 Online chat (optional; CO 10) . . . 30
2.11 Collection management use cases . . . 30
2.11.1 Create collections (COL 1) . . . 30
2.11.2 Add search/browse format (COL 2) . . . 31
2.11.3 Delete collections . . . 32
2.11.4 Remove search/browse formats . . . 32
2.11.5 Edit collection descriptive metadata . . . 33
2.12 Archive management . . . 33
2.12.1 Register an archive . . . 33
2.12.2 Edit archive registration information . . . 35
2.12.3 Unregister an archive . . . 35

2.13 Management of archive/collection registration rights . . . 35
2.13.1 Becoming an archive registrator . . . 35
2.13.2 Becoming a collection registrator . . . 35
2.13.3 Managing registration rights . . . 35

3 System components, interaction, and class specifications 36
3.1 System components . . . 36
3.2 System activity and service interaction . . . 37
3.2.1 Registering as a new user (SU 1) - Mediator perspective . . . 37
3.2.2 Login . . . 37
3.2.3 Registering as a new user (SU 1) - CWS perspective . . . 38
3.2.4 Updating a folder profile on user demand (SU 7), CWS perspective . . . 39
3.2.5 Activating recommendations (SU 10-1) . . . 40
3.2.6 Deactivating recommendations (SU 10-6) . . . 40
3.2.7 Inviting new members (CO 2-1) and registering after invitation (CO 3-1) . . . 40
3.2.8 Rating records (CO 5) . . . 42
3.2.9 On-demand folder profile modification (SU 7-1) . . . 45
3.2.10 Scheduled folder profile modification (SU 7-2) . . . 45
3.2.11 Gathering records without personalization (SU 8) . . . 45
3.2.12 Personalized ad-hoc searching (SU 9-1), SBS perspective . . . 46
3.2.13 Personalized on-demand searching (SU 9-2), SBS perspective . . . 46
3.2.14 Personalized ad-hoc searching (SU 9-1), FRS perspective . . . 47
3.2.15 Personalized on-demand searching (SU 9-2), FRS perspective . . . 47
3.2.16 Receive record recommendation from the system (SU 10-1) . . . 47
3.2.17 Receive collection recommendation from the system (SU 10-2) . . . 48
3.2.18 Receive user recommendation from the system (SU 10-3) . . . 48
3.2.19 Receive community recommendation from the system (SU 10-4) . . . 48
3.2.20 Registering an archive . . . 63
3.2.21 Creating a collection . . . 63
3.2.22 Deleting a collection . . . 64
3.2.23 Adding a search and browse format . . . 64
3.2.24 Removing a search and browse format . . . 65
3.3 Editing collection metadata . . . 65
3.4 Class definitions in detail . . . 73
3.4.1 Access Service . . . 73
3.4.2 Collection Service . . . 74
3.4.3 Search and Browse Service . . . 76
3.4.4 Collaborative Work Service . . . 77
3.4.5 Rating Management Service . . . 80
3.4.6 Filtering and Recommendation Service . . . 82

3.4.7 Mediator Service 83
3.5 Query language 86

4 System architecture **87**
4.1 Services 87
4.2 Replication and distribution 88
4.3 User interfaces 88
4.4 Service autonomy 88
4.4.1 Access Service 88
4.4.2 Collaborative Work Service 88
4.4.3 Rating Management Service 89
4.4.4 Search and Browse Service 89
4.4.5 Collection Service 89
4.4.6 Filtering and Recommendation Service 89
4.4.7 Mediator Service 90
4.5 Customization and extensibility 90
4.6 Functionality and efficiency tests 90

5 Communication protocol **93**
5.1 Remote procedure calls via XML-RPC 93
5.1.1 Method call 93
5.1.2 Method response 94
5.2 Other communication protocols 95

Chapter 1

Cyclades

1.1 Purpose of the Cyclades system

Cyclades will provide an integrated environment for scholars and groups of scholars that use electronic archives. The main goal of Cyclades users is to retrieve those documents, respectively metadata about documents, from electronic archives that are relevant to their interests, and to be made aware of new relevant documents, resp. relevant metadata records, as they become available. Additional goals may be to get acquainted with other users with similar interests, to exchange information with such users in the form of record annotations or even by explicitly sharing records collected into folders. For collaboration, users can form projects where they can share not only metadata records, but also documents of all kinds. Ultimately, users contribute to building "community knowledge" for scientific communities of Cyclades users that these communities may profit from.

The Cyclades system will use the protocol specified by the Open Archives Initiative [1] (OAI) to harvest metadata from any number of archives that support the OAI standard. As the harvesting will be done by one of several interoperable services, Cyclades is not restricted to using open archives, as additional services can be added later, supporting other kinds of electronic archives.

The remainder of this document is structured as follows:

In this chapter (1), we define the terminology to be used in the specification, identify the different user roles, and we give an overview of the use cases. In chapter 2, we specify the use cases in more detail. Then, in chapter 3, we identify the main components of the system which will become autonomous services, describe the interaction between these services for the individual use cases, and define the major classes resulting from this analysis, together with their relevant fields and methods. In chapter 4, we describe the resulting architecture of the Cyclades system, also touching the internal query language and discussing the extent of autonomy and interoperability of the intended services. Finally, in chapter 5, we define the communication protocol that is going to be used in Cyclades.

1.2 Terminology

For the use in the specification, we first define some terms:

- *archive*
  An *archive* is a set of records describing documents by metadata. An archive is uniquely identified by a string, called the *archive identifier*.

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

- *document*
  An arbitrary external file that can be stored in a project folder.

- *archive document*
  A resource that the user is interested in discovering in an archive. A document may be text (papers, reports, journals) or any other kind of media. Each document is described by one or more metadata records that are contained in and retrieved from open archives. Note that an archive need not contain documents. Depending on the organisation hosting the archive, it might as well contain only metadata records.

- *(metadata) record*
  The entities contained in an archive are *records*. Each such record consists of a set of attributes and values describing a document, according to a specific metadata schema. One document can be described by several metadata records using different schemas.

- *metadata schema*
  A set of attribute definitions, including attribute names, the type of each attribute.

- *collection*
  A set of records defined by a set of archives and an optional *filtering query* (i.e. a selection criterion) which uses only Dublin Core attributes. All the records from the given archives which satisfy the query are members of the collection defined. A collection may provide different metadata schemas and different search services for each schema.
  From the user's point of view, a collection is a set of documents with specific formats of search and browse operations associated with them. In this perspective, a collection is described by a set of criteria that specify which are the documents that belong to the collection (*membership condition*) and by the format of the search and browse operations.

- *community*
  A *community* is a set of users sharing a common (scientific, professional) background or view of the world. Within Cyclades, communities are characterized by a shared interest in documents and records. This common interest manifests itself as a hierarchy of common folders where records of common interest are stored and commented.

- *project*
  This is a group of scholars working together closely, presumably in a common project or field of interest. They do not only want to share records of interest, but possibly also other kinds of documents and modify them in a common workspace (i.e. in a common folder system).

- *folder*
  This is a container for metadata records and, in the case of project folders, possibly also documents. Collections can be associated to folders. This means that the user can choose by default from these collections when she formulates a query. Queries can be stored in folders, allowing the user to resubmit or edit them at any later time. Thus, a folder can also be seen as an environment corresponding to a certain topic, where the user is interested only in data concerning this topic.

- *private folder*
  A folder owned by a single user. This kind of folders can only be accessed by their owner. For others, they are invisible.

- *community folder*
  A folder owned by a community. This is a folder that can be accessed and possibly manipulated by several users that thus form a community (see above).

- *project folder*
  A folder owned by a project, i.e. accessible to the members of a project. Only project folders can contain documents, private and community folders can contain only records and queries.

- *subfolder*
  A *subfolder* is a folder that is contained in another folder called the *parent folder*. Each folder can maximally have one parent folder. If a subfolder is copied to another folder, then the folder and its whole contents are copied (not only a reference), thus creating a distinct new folder.

- *home folder*
  Each user has a special folder called the *home folder* which constitutes the root of the folder hierarchy that is visible to the user.

- *user*
  A person registered in the Cyclades environment. A user has an identity, an e-mail address, a password and a home folder. Users can have further attributes like affiliation, postal address, telephone number.

- *rating*
  A relevance value assigned to a record by a user. The user may rate a record directly by assigning it a certain value, or indirectly by choosing the record from a query result list and storing it in a folder. In the latter case, the system assumes the record to be relevant and assignes it a positive rating value automatically.

1.3 Users

All users are persons that are registered to the Cyclades environment. A user may act in different roles and switch from one role to another during the same Cyclades session, if she has the rights needed for that role.

- Every user can act as an *single user*.

- A user who is a member of a community can additionally perform actions in a community context, thus acting as a *community member*.

- A community member with appropriate rights can also administrate a community, thus acting as a *community administrator*.

- A user who is a member of a project can perform additional actions in the project context. Then, the user is acting as a *project member*.

- A project member with appropriate rights can administrate a project and its folders. This kind of project member is called a *project administrator*.

- A user with the appropriate rights can create collections and manage those collections she created herself. Having these rights, she is called a *collection registrator*.

- A user with the appropriate rights can register or unregister an archive or edit the registration information concerning an archive. This kind of user is called an *archive registrator*.

- There must be at least one user to grant other users the rights to register archives and collections. This kind of user is called Cyclades *system administrator*.

1.4 Use cases for the single user

The actions contained in these use cases can be performed by any user of the system, but are limited to the user's private folder structure, respectively to those folders in a community or project folder hierarchy where the user has the appropriate rights.

1.4.1 Joining the Cyclades environment

1. Register as a new user
   The user gives her e-mail address and in turn gets a user name and password with which she can login.
2. Login
   The user enters her user name and password. After login, she is presented with her home folder.
3. View and edit user information

1.4.2 Folder management

4. Create folder
   The user fills in the mandatory information for a private folder (title, ...) plus optionally other fields. The system adds the new folder as a subfolder to the current folder (where the user is currently working). There is always a current folder. After login, the current folder is the user's home folder.
5. Manage folders
   Copy, paste, move, delete, split, merge folders, or let the system manage the folder structure.
6. Edit folder content description
   Edit the textual description of a folder, associate or disassociate collections, move or edit queries.

1.4.3 Gathering records

7. Update folder profile
8. Searching and browsing
9. Personalized searching and browsing
10. Receive recommendations from the system

1.4.4 Record management

11. Move or copy a record from one folder to another
12. Delete a record

1.4.5 Exploring and joining communities

13. View a list of communities (with name, description, and contact information) and subscribe to a community
14. Join a community via invitation of an administrator

1.5 Use cases for a community

The following actions can be performed in addition to those for the single user, whenever the user is in a community context (i.e. in a community folder).

1.5.1 Setting up a new community

1. Create a new community folder
   In her home folder, given the appropriate rights, the user can create a new community by creating a community folder. She enters a description for the new community and defines if other users can subscribe without or only with invitation. As creator of a community, she automatically gets the rights to administrate the community and thus becomes an administrator of the community.
2. Invite and kick out members of the community, and define access rights for members
   Only users with appropriate rights, by default the community administrators, can do this.

1.5.2 Joining and leaving a community

3. Register after invitation, or subscribe without being invited (according to the community properties specified by the community administrator).
4. Leave the community by deleting the community folder from the personal home folder

1.5.3 Additional record handling in community folders

5. Rate a record
6. Annotate a record

1.5.4 Awareness

7. Edit notification preferences
   Activity reports may be generated to help the user get aware of changes in the community folders (e.g. new records). The user can define for which actions she wants a report, and when she wants the report to be generated.
8. Catch up

1.5.5 Communicate with other community members

9. Create and add a comment to a discussion forum
10. Online chat (optional)

1.6 Use cases for a project

The following actions can be performed in addition to those for the single user, whenever the user is in a project context (i.e. in a project folder).

1.6.1 Setting up a new project environment

1. Create a new project folder
   The user enters a description for the new project. The creator of a project gets automatically the rights to administrate the group and thus becomes an administrator of the project.
2. Invite and kick out members of the project, and define access rights for members
   Only users with appropriate rights, by default the project administrators, can do this.

1.6.2 Joining and leaving a project

3. Register after invitation

4. Leave the project by deleting the project folder from the personal home folder

1.6.3 Additional folder content handling in project folders

5. Rate a record

6. Annotate a record

7. Upload a document to a project folder

1.6.4 Awareness

8. Edit notification preferences
   Activity reports may be generated to help the user get aware of changes in the project folders (e.g. new documents). The user can define for which actions she wants a report, and when she wants the report to be generated.

9. Catch up

1.6.5 Communicate with other project members

10. Create and add a comment to a discussion forum

11. Online chat (optional)

1.7 Use cases for collection management

1. Create a collection

2. Edit collection description

3. Delete a collection

1.8 Use cases for archive management

1. Register an archive

2. Edit archive registration information

3. Unregister an archive

1.9 Use cases for user archive/collection registration rights

Not every user can register an archive or define a new collection. The right to perform these actions can be granted to a registered user of the Cyclades system by a system administrator. The management of archive and collection registration rights therefore gives rise to the following use cases:

1. A user asks to become an archive registrator
2. A user asks to become a collection registrator
3. Managing registration rights
   An administrator of the Cyclades system grants or denies a user the right to register archives or collections.

Chapter 2

Use cases in detail

This chapter describes the use cases identified in chapter 1 in more detail from the user's perspective. The resulting system activity and internal communication is described in chapter 3.

2.1 Joining the Cyclades environment

The process of becoming a user of the system is called registration. This can be done either by an administrator or via an automated registration process. We assume here the case of an automated registration (otherwise we would need an additional actor and use case for this). The effect of a successful registration is that the user has a Cyclades account consisting of a user name, a password, and a home folder, and may now log into the Cyclades system using user name and password. In addition to the home folder, the user is also allocated with two more specific folders, the waste basket and the clipboard.

Apart from user name, password and home folder, the user may have additional attributes with respect to the Cyclades system. These are full name, address, e-mail address, affiliation etc., but may also include privacy attributes for allowing (or forbidding) certain system behaviour (e.g. to recommend the user to other users of the system as having certain interests, or to propagate the user's ratings of records in her folders along with a recommendation of a record to other users). The additional user attributes are summarized in the user information, which may be viewed and changed by the user while she is a logged into the Cyclades system. Logging into the system is, of course, also a prerequisite for making use of the other available functions of the system.

2.1.1 Register as a new user (SU 1)

We assume that the Cyclades system is available as a free service on the Web and that registration is done via an automated process, i.e. any person may register herself without interference of another person. The only precondition is a valid e-mail address. Only one registration is possible under a given e-mail address.

If a user of the Cyclades system has to pay for the services of the system, the registration process has to be extended to include, e.g. credit card numbers, legal agreements and an actor 'user administrator' who is responsible for authorizing the financial side of this transaction. Also, an additional Billing Service would be needed.

- The user visits the Cyclades Web site (e.g. http://www.cyclades.gr/) and is presented with the Cyclades home page containing information about the system, news, an access for registered users, and an entry point to the registration process for new users.

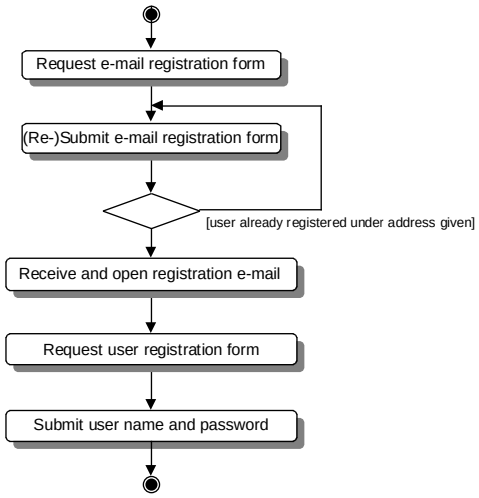

Figure 2.1: Register as a new user - activity diagram

- The user chooses to register and is presented with a registration form requesting a valid e-mail address. The user submits the completed form. If a user has already been registered under the e-mail address given, the user is notified and asked to give a different e-mail address. If not, the registration stops here.
- If no user has been registered under the given e-mail address, the system will send back an e-mail with registration details some time later.
- The user opens the e-mail from Cyclades containing directions and a URL where to register. Hitting on the URL presents a form to enter user name and password. After completion and submission of this form, the user is a registered Cyclades user and is automatically forwarded to her home folder (i.e., can log in with user name and password). Newly registered users have one folder of their own, their (empty) home folder.

2.1.2 Login (SU 2)

Login is done by visiting the Cyclades Web site and using the access for registered users. After login the user is presented with a view of her home folder.

- The user visits the Cyclades Web site and is presented with the Cyclades home page containing (amongst others) a 'Login' button for registered users.
- The user hits this button and is asked for user name and password.
- After having entered the right user name and password, the user is presented with a view of her home folder. The home folder contains only subfolders (private, community or project),

no records, queries, discussions or other documents. For a newly registered user the home folder is empty.

2.1.3 View and edit user information (SU 3)

- The user hits the function for viewing and editing user information stored in the Cyclades system, and is presented with a form containing this information which may now be edited. This information includes full name, affiliation, postal address, phone and fax numbers, and user preferences. User preferences pertain to privacy (e.g. 'Never recommend me to other users of the system'), automatic system action (e.g. 'Always ask me before rearranging my folders'), awareness (e.g. 'Send me an awareness report on create actions in all of my folders every day'), and user interface (e.g. 'Set my user interface language to Greek').

- After having completed the form, the new user information is valid and will be used by the system.

2.2 Folder management

The user creates an environment where to store relevant records by creating private subfolders of her home folder, which are labelled with the names of the domains of her interest. She may prefer a flat structure or a fine-grained hierarchy. The structure may be changed manually by splitting or merging folders, creating new folders, new subfolders or deleting existing folders as is described below (use case SU 5). The name of a folder is meaningful to the user herself.

Apart from the names of the folders, the user may further specify the domain of interest connected with a particular folder by giving an informal textual folder description. In order to focus subsequent search processes, the user may associate certain collections to a folder which may be selected from the list of all existing collections. This information (name, description, associated collections) may be edited by the user any time.

The Cyclades system may also suggest changes in the private folder structure if so requested by the user; these suggestions are inferred from the contents that the user puts into her folders.

Apart from records which are the primary contents of folders, the user may also store interesting or successful queries in her folders that she created and used while gathering records (cf. below). These queries are represented at the user interface as objects that may be moved around or deleted like other objects (folders, records; cf. the use cases SU 5, 11 and 12 for moving and deleting objects at the user interface which are also valid for query objects). The association of a query to a folder is established by saving a query in a folder (cf. use case SU 8); the association is managed by moving and deleting queries. Queries may be reused when searching and browsing.

2.2.1 Create folders and subfolders (SU 4)

- The user selects the function for creation of a new private folder and is presented with a form where name and description for the new folder may be entered, and collections may be associated to the folder (multiple selection lists).

- The user completes and submits the form. This will create a new subfolder of the current folder, i.e. the folder in the user's focus. With a new user this will be the user's (empty) home folder, because this is her only folder.

- The user repeats this action to create a second folder. She navigates to the newly created folder by hitting on the icon or name representing the folder, thus making this folder the current folder.

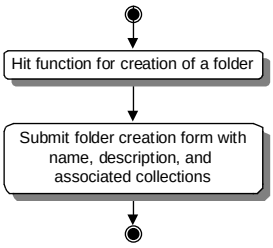

Figure 2.2: Create a folder - activity diagram

- The user proceeds as above to create a new private subfolder of the newly created folder; this newly created folder is then a sub-subfolder of her home folder.

2.2.2 Manage folders (SU 5)

Folder management includes copying and moving folders from one folder to another, deleting folders, and merging and splitting folders. The user may also choose to let the system help her in managing her folders. The use case 'Manage folders' is accordingly split into six use cases 5-1 - 5-6. Folder management makes use of two specific user folders: the waste basket and the clipboard. The waste basket contains deleted objects, and the clipboard contains objects to be transferred from one folder to another. Objects that are shown at the user interface may be selected (and deselected) for further action; this feature is used in the use case descriptions below.

- Copy a folder from one folder to another (SU 5-1)
  - The user selects the folder to be copied in the current folder.
  - The user hits the copy function. The selected folder is transferred to the clipboard.
  - The user navigates to the destination folder making it the current folder.
  - The user hits the paste function. The contents of the clipboard (the folder to be copied) are added to the contents of the destination folder.

  It is clear from the description that the copy operation may also be applied to more than one folder at a time, and that the copy operation may also be applied to different kind of objects at a time, i.e. two folders and three records could be copied by selecting them in step one above, the rest of activity is the same.

Copying a folder copies all its contents.

- Move a folder from one folder to another (SU 5-2)
  - The user selects the folder to be moved in the current folder.
  - The user hits the cut function. The selected folder is transferred to the clipboard and removed from the current folder.
  - The user navigates to the destination folder making it the current folder.
  - The user hits the paste function. The contents of the clipboard (the folder to be moved) are added to the contents of the destination folder.

  Again, it is clear that a whole set of objects may be moved from one folder to another by using the above sequence of steps.

  Moving a folder moves all its contents.

- Delete a folder (SU 5-3)
  - The user selects the folder to be deleted in the current folder.
  - The user hits the delete function. The selected folder is transferred to the waste basket and removed from the current folder.

  A deleted folder may be undeleted again by navigating to the waste basket, selecting the folder to be undeleted, and hitting the undelete function. By hitting the destroy function, the folder is gone for good. Undeleting lets a folder appear in its original parent folder.

  Again, it is clear that a whole set of objects may be deleted, undeleted and destroyed using the above sequence of steps. Deleting, undeleting and destroying a folder deletes, undeletes and destroys all its contents, respectively.

- Split a folder into two (SU 5-4)
  - The user navigates to the parent folder of the folder to be split.
  - The user creates a new subfolder.
  - The user moves some of the contents of the folder to be split to the new subfolder.

- Merge two folders into one (SU 5-5)
  - The user moves the complete contents of one of the folders to the other.
  - The user deletes and destroys the empty folder.
  - The user edits the content description of the merged folder.

- Have the system manage the folder structure (SU 5-6)
  - The user entitles the system to manage the structure of her private folders by setting the respective user preference option in the user information.
  - The system will put suggestions for folder restructuring into the Recommendations folder.
  - The user opens the system suggestion and is presented with a form showing the suggested actions (create new subfolder, split folders, merge folders along with movement of contents).
  - The user approves of these actions or not. In case of approval the suggested actions are executed immediately.

  It would also be possible to have the system manage the folder structure on its own without the prior approval of the user. The use case would become simpler.

2.2.3 Edit folder content description (SU 6)

The contents of a folder are described by name, description, associated collections and associated queries. Since queries are represented differently at the user interface as objects of their own we have two use cases.

- Edit folder name, description and associated collections (SU 6-1)
  - The user requests to update folder details and is presented with a form where the folder attributes name, description and associated collections may be viewed and edited.
  - The user edits this information and submits the form.
- Delete, move, copy queries (SU 6-2)

  This works exactly as for records and is described in SU 11 'Move or copy records' and SU 12 'Delete records'

2.3 Gathering records

2.3.1 Update folder profile (SU 7)

Folder profiles are updated by bringing to bear records that show a shift in the user's interests that are represented by this folder. New records that had been retrieved for this folder and have instead been saved by the user in another folder are an indication that the folder profile should be updated to exclude them (i.e. to avoid "requesting" records similar to them in the future). New records that have been saved by the user in this folder (and that had possibly been gathered for another folder) are an indication that the folder profile should be updated to include them (i.e. to "request" also records similar to them in the future).

The modification of a folder may be explicitly requested by a user (*on-demand folder profile modification*) or may be invoked by the system, typically at regular intervals (*scheduled folder profile modification*). A user folder profile modification is performed for a single folder profile, while scheduled folder profile modification is performed at the same time for all of the folder profiles of a given user. Only on-demand folder profile modification gives rise to a use case, since scheduled folder profile modification occurs in the background, without intervention on the part of the user.

1. On-demand folder profile modification (SU 7)
   - The user decides to request the modification of the profile for the current folder. She does so by hitting the "Modify current folder profile" function. The system modifies the current folder profile and acknowledges to the user the successful completion of the operation.

2.3.2 Searching and browsing without personalization (SU 8)

The user looks for records that are relevant in her current context (i.e. in the folder she is in). See also figure 2.3.

1. Initiate search and browse from the current folder

   The folder where the user starts searching is called the *current folder*. The user clicks on an appropriate button or link, and thus enters the search and browse environment.

2. Select the collection to be searched

   If there are collections associated to the current folder, the user is presented with a list of associated collections, otherwise, she is presented with a list of all available collections to choose from. For each collection, the user can:

   - Browse the collection information (topic, metadata, search functions...)
   - Select this collection to be searched in the query (this automatically unselects any previously selected collection)
   - Unselect the collection, if it was selected before, so that no collection is specified

   If the user doesn't specify which collections to search, then all collections associated to the folder, respectively all collections in the system will be searched.

3. Select an existing query

   If there are queries stored in the current folder, the user can choose one of them to edit and submit in the search process.

   If the user has already submitted queries before (without leaving the search and browse environment), then she can browse a history list of old queries and choose one of them to edit and submit again.

4. Edit query

   If the user doesn't choose an existing query, then she is presented with an empty query, otherwise she can edit the query she chose in the previous step.

   The user can:

   - Add, change or delete query conditions (according to the query function available)
   - Browse the metadata schemas that are available for the collection that is to be searched
   - For each metadata schema, browse attribute values (if possible)
   - Save the query to the active folder

5. Submit query

   After editing the query, the user submits it to the system.

6. Handle results

   The system returns a list of results (i.e. a list of records). According to what the user specified, the records might be shown only partially (e.g. only name, date), or already completely.

   The system maintains a history of result lists. The user can switch between different result lists and save relevant records. Those records are passed to the folder environment where they are stored and thus made persistent.

2.3.3 Personalized searching and browsing of collections (SU 9)

Personalized searching is based on folder profiles. A folder profile is a compact representation of the interests of the user relative to this folder. In the case of community (resp. project) folders, the folder profile is a representation of the interests of the entire community (resp. project). A folder profile is initially created by the system from the textual description associated to the folder by the user. A folder profile is then updated by the system, drawing information from the records for which the user has shown interest (i.e. records which she has saved into this folder) and for which the user has shown lack of interest (i.e. records originally gathered for this folder which she has instead saved elsewhere). In the case of a community (resp. project) folder profile, the saving

actions considered are those performed by any user who has the rights to perform them on this folder, and the identity of the particular user who has actually performed them is irrelevant. Only records are taken in consideration for updating folder profiles; this also means that, in the case of project folders, documents added by members of the project are not considered.

Personalized ad-hoc searching is achieved by using the folder profile as a post-filter on records that are retrieved by *explicit* user queries. *Personalized on-demand searching* is achieved instead by using the folder profile as a post-filter on records that are retrieved by *implicit* user queries, i.e. by user requests to retrieve from the collections associated to the folder all newly gathered records relevant to the folder profile.

2.3.4 Receive recommendations from the system (SU 10)

Recommendations of objects (i.e. records, collections, users, communities) are issued to users based on other users' implicit or explicit ratings of records, and on the perceived similarity between the interests of the user, as represented by a given folder, and the interests of these other users, as represented by their folders. All recommendations are specific to a given user folder; this means that the recommendation of a given object to a user always pertains to a given folder, and has to be understood in the context not of the general interests of the user, but of the interests of the user represented by that folder. Recommendations are also issued to communities and projects based on the ratings provided by all users belonging to the community or project when operating in that community or project. Hereafter, for ease of exposition we will only refer to recommendations to user folders, since recommendations to community folders or project folders operate in a completely analogous way.

Records may be recommended to a user folder only if the system recognizes that this record was saved in a folder (owned by a different user) which is highly similar to this folder. This condition acts only as a prefilter; records to be actually recommended are then chosen from this initially selected pool by estimating the likely rating that the user might give to the record based on a comparison between the rating patterns of the user and those of the users who actually rated the record positively.

Collections are recommended to a user folder by first selecting those user folders which represent the most similar interest patterns with respect to this folder, and then by choosing from the collections associated to these folders only the "top" collections according to some quality criterion.

Users are recommended to a user folder by first selecting those user folders which represent the most similar interest patterns with respect to this folder, and then by choosing from the owners of these folders only the "top" users according to some quality criterion.

Communities are recommended to a user folder by first selecting those community folders which represent the most similar interest patterns with respect to this folder, and then by choosing from the communities owning these folders only the "top" communities according to some quality criterion.

The user may choose whether she wants to receive recommendations relative to a specific folder. The timing of delivering the recommendations is decided by the system, possibly according to a policy that takes into account the different frequencies of update of the different collections (so that folders referring to very "dynamic" collections receive more frequent updates), and possibly according to a policy that differentiates the various types of objects (e.g. communities are recommended less frequently than records). We assume that in the interface recommended records are displayed differently from gathered records; this would allow the user to distinguish records obtained based on other users' ratings from records obtained based on the user's own profile.

We have the following use cases.

1. Activate resp. deactivate recommendations
    - The user decides that new objects of a certain type (records, collections, users, commu-

nities) maybe recommended to the current folder. She does so by hitting the appriopate flag.

2. Receiving recommendations.

   - The system delivers recommendations at system specific time the recommended objects. Recommended records are shown in the relevant folder, like the other records contained therein, and are distinguishable from gathered records by their creator (a pseudo-user "Cyclades Recommender"). Recommended users, communities and collections are indicated by three different icons attached to the relevant folder. Hitting on the "Recommended ⟨objects⟩" icon shows a list of the recommended ⟨objects⟩.

2.4 Record management

Record management includes moving, copying and deleting records. Records are created while gathering records, they are not edited.

2.4.1 Move or copy records from one folder to another (SU 11)

- The user selects one or more records in the current folder.
- The user selects the cut or copy function. The selected records are transferred to the user's clipboard. If the cut function has been hit (move), the selected records are removed from the current folder.
- The user navigates to the destination folder, thus making it the current folder.
- The user hits the paste function. The contents of the clipboard (the records to be moved or copied) are added to the contents of the destination folder.

2.4.2 Delete records (SU 12)

- The user selects one or more records in the current folder.
- The user hits the delete function. The selected records are removed from the current folder and transferred to the user's waste basket.

Deleted records may be undeleted again by navigating to the waste basket, selecting the records to be undeleted, and hitting the undelete function. By hitting the destroy function, the records are gone for good. Undeleting lets the records appear in their original folder.

2.5 Exploring and joining communities

The existence of communities in the Cyclades system is visible to all registered users of the system. While members of a community have access to the community's folders and their contents, non-members may only view some top level information of this community. This information includes the name of the community, its description (by default the description of the community home folder) and the names and e-mail addresses of its adminstrators. Given this information, users may choose to join a community, thus changing from a single user to a community member.

2.5.1 View list of communities and subscribe to a community (SU 13)

- The user selects the 'view communities' function and is presented with a list of the communities that exist on the system. The list of the communities contains the name, description and contact information (name and e-mail address of the administrators) of the community.
- For communities that have chosen to be open for free subscription, the community list contains a 'subscribe' button below the listing of the community. By hitting this button, the single user becomes a member of that community.

2.5.2 Join a community via invitation of an administrator (SU 14)

- The user sends an e-mail to one of the administrators of an interesting community in order to be invited. If the administrator actually invites the user, she becomes a member thereby (cf. the corrsponding use case CO 2-1 'Invite new members').

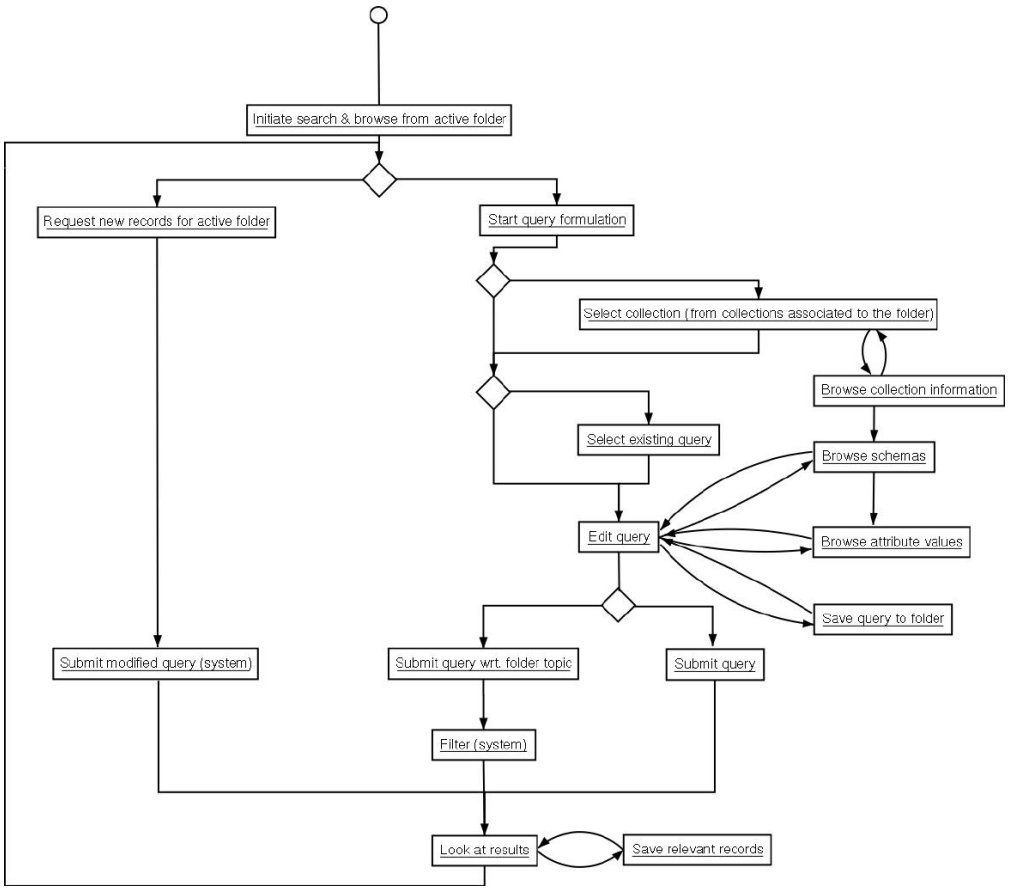

Figure 2.3: Search and browse - activity diagram

2.6 Setting up a new community

Cyclades communities consist of a hierarchy of community folders. At the root of this hierarchy tree is the community's home folder, which may be thought of as representing the community. Community folders (like private folders) may contain only records, no other material. They have the same functionality as private folders with regard to folder management, gathering records, record management (cf. use cases SU 4 - 12). Community folders also have the same attributes as private folders (name, description, associated collections).

Unlike private folders, community folders may be populated with other users via invitation or subscription. All community members share the community home folder of a community, they may leave, however, subfolders they have no interest in. All community members have at least reading access the community's folders. Communities may, however, follow different policies with regard to other access rights (insert/delete records, change folder structure, invite new members). Communities may also choose to be open for free subscription, meaning that registered Cyclades users may become members of the community without being invited by an administrator (or member, if members of that community have the right to invite others).

Home folders of communities are always subfolders of the home folder of the community members.

2.6.1 Create a new community (CO 1)

A community is created by a single user who–in the role of the community administrator–creates a new community by creating its home fold er. Initially, this home folder is empty, and the administrator is the only member of the community.

- The administrator hits the 'Community' function (can only be invoked in the administrator's personal home folder). The administrator is presented with a form where to enter the name, a description and the collections to be associated describing the community's interests.
- The administrator submits this form thereby creating a new (empty) community home folder as subfolder of her home folder with herself as only member.

The administrator may then continue to build an initial structure for the community, and collect a first set of records for these community folders like she would do for a private domain of interest. Alternatively, she may leave this for future members of the community, and continue by inviting these members to the community home folder.

2.6.2 Manage community membership and access rights (CO 2)

Management of community membership and access rights consists of inviting new members to a community, throwing out existing members and defining and changing the access rights for its members. Management functions may be executed in the home folder of the community. Invitation or expulsion have the effect of inviting the new member to, or throwing out the old member from, all folders of the community.

Also ordinary members can execute member management functions if they are given the appropriate access rights by the administrator.

- Invite new members (CO 2-1)
  - The administrator hits the 'Add member' function in the home folder of the community. She is presented with a form where to specify the invitees and the message with which the invitees are invited.

  - There are two alternatives for specifying a new member to invite, either by user name for already registered users or by e-mail address for a future member that still has to register. The invitation of an existing user has the effect that the community home folder appears as a subfolder of the invited user's home folder. The invitation of a new user results in a similar procedure as if this person had tried to register herself (cf. use case CO 3-1 'Register after invitation'). There are also two alternatives for the message to be sent to the invitees: it may be an automated message for which the administrator has to select the appropriate language, or it may be a specific message to be specified by the administrator herself. In the latter case she may also have this message only be sent to the invitees who are no registered users or to all invitees.
  - By submitting this form, all invitees become members of the community. Non-registered invitees will receive an invitation to register. Invitees who are already registered users will have the community home folder with all its subfolders in their personal home folder.

- Kick out existing members (CO 2-2)

  - The administrator hits on the membership icon attached to the community home folder and is presented with a list of the present members of the community. Members shown only with their e-mail address are invited members who have not yet registered.
  - The administrator selects the users she wants to kick out and hits the 'Remove' function. This immediately removes these members from the community having the effect that these user have no longer access to the community's folders.

- Edit access rights of the community (CO 2-3)

  The initial administrator of a community can specify an access policy for a community by specifying access rights for present and future administrators and members. There is a default access right setting when the community has been newly created. Several access right groups are distinguished: read and copy; write and update; delete; invite other members; etc. Access rights may also include the right to change access rights (usually reserved to administrators). Access policies may be liberal or rather restricted. The minimal access right for a community member is read and copy access. Access rights are usually specified for the two roles in a community: administrator and member. These role settings may be overwritten for particular users belonging to the community, if this becomes necessary. The current access right settings may be viewed on the info page of the community's home folder.

  - The administrator views the current access right settings in the info page of the community home folder by hitting the info function for this folder. The info page shows (amongst other information) a table with the access groups as columns and the administrators and members as rows. The current settings are the table values. (In an additional form, the assignment of users to these roles may be managed.)
  - The administrator can select the access function and is presented with a form containing a table of the same format where the table elements may be changed. By submitting this form, the new access right settings are valid for the community.

2.7 Joining and leaving a community

To be able to work inside a community, a user has to become a member of a community. This is done by invitation or subscription. If a member has no longer interest in a community, she may leave the community as is described below.

2.7.1 Register after invitation/subscribe (CO 3)

Registering after an invitation is done as described in the use case SU 1 'Register as a new user' with the exception that the process starts with the recption of the invitation e-mail. Also, the home folder of a new user registering after invitation is not empty but contains already the community home folder. Subscription is even simpler by hitting the subscription button in the community list as described in the use case SU 13 'View communities and subscribe to a community'.

- Register after invitation (CO 3-1)
  - The Cylades system sends an e-mail with registration details.
  - The user opens this e-mail containing directions and a URL where to register. Hitting the URL presents a form to enter user name and password. After completion and submission of this form, the user is a registered Cyclades user and is automatically forwarded to her home folder (i.e., can log in with user name and password). Users that have registered after an invitation have their home folder which contains the home folder of the community to which they have been invited.
- Subscribe to a community (CO 3-2)

  This use case is identical to the use case SU13 'View list of communities and subscribe to a community'.

2.7.2 Leave/Delete a community (CO 4)

A community member who wants to leave a community simply deletes the community home folder from her home folder; the community home folder is then transferred to the user's waste basket (cf. use case SU 5 'Manage folders').

If the member chooses to undelete the community folder from her waste basket, she becomes a member again. By actually destroying the community home folder in her waste basket, she leaves the community for good. So, the user is still able to log into the Cyclades system as a whole, but not into the specific community she has left. Also, the community home folder still exists and is visible for the other community members.

When all members have left a community for good, the community home folder is actually deleted.

2.8 Additional record handling in community folders

Additional functionality in record handling consists in the possibility to rate records with regard to their relevance to the topic concerned, and to annotate records in the form of threaded discussions by the members of the community.

2.8.1 Rate a record (CO 5)

Each member of a community may rate the records contained in the community's folders. Each member has one rating per record. This rating may be changed by the original rater. The info page of a record shows all ratings, the 'top view' of a record as contained in a folder shows a summary rating, e.g. the average or median rating.

- The user selects one or more records she wishes to rate.
- The user hits the function for rating. She is presented with a rating table: the rows correspond to the records to be rated, the columns to the available rating values, e.g. 'irrelevant',

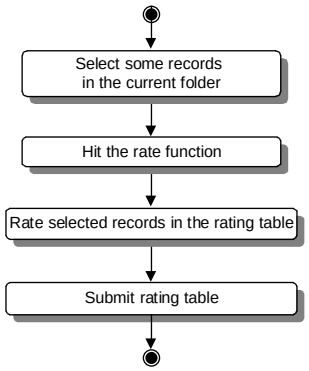

Figure 2.4: Rate a record - activity diagram

'relevant', 'very relevant'. The elements of the table are radio buttons, each row forms a radio button group. There is an additional column 'Not yet rated' which is initially pressed if the user has not yet rated that particular record. Otherwise, the radio button is pressed corresponding to the last rating of that particular record by the user.

- The user enters ratings by pressing radio buttons in the rating table. She may or may not release these ratings to be given away with her name for public use in recommendations.
- By submitting the rating form, the ratings are stored and shown.

2.8.2 Annotate a record (CO 6)

Each member of a community may annotate the records contained in the community's folders. Annotations take the form of discussion forums attached to a record (cf. the use case CO 9 'Create and add a note to a discussion forum' below). Each member of a community may also comment other members annotations, or add her own notes. There is no difference in behaviour, the only difference is in the name: names of discussions may be freely chosen, names of annotations, i.e. discussions attached to a record, have the name 'Notes to: <name of the record>'.

- Add a first annotation to a record (CO 6-1)
  - The user hits the 'Add Note' function attached to the record to be annotated. She is presented with a form to enter a note as annotation of the record. This note has a type (cf. CO 9 below), a subject and the free text annotation as contents.
  - After submission of the form, the annotation is stored and indicated by a respective icon attached to the record. Hitting on this icon will show the annotation.

- Add more annotations to a record (CO 6-2)

  This works exactly as use case CO 9-2 'Add a note to a discussion forum' below. Annotations to a record may be deleted by deleting the contents of the annotation's discussion. The original author may also edit a note in an annotation.

2.9 Awareness

Keeping track of what is going on in a shared environment like community and project folders is of particular importance in an asynchronous mode of working as in the Cyclades system: in the majority of cases users do not use the system at the same time, but at different times. Also, system actions like recommendations may happen at times when the user is not using the system. So there must be ways that a user becomes aware of what other users or system services have done in folders that the user shares with these other users or certain system services.

The Cyclades system supports two ways of becoming aware of other parties' actions: event icons and activity reports per e-mail.

Event icons symbolize events that have occured to or within objects that are displayed at the user interface (folders, records, general documents, queries, discussions). There are different types of events and according event icons: read events, create events, move events, change events. The icons are attached to the objects where the event took place, e.g. the create event icon is attached to a new record in a folder. Folders may have a specific event icon attached ('events inside') indicating that events happened inside the (closed) folder. The user may acknowledge that she has seen the event icons by a so-called catch up action. She may do this for single objects or complete contents of folders.

Activity reports are sent via e-mail to the user and contain information about the event and where it happended, e.g. `|-- CYCLADES|-- System Specification | created by Fischer, 2001-05-22 09:44`. The user may request to receive these reports immediately after the event or to receive a daily summary.

The awareness mechanisms provided by Cyclades can be configured by the user with respect to receiving what type of awareness information for what type of events for what types of objects.

The following use cases describe the configuration of the awareness mechanisms and the catch up action. The actual process of getting aware of changes effected by other users is not described in separate use cases as it happens while reading e-mail or implicitly while viewing the folder structure.

2.9.1 Edit event notification preferences (CO 7)

Awareness configuration in Cyclades is on a per object basis. The default awareness configuration is valid for all folders of a user and their contents. This default configuration may be overridden for particular objects.

- Edit event notification preferences (CO 7-1)

  This use case is a sub use case of the 'view and edit user information' use case (SU 3).

  - The user hits the 'Events' function within the user information editing environment and is presented with a table representing her current settings of event notification preferences. The rows of the table correspond to the activation status of the awareness mechanisms and the event types (read, create, move, change) and the columns to the awareness mechanisms (icons, immediate reports, daily reports). The table entries are check boxes.

  - The user configures her personal preferences by setting or resetting the check boxes in the event notification table. By submitting the table, her preferences are stored and activated.

- Edit event notification preferences for a specific object (CO 7-2)

  - The user hits the info button of the object she wants to change the default event notification for and is presented with detailed information about the object chosen. The object may be a folder, query, record, or discussion forum.
  - The user hits the 'Events' function on the object's info page and is presented with a table representing her current settings of event notification preferences for this object which is equal to the table for editing the default configuration.
  - The user configures her personal preferences by setting or resetting the check boxes in the event notification table. By submitting the table, her preferences are stored and activated for the object chosen.
  - The user may also choose to activate the default configuration for the particular object.

2.9.2 Catch up (CO 8)

- The user selects one or more objects (folders, records, queries, discussions, other documents) she wants to catch up on. This means that she acknowledges to have seen the event icons and wants these icons to go away, so that the screen is less cluttered and new event icons become more prominent.
- The user hits the catch up function. The event icons of the objects that the user has selected in the first step disappear.

Note that the catch up action is on a per user basis. Other users' view of the same folder may be completely different with regard to event icons. The appearance also depends on the event notification preferences of a user: some users may have chosen to receive no event icons at all.

2.10 Communicate with other community members

Communication with other community members is supported in two forms in Cyclades: by discussion forums and an optional on-line chat facility.

2.10.1 Create and add a note to a discussion forum (CO 9)

Each member of a community may start discussion forums in the folders of a community. These discussion forums work like threaded discussions. Once started, members may add notes as comment on other people's notes or as starting points for new threads. Notes have types indicated by icons (just 'Note', or 'Pro'/'Con', 'Angry', 'Important', 'Idea').

- Create a discussion forum (CO 9-1)

  - The user hits the function for adding a discussion forum to the current folder.
  - The user is presented with a form where to enter the name of the discussion, and type, subject and contents of the first note of this discussion. This first note is the first top level thread of the discussion.
  - By submitting this form, a new discussion is created and shown as a separate object in the folder contents.

- Add a note to a discussion forum (CO 9-2)
  - The user hits on the name or icon representing the discussion she wants to add a note to and is presented with a view of the discussion showing all existing notes organized in threads.
  - The user may now choose to add a note as a comment to an existing note by hitting the 'Reply' function attached to an existing note and is presented with a form where to enter type, subject and contents of the new note.
  - By submitting this form, the new note is added to the discussion as a comment to the note chosen.
  - The user may also choose to start a new thread by hitting the 'Add Note' function which will present the user with a form to enter a new note (type, subject, contents as before).
  - By submitting this form, the new note is added to the discussion as the starting point of a new top level thread.

  Discussions may be moved, copied and deleted like folders (cf. use case SU 5 'Manage folders'). Notes within discussions may also be moved, copied and deleted like records in a folder (cf. record management use cases SU 11 and 12). Moving and copying of notes is limited to within and between discussions.

2.10.2 Online chat (optional; CO 10)

Users who got information about other parties with similar interests—either as a single user or as the administrator of a community—can contact the other party via an online chat. The user checks if the other party is online, in case the other party is online, the user starts an online real-time text-chat with the other party.

2.11 Collection management use cases

2.11.1 Create collections (COL 1)

Collections can be created by the Cyclades system administrator and by any user who has the collection registrator rights, by the community administrators and by the archive registrators. The Cyclades system administrators can edit/remove any collection, the collection registrators, the community administrators and the archive registrator can only edit/remove the collections that they have created.

The creation of a collection answers to the need of identifying a subset of the global information space that is meaningful for a specific group of users. The creator will usually browse the list of existing collections in order to find out whether a collection meeting the required characteristic is found. If not, she will create a new one. The creation of a collection is done by specifying the collection membership condition, the identifier of the user who makes the creation request and other specific information. The membership condition is a condition on a set of fields which include the Dublin Core metadata fields plus other specific fields, like availability of metadata descriptions in a given format, or the identifier of the archive that maintains the documents. All the documents that satisfy the membership condition up to an established threshold belong to the collections. A collection is created only if the specified user has the required rights, otherwise an error message is returned. A new collection is associated with an empty set of search and browse formats.

Each time a new archive is registered, the archive registrator also creates a new collection that maintains all the documents of the new archive.

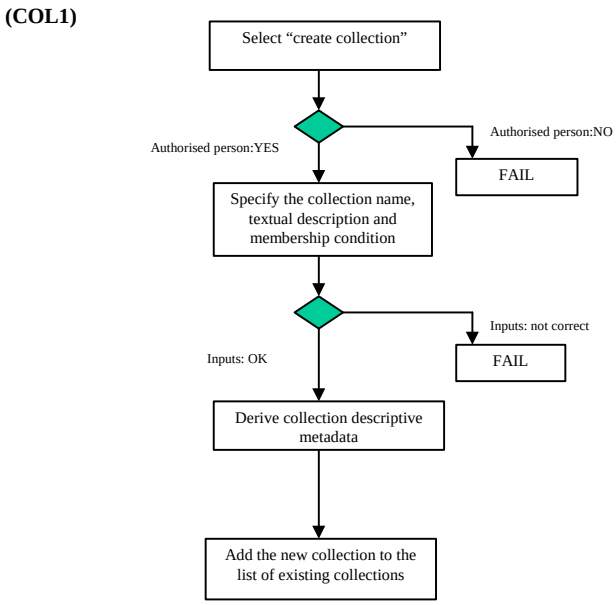

Figure 2.5: Create a collection - activity diagram

As a default, the system maintains a collection named *Cyclades* which includes all the documents harvested by Cyclades. Dublin Core is the only metadata format avalable for searching and browsing on this collection.

As a result of a successful creation the system generates the collection metadata set. This set contains the information that is the Cyclades services need to support a collection-based structured view of the information space. In particular, it maintains the list of archives that disseminate descriptive records about the documents in the collection and the *collection filtering condition*. This condition allows to select, among the records harvested by the associated archives, those belonging to the collection.

The activity diagram (figure 2.5, COL 1) shows the steps involved in the creation of a collection.

2.11.2 Add search/browse format (COL 2)

For each existing collection the collection creator can define which format can be used for searching and browsing on that collection. Regarding the search, the format determines the query condition fields, the schema of the records returned as result and how they are displayed. As far as the browse, it specifies the browsable fields, and again, the schema of the records returned as result and how they are displayed. The system allows to associate more than one search/browse format with the same collection. For example, simple and refined search formats may be associated with a collection in order to support different search granularity.

Search and browse formats can be added to any exiting collection. Only the creator of the collection and the collection administrator have the rights of associating new search and browse formats.

The adding process starts by browsing the list of schemas supported for that collection and then

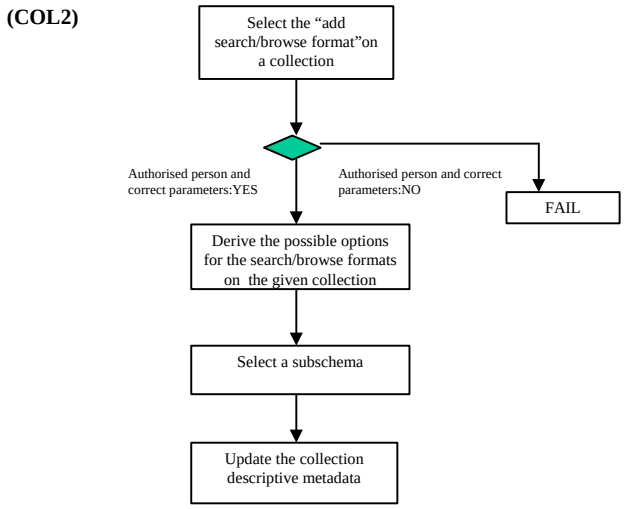

Figure 2.6: Add a search/browse format for a collection - activity diagram

selecting one of them. In so doing the creator defines which will be the schema of the records returned by the search/browse. Then the subset of the selected schema that will characterize the new search/browse format is chosen. This defines the subschema that will be available in the formulation of the query and that will be displayed as result of a search/browse operation. If the specified subschema is legal and who require the addition of a format has the rights to do it, i.e. he/she is either the collection owner or the collection administrator, the new search/browse format becomes available in all the folders that had selected the corresponding collection.

The activity diagram (figure 2.6, COL 2) shows the steps involved in the creation of search and browse format.

2.11.3 Delete collections

A collection can be removed by its owner or by the collection administrator. The request must specify the name of the collection. In order to select the precise identifier of the collection he/she can browse the list of available collections. The deletion is performed only if the specified collection exists and if the requester is an authorised user. A deleted collection is not anymore accessible in any of the folders that refers it.

2.11.4 Remove search/browse formats

A search and browse formats on a collection can be removed by sending a request that specifies the name of the collection and the format to be removed. The request is satisfied only if it refers to an existing collection and an existing format and if the requester is either the owner of the collection or the collection administrator. When a format is deleted, the corresponding search and browse

operations becomes not accessible anymore in any of the folders that refers it.

2.11.5 Edit collection descriptive metadata

The owner of a collection or the collection administrator may decide to change some of the collection metadata such as, for example, the textual description. In order to it, it invokes a request by specifying the name of the collection and the collection metadata field to be changed. This request is satisfied by the system only if the collection exists, the specified field can be modified and the requester is an authorised user.

2.12 Archive management

The Cyclades system has to know about the archives it should gather records from, and for each archive, which metadata formats it should use. Furthermore, in order to define collections later on, the user might need a description of the archive content. This information on the individual archives can change with time (e.g., there can be additional metadata formats, or the topic can shift), so it must be possible to edit this information later.

2.12.1 Register an archive

- The user enters the archive registration environment and chooses the function to add an archive.
- The user enters the primary url for the archive.
- The user is presented with the metadata schemas available for the new archive and chooses the schemas to be used in the record gathering process. Only records in these formats will be available to the Cyclades system from this archive.
- The user enters additional information about the archive, i.e. a plain text description, languages and dates covered, additional urls (mirros), topics and so on.
- The user confirms the archive registration and is presented with a success message or a failure report.

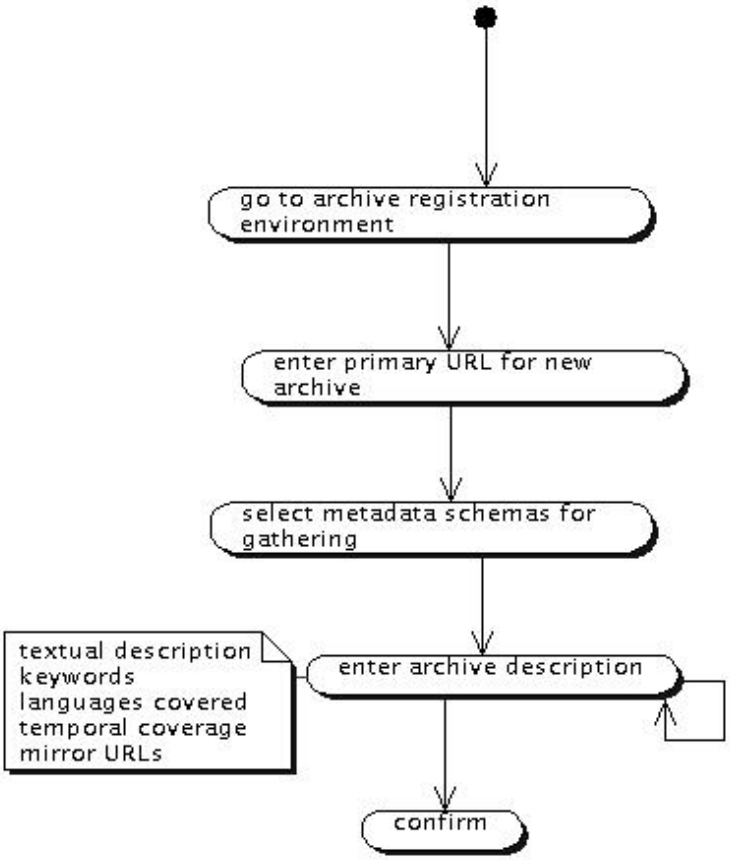

Figure 2.7: Register an archive - activity diagram

2.12.2 Edit archive registration information

- The user enters the archive registration environment and chooses the function to edit the registration information for an archive.
- From a list of archives (depending on the user's rights), the user chooses the archive she wants to manage.
- The user adds or deletes the schemas to be used in the record gathering process. Only records in these formats will be available to the Cyclades system from this archive.
- The user edits the additional information about the archive, i.e. a plain text description, languages and dates covered, additional urls (mirros), topics and so on.
- The user confirms the new archive information and is presented with a success message or a failure report.

2.12.3 Unregister an archive

- The user enters the archive registration environment and chooses the function to unregister an archive.
- From a list of archives (depending on the user's rights), the user chooses the archive she wants to unregister.
- After the user has confirmed this action, the archive is no longer available to Cyclades. No new records will be gathered from this archive, no changes propagated.

2.13 Management of archive/collection registration rights

2.13.1 Becoming an archive registrator

The user wants to register a new archive, but has not yet the appropriate rights to do this. She chooses the function "become an archive registrator" and is presented with a form where she fills in her name, e-mail address, professional background and a description of the archive she wants to register. After the Cyclades system administrator has granted or denied her the right to register archives, the user is notified via e-mail.

2.13.2 Becoming a collection registrator

The user wants to create a new collection, but has not yet the appropriate rights to do this. She chooses the function "become an archive registrator" and is presented with a form where she fills in her name, e-mail address, professional background and a description of the archive she wants to register. After the Cyclades system administrator has granted or denied her the right to register collections, the user is notified via e-mail.

2.13.3 Managing registration rights

Using an appropriate user interface, the administrator of the Cyclades System specifies, a user's rights to register archives or collections. The administrator specifies userId, the type of access right (archive or collection) and whether this access right should be enabled or disabled.

Chapter 3

System components, interaction, and class specifications

In this chapter, we identify the main components of the system, describe their interaction required for the use cases (see chapter 2) and specify their publicly known classes with their respective fields and methods. Finally, we give a short sketch of the query languge.

3.1 System components

The Cyclades system provides the user with different environments according to the actions the user wants to perform. Most of the user's work will be done in her folders, i.e. in a folder environment. As all the actions in this environment closely relate to each other, they will be implemented by one system component, called the *Collaborative Work Service* (CWS). This service also includes a rating management component (RMS).

Searching and browsing is an activity loosely connected to a folder, but otherwise autonomous. Thus, it will be implemented by a separate system component, called the *Search and Browse Service* (SBS).

Although the Cyclades system will deal with open archives only, for the beginning, it is meant to be open for other data sources in the future. Thus, the access of the actual metadata is implemented in a separate system component, the *Access Service* (AS). This way, in order to use data from other kinds of archives in the future, only an appropriate Access Service will have to be implemented.

The management of collections is a separate task that will be implemented by yet another separate system component, the *Collection Service* (CS).

As the system could work without recommendation and personalization facilities, and as there exist a number of different personalization paradigms, this functionality will also be implemented by a separate system component, the *Filtering and Recommendation Service* (FRS). Thus, the recommendation and personalization paradigms of the whole system can be changed easily by replacing the Filtering and Recommendation Service with one that implements other algorithms.

All of these services should be able to work together in a distributed environment. Security and system administration should be provided for centrally. Therefore, we define a system component to mediate between the individual services and to manage user sessions: the *Mediator Service* (MS).

In the remainder of this chapter, we describe the interaction of the different services for the individual use cases (see chapter 2), and name the main classes that will be used to implement the services.

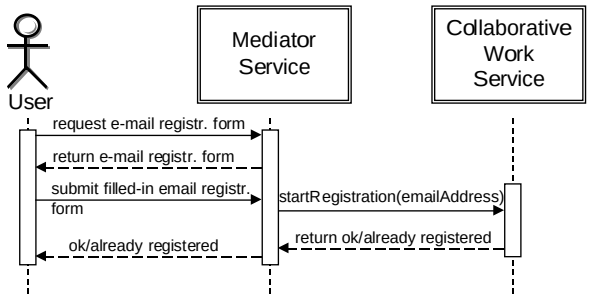

Figure 3.1: Registration (Mediator perspective) - interaction diagram

3.2 System activity and service interaction

In this section, we describe the system activity and the interaction between the services for some of the use cases we listed in chapter 2. As the other use cases result in similar interaction, or in no service interaction at all, we restrict our description to those listed in the following.

3.2.1 Registering as a new user (SU 1) - Mediator perspective

- The user requests the e-mail registration form from the Mediator Service by hitting the Register function.
- After this form has been submitted with an e-mail address, the Mediator Service forwards this address to the Collaborative Work Service via the method `startRegistration`.
- The Collaborative Work Service either returns `true` when no user so far has registered under the address given, or returns `false` otherwise.
- In the first case, the Mediator Service confirms the successful submission of an e-mail address for registration, in the second case it produces an error message along with a form to re-submit an e-mail address.

3.2.2 Login

The user is performing the process of logging into the system by filling and submitting her username and password to the Mediator Service.

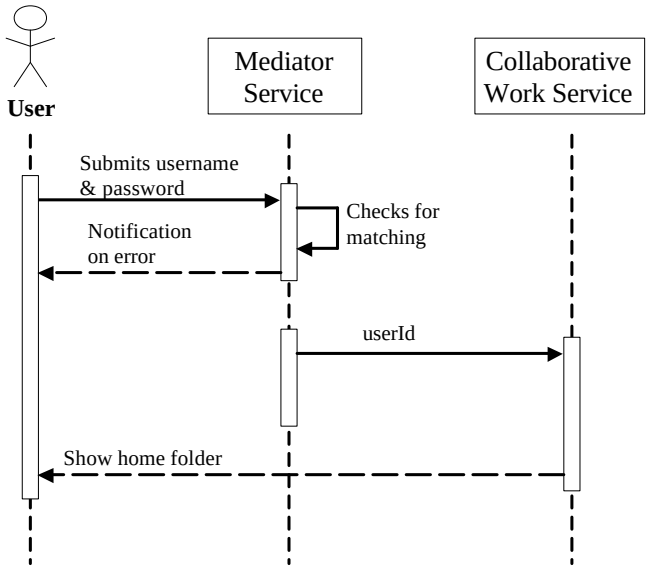

Figure 3.2: Login - interaction diagram

- The Mediator Service receives the username and password of the user
- The Mediator Service checks for matching between that particular username and password
- On error Mediator Service notifies the user
- Otherwise it sends that user's id to the Collaborative Work Service, which then presents the user with her home folder.

3.2.3 Registering as a new user (SU 1) - CWS perspective

The process of registering as a new user consists of two steps: the user submits her e-mail address, and afterwards receives an e-mail message containing a URL with the location where to register. The URL contains as parameter a personal registration token for the user, thus ensuring that the registered user actually is in possession of the e-mail address given in the first step.

- The Collaborative Work Service receives an e-mail address via the `startRegistration` method from the Mediator Service in order to register a new user under address given.
- The Collaborative Work Service checks whether a user has already been registered under the e-mail address given.
  - If so, it returns `false` to the Mediator Service.
  - If not, it creates a preliminary user object[1] and returns `true` to the Mediator Service.

[1] A preliminary user object is a user object that does not have a proper user environment (home folder, waste basket, user info etc.) but simply consists of an e-mail address.

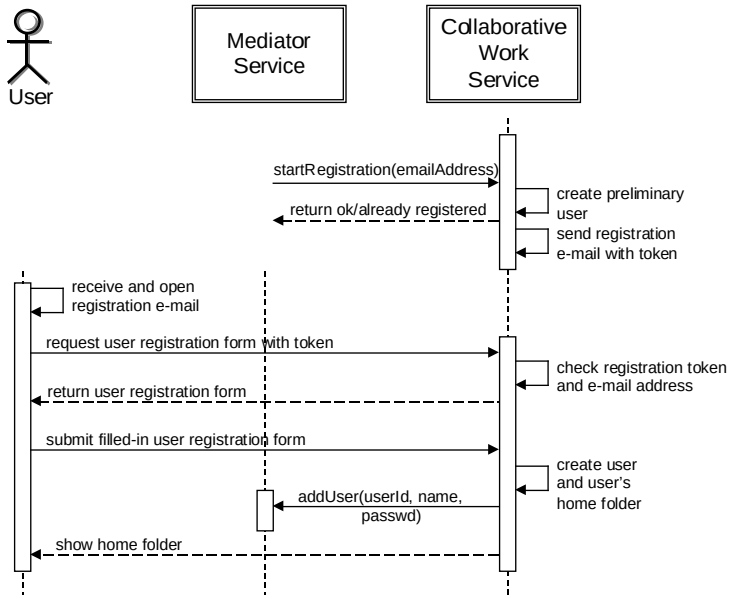

Figure 3.3: Registration (CWS perspective) - interaction diagram

- After creation of the preliminary user object, the Collaborative Work Service constructs the registration URL along with the token and sends it to the e-mail address given. This ends the first step of the registration process.

- When the user receives the registration e-mail, opens it and hits the registration URL, the Collaborative Work Service checks the token and the e-mail address and provides the user registration form. In case the token and the e-mail address do not match, an error message is sent back.

- After the user has submitted user name and password with the user registration form, the Collaborative Work Service creates a user object complete with user environment, notifies the Mediator Service of the newly registered user via the `addUser` method, and presents the new user with her home folder.

3.2.4 Updating a folder profile on user demand (SU 7), CWS perspective

- The user decides to request the updating of the profile for the current folder. She does so by hitting the Update profile function.

- The Collaborative Work Service forwards this request to the Filtering & Recommendation Service via the `updateFolderProfile` method.

- After the Filtering & Recommendation Service has returned an acknowledgement of the successful folder profile update, the update is confirmed to the user.

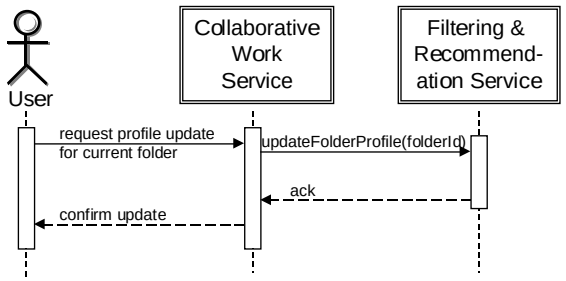

Figure 3.4: On-demand folder profile update (CWS perspective) - interaction diagram

3.2.5 Activating recommendations (SU 10-1)

- The user decides to activate the recommendation service for the current folder. She does so by hitting the Activate recommendations function.
- The Collaborative Work Service forwards this request to the Filtering & Recommendation Service via the `setRecommendationYesNo` method; the service will now start to produce recommendations for the current folder.

3.2.6 Deactivating recommendations (SU 10-6)

- The user decides to deactivate the recommendation service for the current folder. She does so by hitting the Deactivate recommendations function.
- The Collaborative Work Service forwards this request to the Filtering & Recommendation Service via the `setRecommendationYesNo` method; the servcie will now stop to produce recommendations for the current folder.

3.2.7 Inviting new members (CO 2-1) and registering after invitation (CO 3-1)

If a user has the necessary access rights within a community or a project, she as the inviter may invite another person—the invitee—to become a member of this community or project, even if the invitee is not yet a registered user of the Cyclades system.

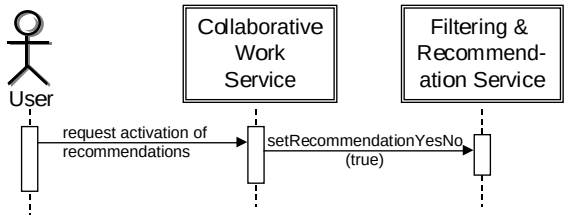

Figure 3.5: Activate recommendations - interaction diagram

- The inviter invites new members to a community or project by hitting the Add member function which presents a form where she may enter already registered users by user name and not registered persons represented by their e-mail address.

- Upon submission of the invitation form, the Collaborative Work Service creates preliminary user objects for those new members who are given as e-mail addresses and are not already registered under this e-mail address. It then constructs registration URLs along with the respective tokens and sends them to the e-mail addresses given.

- As a response to the inviter, the Collaborative Work Service indicates the new members at the folders of the community or project to which they where invited.

- When an invitee receives the registration e-mail, the rest of the registration process is very similar to the second phase of the registration of a new user who registers on her own initiative. The invitee opens the e-mail and hits the registration URL.

- The Collaborative Work Service checks the registration token and the e-mail address. In case the token and the e-mail address do not match, an error message is sent back, otherwise a user regsitration form is provided to the user.

- After the invitee has submitted user name and password with the user registration form, the Collaborative Work Service creates a user object complete with user environment, notifies the Mediator Service of the newly registered user via the `addUser` method, and presents the new user with the folder to which she has been invited.

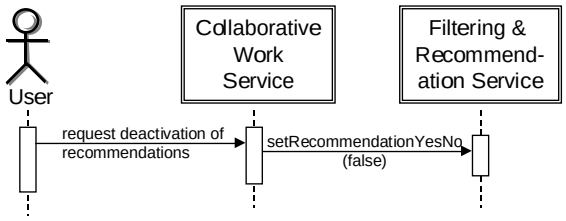

Figure 3.6: Deactivate recommendations - interaction diagram

3.2.8 Rating records (CO 5)

- The user rates records by selecting a number of records and hitting the Rate function. She then assigns a rating value to each record that is shown in the rating form.
- After the user has submitted the ratings, the Collaborative Work Service stores the ratings in the record objects concerned. It then notifies the Rating Management Service of these ratings by repeatedly calling the `saveRating` method of this service for each record that has been rated. As parameters of the method all necessary data are transmitted: the IDs of the record that has been rated, of the user that has rated, and of the folder in which the rating has happened plus the rating value and the time of the rating.
- The Rating Management Service stores these data in its database for further use by the Filtering & Recommendation Service.
- The Collaborative Work Service indicates the new ratings at the records concerned.

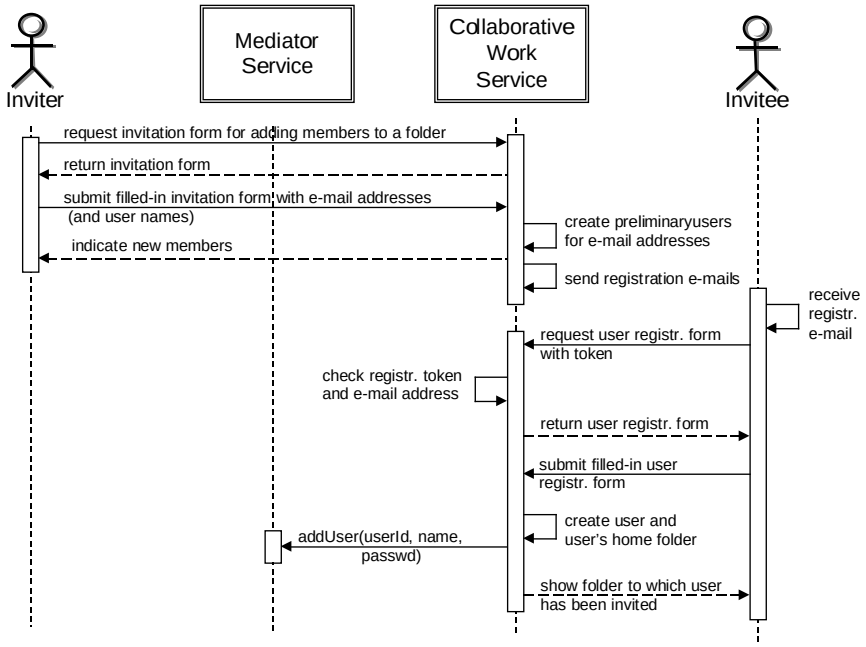

Figure 3.7: Invitation - interaction diagram

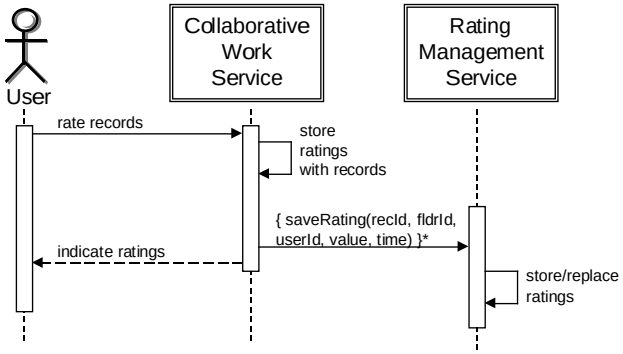

Figure 3.8: Rate records - interaction diagram

3.2.9 On-demand folder profile modification (SU 7-1)

- The Filtering & Recommendation Service receives a request to modify the profile for the current folder. It then requests to the Collaborative Work Service the set of record IDs that have been saved into the folder after the last time this folder profile has been updated; this is indicated by the `profileUpdateTimeStamp` (there is one for each folder), maintained by the Filtering & Recommendation Service.
- After receiving this set of record IDs, the Filtering & Recommendation Service requests to the Access Service the set of indexed terms corresponding to each of these record IDs (i.e. a vector of weighted terms for each record).
- After receiving this set of indexed terms, the Filtering & Recommendation Service updates the folder profile, updates the `ProfileUpdateTimeStamp` for the folder profile, and acknowledges completion of the operation to the requesting service.

3.2.10 Scheduled folder profile modification (SU 7-2)

- The Filtering & Recommendation Service requests to the Mediator Service the user IDs of all users.
- Upon receiving these IDs, for each user ID the Filtering & Recommendation Service requests from the Collaborative Work Service the set of folder IDs associated to this user.
- Upon receiving these IDs, for each folder ID the Filtering & Recommendation Service requests to the Collaborative Work Service the set of record IDs that have been saved into the folder after the last time this folder profile has been updated; this is indicated by the `profileUpdateTimeStamp`, one for each folder, which is maintained by the Filtering & Recommendation Service.
- After receiving this set of record IDs, the Filtering & Recommendation Service requests to the Access Service the set of indexed terms corresponding to these record IDs (i.e. a vector of weighted terms for each record), and requests to the Rating Service the ratings that other users have given to the records.
- After receiving the results from both Access Service and Rating Service, the Filtering & Recommendation Service updates the folder profiles and then updates the `profileUpdateTimeStamp` and the `ratingUpdateTimeStamp` corresponding to the updated folder profiles.

3.2.11 Gathering records without personalization (SU 8)

When a user wants to enter search and browse mode, the Mediator calls the Search and Browse Service with the user id, and with the id of the user's current folder.

- The Search and Browse Service asks the Collaborative Work Service for the id of the collections associated to the folder. Then, the user is presented with the Search and Browse Service's interface.
- If the user wants to choose a collection to be searched, the Search and Browse Service asks the Collection Service for the metadata of all the collections associated to the folder and presents the user a list to select a collection from.
- The user selects a collection (optional step). If the user doesn't choose a collection, then the search will use all the collections available.
- The Search and Browse Service asks the Access Service for the metadata schemas available for the chosen collection (or available at all, if no collection was specified).

- The user browses the available metadata schemas and chooses one for the search.
- For some metadata schema and attributes, the user can also browse the existing attribute values. In this case, the Search and Browse Service retrieves a list of attribute values for a given schema and attribute from the Access Service.
- Now, the user can create or edit a query. The Search and Browse Service asks the Collaborative Work Service for the queries that are stored in the user's current folder. The user can select one of these queries, a query from the Search and Browse Service's query history (if the user has already submitted previous queries from the same search and browse window), or formulate a new query from scratch.
- The user edits the current query.
- The user may also save the query to the current folder. In this case, the Search and Browse Service sends to the Collaborative Work Service the folder id, the user id, and the query to be stored.
- The user submits the query without the personalization option. This results in the Search and Browse Service submitting the query directly to the Access Service. It gets back a weighted list of records with further information and displays it in a result list.
- The user browses the result list, or chooses previous result list (if existing) to browse.
- The user marks some records as relevant and requests to save them to the current folder. The Search and Browse Service sends the list of relevant records, the folder id and the user id to the Collaborative Work Service.
- The user returns to some previous function, or quits the search and browse environment.

3.2.12 Personalized ad-hoc searching (SU 9-1), SBS perspective

If the user chooses the personalization option when submitting the query, then the Search and Browse Service sends the request to the Filtering and Recommendation Service, instead of sending it to the Access Service directly (for the interaction of the Filtering and Recommendation Service, see subsection 3.2.14).

3.2.13 Personalized on-demand searching (SU 9-2), SBS perspective

Directly after entering the search and browse environment, the user can also request new records for the collections of her current folder, without having to formulate a query herself.

- In this case, the Search and Browse Service asks the Collaborative Work Service for the IDs of the collections associated to the current folder.
- The Search and Browse Service asks the Filtering and Recommendation Service for new records for this folder and collections.
- The Filtering and Recommendation Service returns a set of records.
- The Search and Browse Service treats these records as a result set. The user can browse this set or other result sets, mark records as relevant, and save them to the folder (like in subsection 3.2.11).

3.2.14 Personalized ad-hoc searching (SU 9-1), FRS perspective

- The Filtering & Recommendation Service receives a request to evaluate a given query with respect to a given folder topic (*personalized ad-hoc searching*); the maximum number of documents to be returned is also indicated.
- The Filtering & Recommendation Service passes the query to the Access Service, indicating the maximum number of documents to be returned and a void timestamp (e.g. the one corresponding to the time of initialization of the system).
- The Access Service returns the set of records requested, together with an internal representation for each of them consisting of a set of indexed terms.
- The Filtering & Recommendation Service filters the records received from the Access Service (using the indexed terms), returns the filtered records to the Search & Browse Service, and moves to Step 3.2.14.
- [Look at results (from history/folder) (as in non-personalized search)]

3.2.15 Personalized on-demand searching (SU 9-2), FRS perspective

- The Filtering & Recommendation Service receives a request to check whether new records relevant to the topic of a folder have been gathered for a given collection since the user last checked this (*personalized on-demand searching*); the maximum number of documents to be returned is also indicated.
- The Filtering & Recommendation Service issues a query to the Access Service asking for new records gathered for the collection after the `onDemandTimestamp` relative to the current folder (which specifies the time of the last such query issued from this folder).
- The Access Service returns the set of records requested, together with an internal representation for each of them consisting of a set of indexed terms.
- The Filtering & Recommendation Service filters the records received from the Access Service (using the indexed terms), and returns the filtered records to the requesting Service. The Filtering & Recommendation Service performs the filtering operation based on the profile it maintains for this folder.

3.2.16 Receive record recommendation from the system (SU 10-1)

- The Filtering & Recommendation Service selects the k most similar folders to the current folder. This selection activity uses both folder profiles, which are persistently stored by the Filtering & Recommendation System, and "folder rating profiles" (i.e. normalized vectors of the ratings given by a user within the folder), which are also persistently stored by the Filtering & Recommendation System.
- For each of the k folders selected in the previous step, the Filtering & Recommendation Service requests to the Collaborative Work Service the records contained in the folder that have been gathered after `recommendedRecordsTimeStamp`, a timestamp maintained by the Filtering & Recommendation Service that records the last date in which this folder was selected for recommending records to the current folder (`recommendedRecordsTimeStamp` is thus an attribute of a pair of folders, and not of a single folder).
- For each of the record IDs returned, the Filtering & Recommendation Service computes a prediction of the likely rating that the user might give to the record. This computation is based on a comparison between the rating patterns of the user and those of the users who actually rated the record positively.

- The Filtering & Recommendation Service selects the s "top" records (i.e. the ones with the highest predicted rating), where s is determined according to some thresholding policy, communicates these recommendations to the Collaborative Work Service, and updates the `recommendedRecordsTimeStamp` of the k selected folders with respect to the current folder.

3.2.17 Receive collection recommendation from the system (SU 10-2)

- The Filtering & Recommendation Service selects the k most similar folders to the current folder, in the way specified for SU 10-1, and requests to the Collaborative Work Service the IDs of the collections that are associated to these folders.
- Among the collection IDs returned by the Collaborative Work Service, the Filtering & Recommendation Service selects the s "top" collections according to some quality criterion (e.g. the collections that are referred to by the highest amount of selected folders), where s is determined according to some thresholding policy.
- Among the s pre-selected collections, the Filtering & Recommendation Service selects the $s' \leq s$ collections that have not yet been recommended to the current folder, checking against `recommendedCollections`, a list of collections already recommended to the current folder which is maintained by the Filtering & Recommendation Service.
- The Filtering & Recommendation Service communicates the recommendations for these s' collections to the Collaborative Work Service, and adds these s' collections to the `recommendedCollections` list of the current folder.

3.2.18 Receive user recommendation from the system (SU 10-3)

- The Filtering & Recommendation Service selects the k most similar folders to the current folder, in the way specified for SU 10-1, and requests to the Collaborative Work Service the IDs of the owners of these folders.
- Among the user IDs returned by the Collaborative Work Service, the Filtering & Recommendation Service selects the s "top" users according to some quality criterion (e.g. the users who own the highest amount of selected folders), where s is determined according to some thresholding policy.
- Among the s pre-selected users, the Filtering & Recommendation Service selects the $s' \leq s$ users that have not yet been recommended to the current folder, checking against `recommendedUsers`, a list of users already recommended to the current folder which is maintained by the Filtering & Recommendation Service.
- The Filtering & Recommendation Service communicates the recommendations for these s' users to the Collaborative Work Service.
- The Collaborative Work Service selects from these pre-selected s' users the $s'' \leq s$ users who have declared their willingness to be recommended to other users, and recommends them to the current folder. In the meantime, the Filtering & Recommendation Service adds these s' users to the `recommendedUsers` list of users already recommended to the current folder.

3.2.19 Receive community recommendation from the system (SU 10-4)

- The Filtering & Recommendation Service selects the k most similar community folders (i.e. folders owned by a community) to the current folder, in the way specified for SU 10-1, and requests to the Collaborative Work Service the IDs of the communities that are associated to these folders.

- Among the communities returned by the Collaborative Work Service, the Filtering & Recommendation Service selects the s "top" communities according to some quality criterion (e.g. the communities who own the highest amount of selected folders), where s is determined according to some thresholding policy.
- Among the s pre-selected collections, the Filtering & Recommendation Service selects the $s' \leq s$ communities that have not yet been recommended to the current folder, checking against `recommendedCommunities`, a list of communities already recommended to the current folder which is maintained by the Filtering & Recommendation Service.
- The Filtering & Recommendation Service communicates the recommendations for these s' communities to the Collaborative Work Service, and adds these s' communities to the `recommendedCommunities` list of communities already recommended to the current folder.

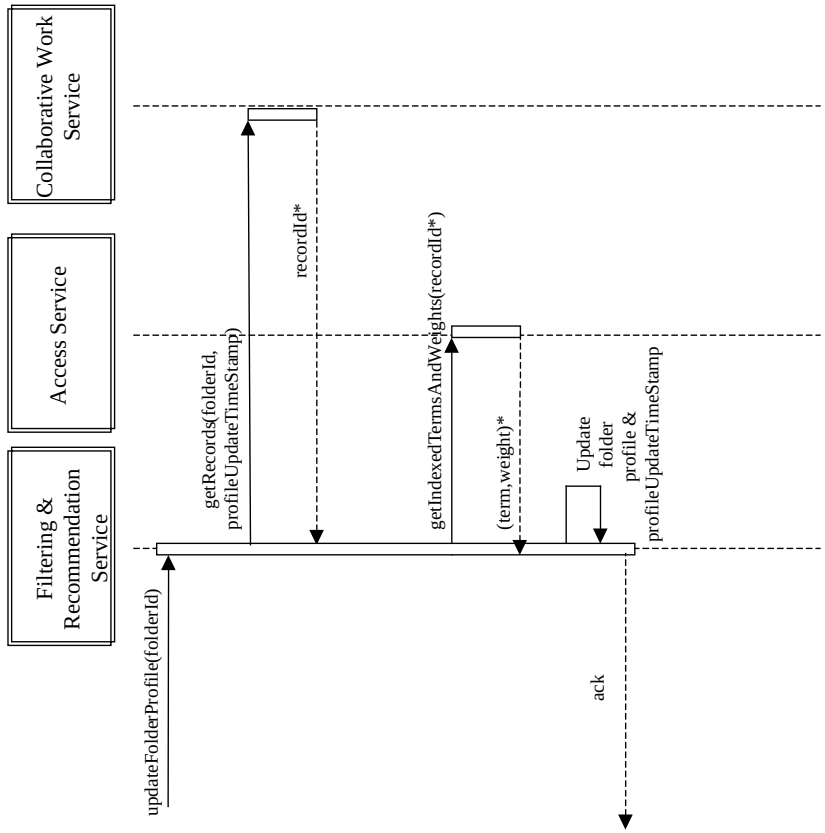

Figure 3.9: On-demand folder profile modification - interaction diagram

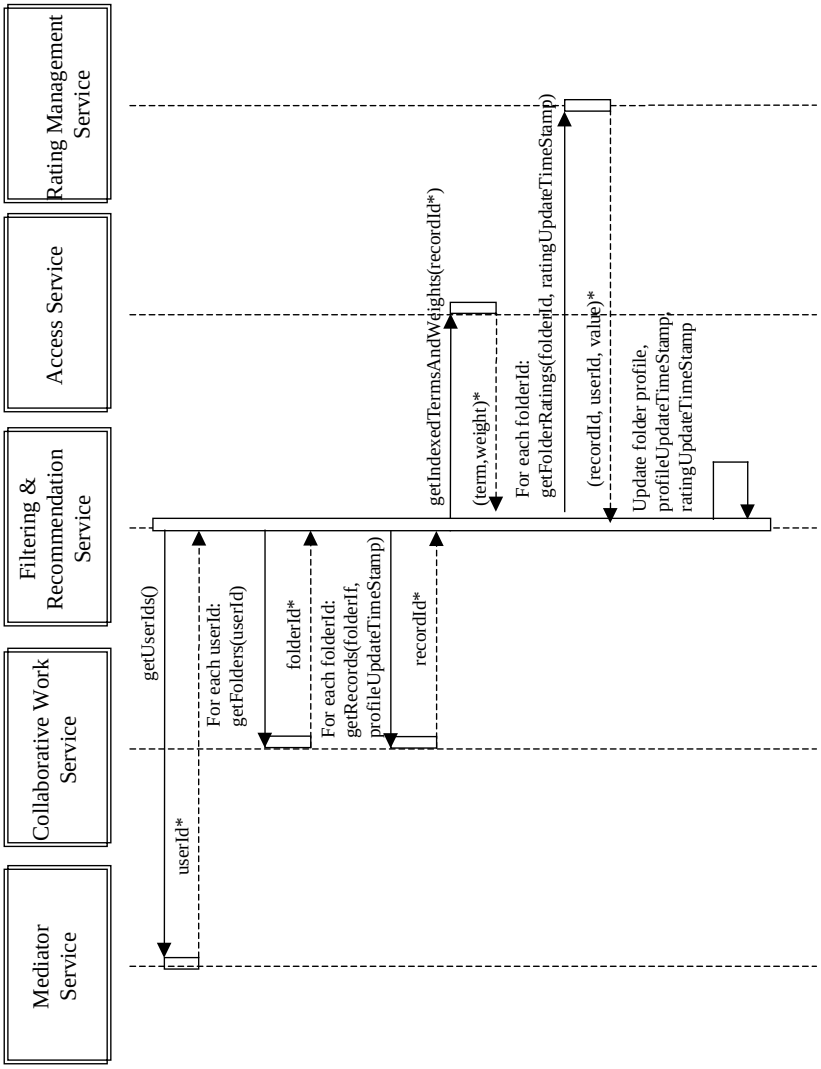

Figure 3.10: Scheduled folder profile modification - interaction diagram

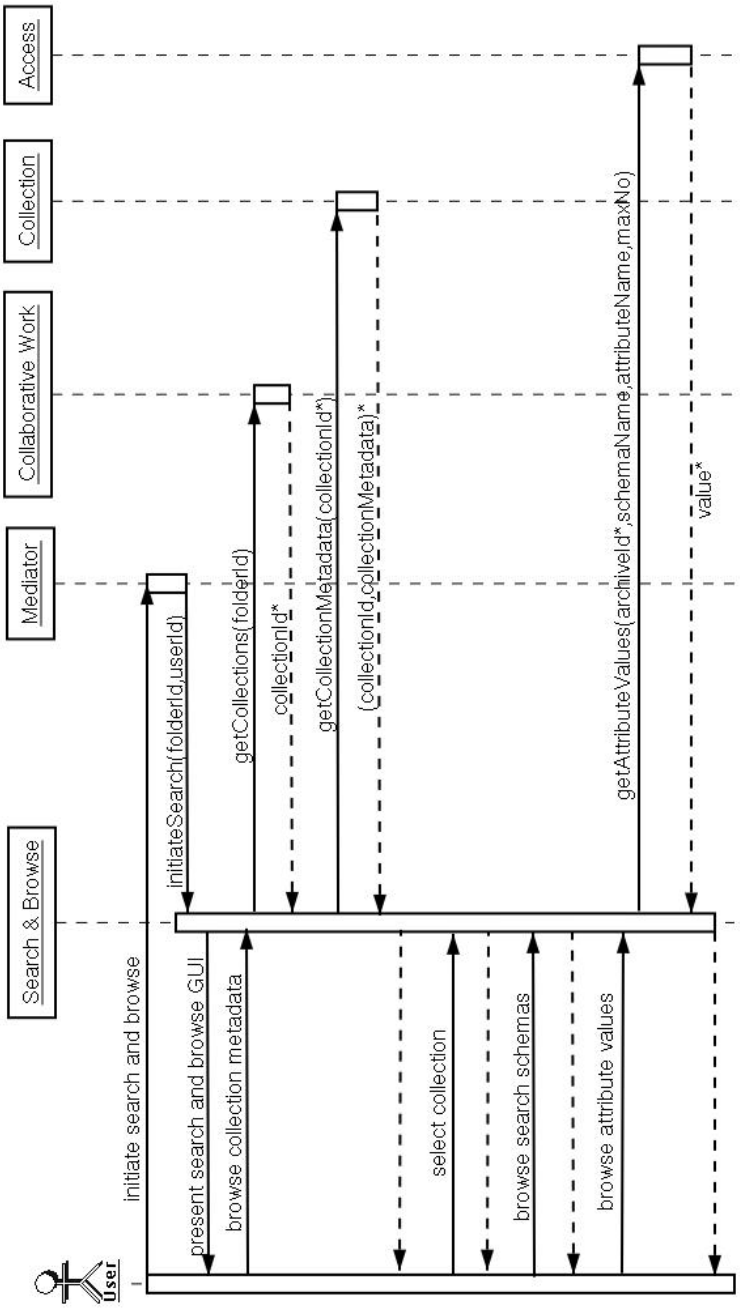

Figure 3.11: Search and browse, select collections and schemas - interaction diagram

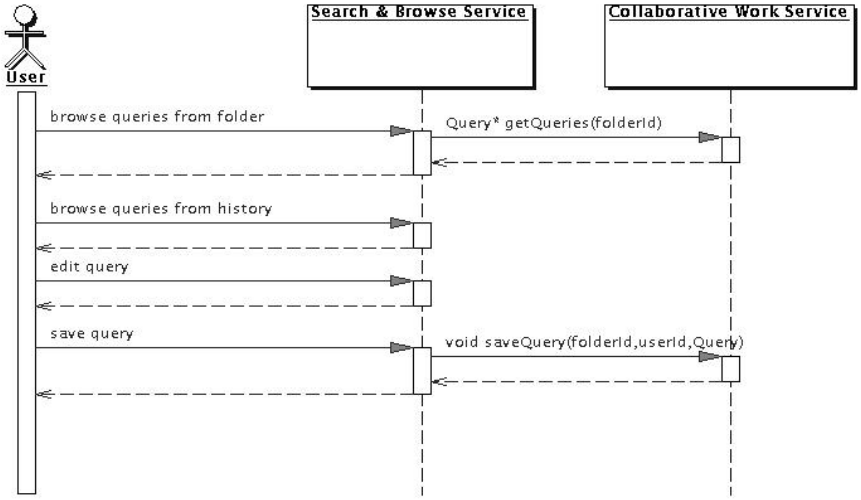

Figure 3.12: Search and browse, edit query - interaction diagram

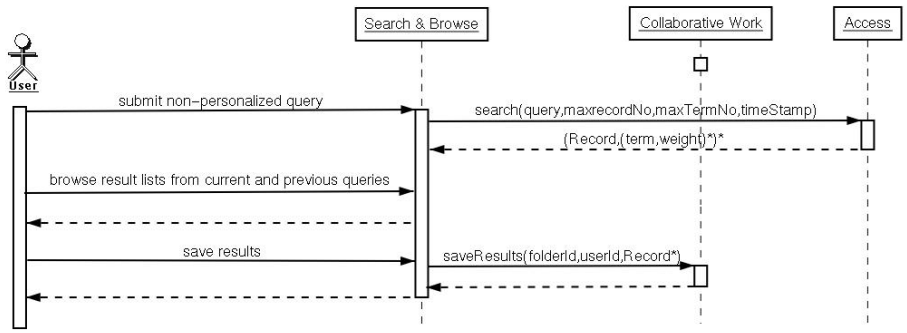

Figure 3.13: Search and browse, submit query without personalization - interaction diagram

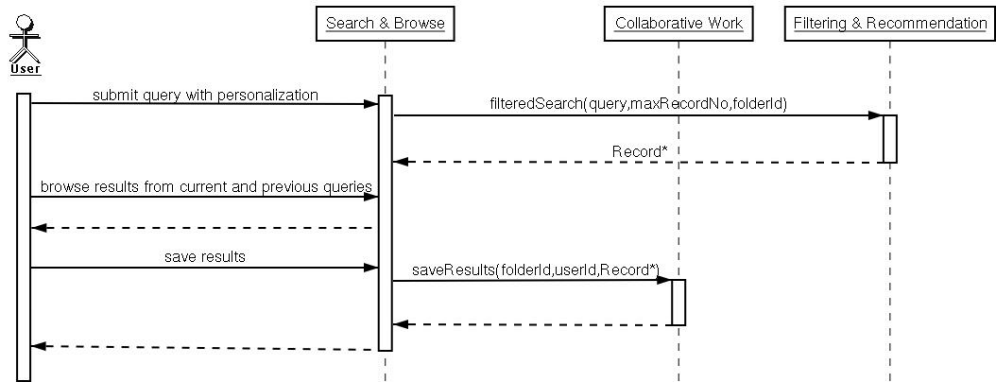

Figure 3.14: Search and browse, submit query with personalization - interaction diagram

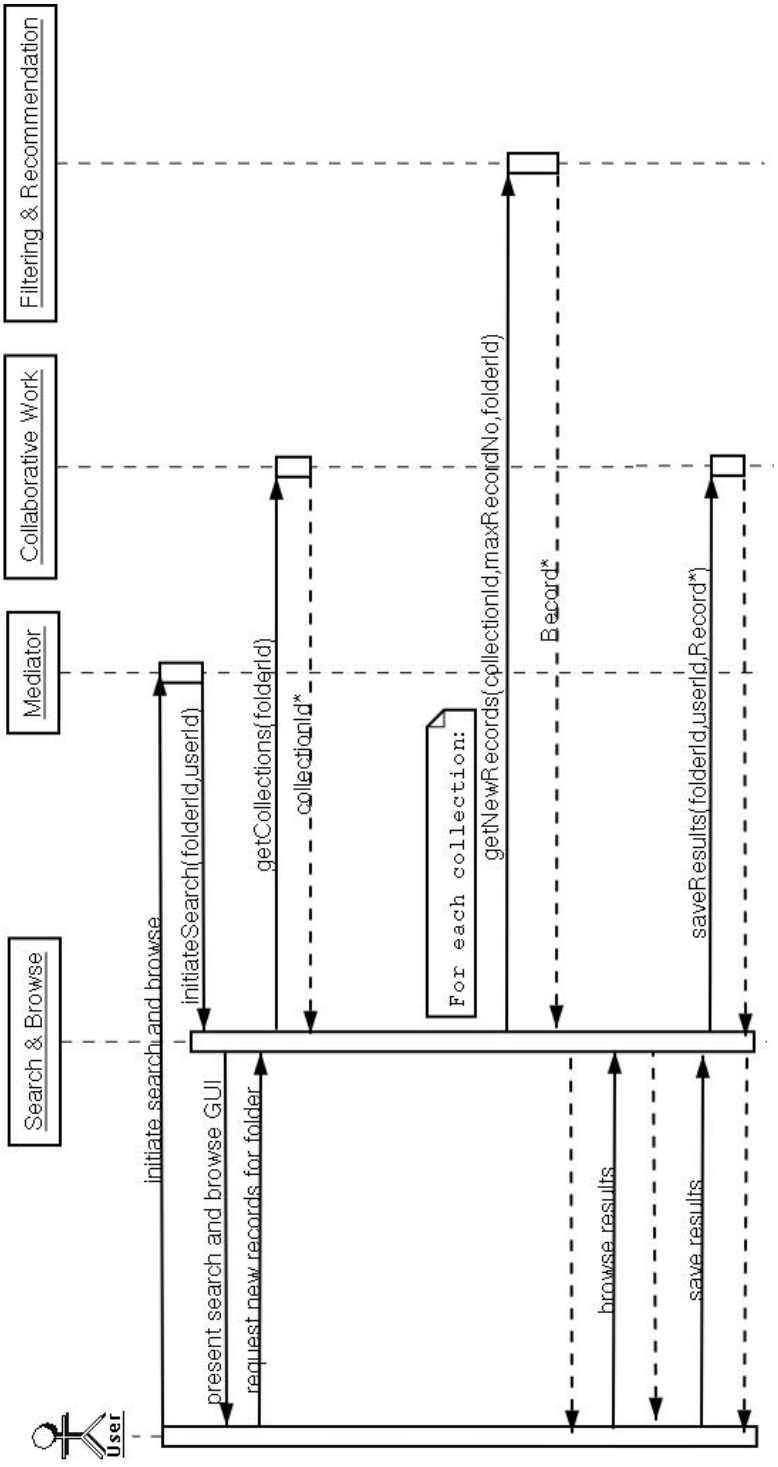

Figure 3.15: Personalized on-demand searching (SBS perspective) - interaction diagram

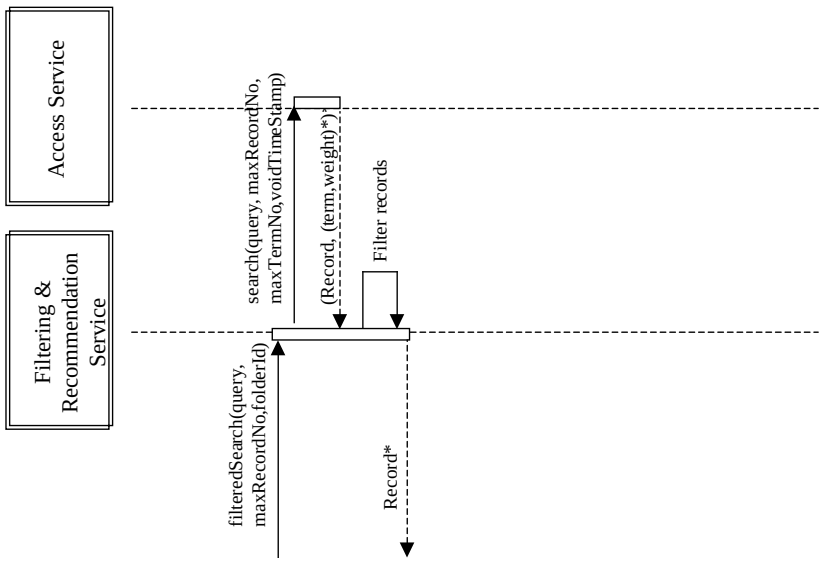

Figure 3.16: Personalized ad-hoc searching (FRS perspective) - interaction diagram

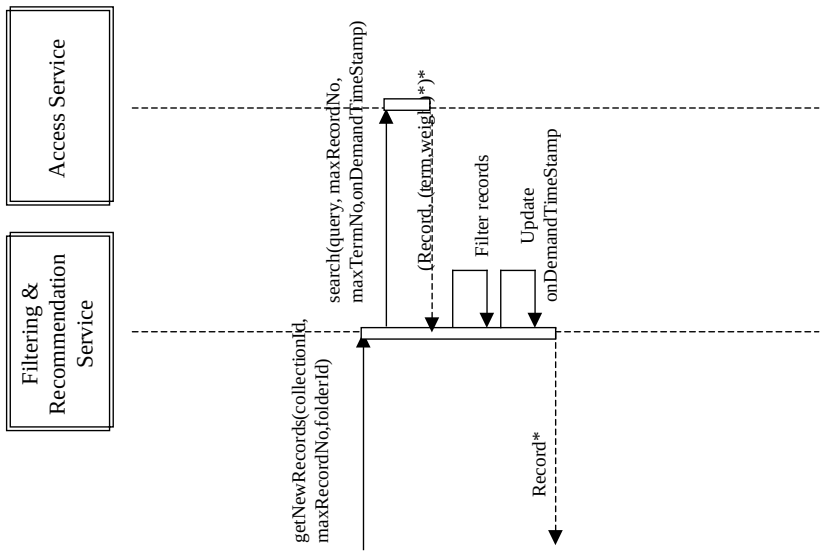

Figure 3.17: Personalized on-demand searching (FRS perspective) - interaction diagram

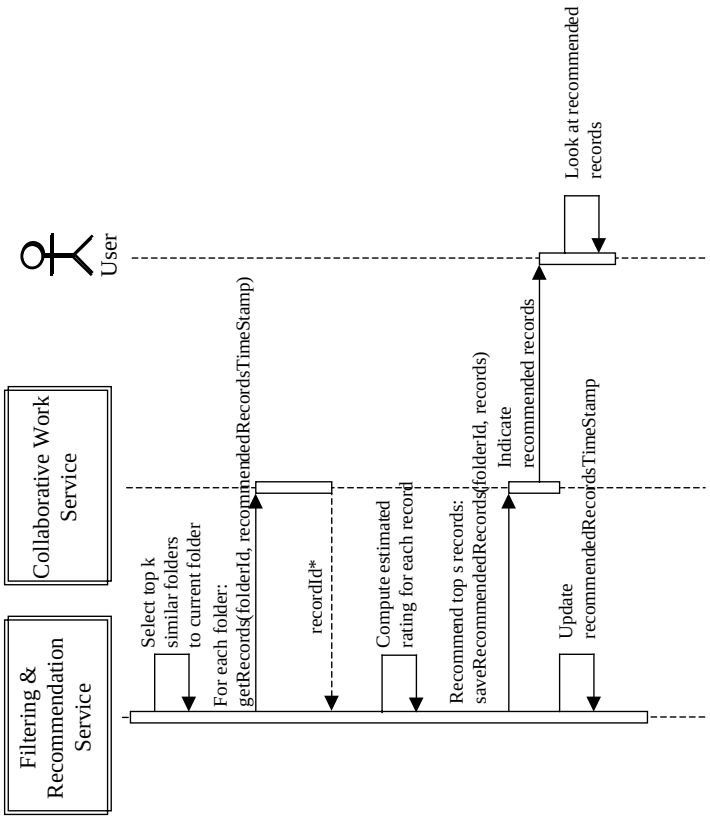

Figure 3.18: Recommend records - interaction diagram

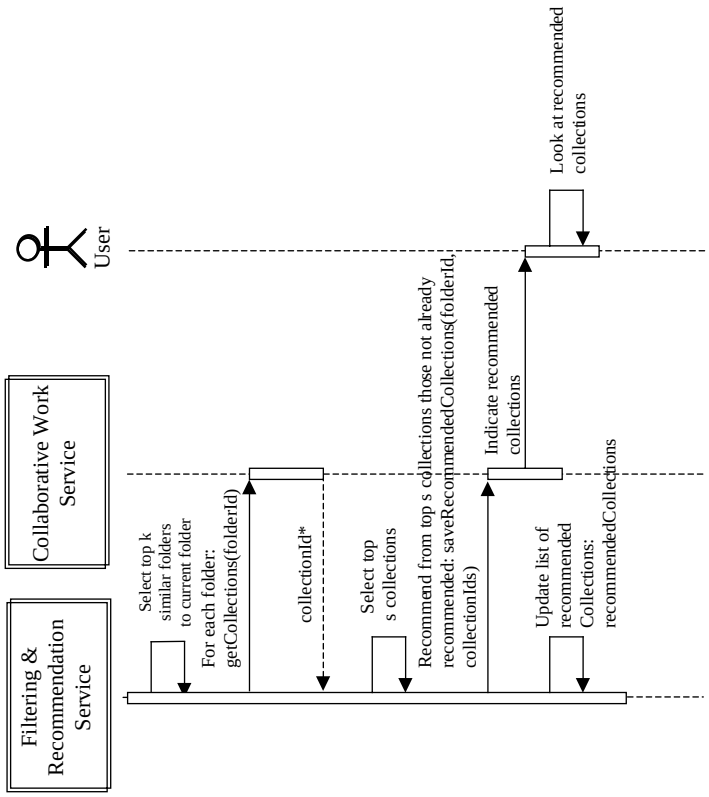

Figure 3.19: Recommend collections - interaction diagram

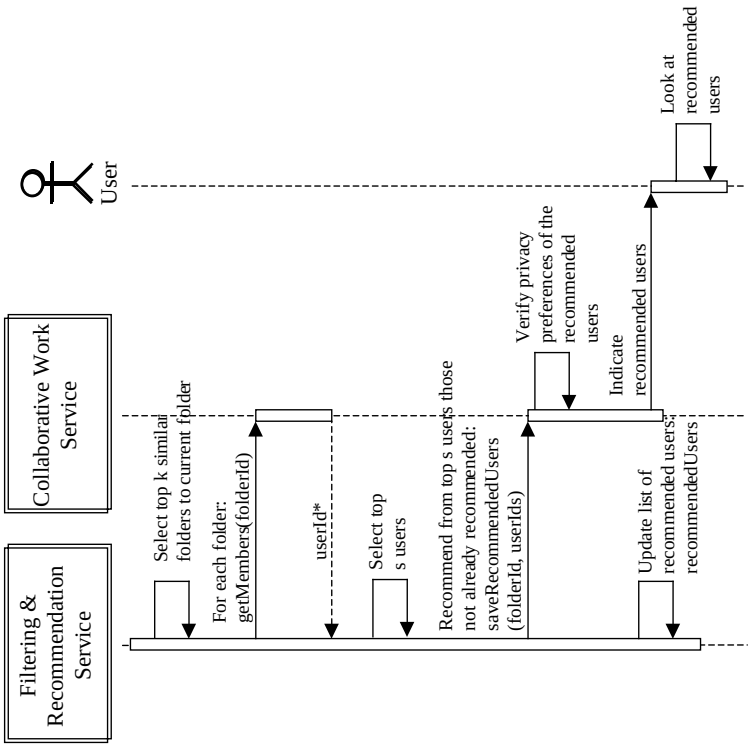

Figure 3.20: Recommend users - interaction diagram

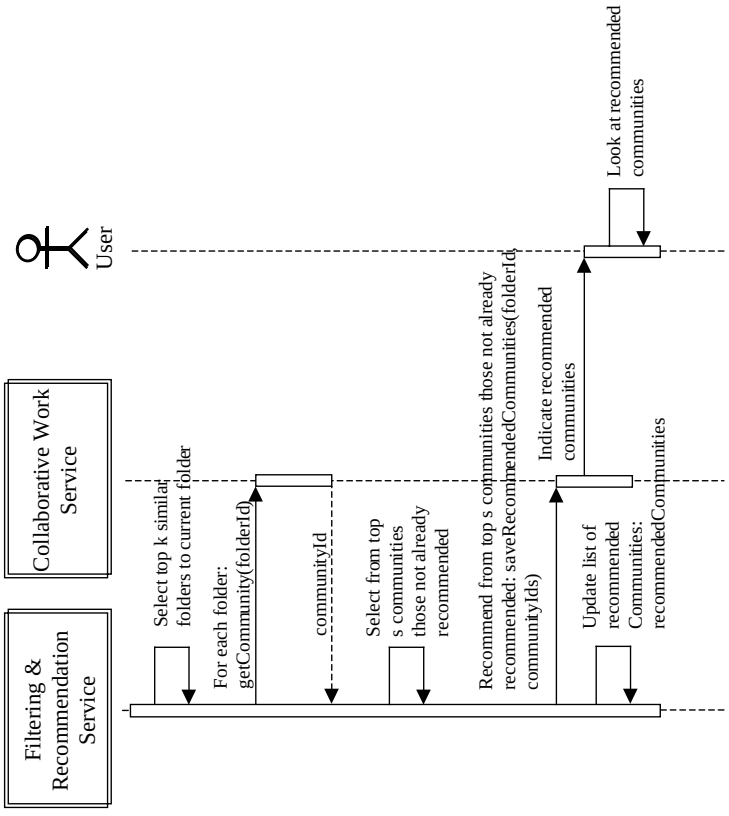

Figure 3.21: Recommend communities - interaction diagram

3.2.20 Registering an archive

The user requests to register an archive. After the Mediator Service has checked that the user has the appropriate rights to do so, it initiates a request to the Access Service and presents the Access Service's user interface.

- The user enters the URL of the new archive.
- The Access Service sends an Identify request to the archive and gathers metadata about the archive. If the request fails, the Access Service returns an error message to the user and ends the activity.
- If the Identify request is successful, the Access Service cross-checks the available metadate schemas for the archive with the available search functions in its database. Then, it presents the user the information about the archive that could be automatically retrieved, i. e. metadata schemas available, the search functions for those metadata schemas, the archive's long name etc. .
- The user enters further metadata about the archive, e. g. the languages covered, the dates covered, the overall topic of the archive, a list of keywords, a textual description, and selects the metadata schemas and search functions to be used with the archive.
- From this information, the Access Service creates an archive object and notifies the user that the archive has been registered successfully.
- According to an internal schedule, the Access Service gathers and indexes metadata records from the archive.
- After the first gathering action, the Access Service requests the Collection Service to create a default collection for the new archive.
- As soon as the default collection has been created, the Access Service requests the Collection Service to add a list of search and browse formats to the collection.

3.2.21 Creating a collection

A collection can be created as a side effect of the registration of a new archive (Case 1) or to satisfy the need of a community (Case 2). The two cases are identified by the type of membership condition: in the first case the membership condition is a simple condition on the field "archive"; any other membership condition is permitted in the second case. Different interaction diagrams apply to the two cases.

COL1 - Case 1

The Access Service sends a request for the creation of a new collection to the Collection Service. This request initiates the following process:

- The Collection Service generates an identifier for the new collection and sends it to the Access Service.
- The Access Service invokes the initialisation of the new collection by sending to the Collection Service the appropriate parameters.
- The Collection Service checks these parameters. If they are correct, it asks to the Access Service to provide a description of the archive that corresponds to the new collection.
- Once the Collection Service has received the requested information from the Access Service, it generates the collection specific metadata, creates a collection, and registers it as a new member of the set of existing collections.

- The Collection Service then notifies the identifier and the name of the new collection to the Collaborative Working Service.
- As final step, the Collection Service requires to the Access Service additional information about the content of the harvested archive and then transforms this information for its internal purposes.

COL1 - Case 2

The user requests to create a new collection. The Mediator Service passes this request to the Collection Service that that initiates the following process:

- The Collection Service generates an identifier for the new collection and displays to the user an appropriate form for submitting the requested parameters.
- The user fills the form and submits the initialisation request to the Collection Service.
- The Collection Service checks the received parameters. If they are not correct it sends an error message to the user. Otherwise, it generates the collection specific metadata, creates a collection, and registers it as a new member of the set of existing collections.
- The Collection Service then notifies the identifier and the name of the new collection to the Collaborative Working Service.

3.2.22 Deleting a collection

The user requests the deletion of a collection. The Mediator Service passes this request to the Collection Service that initiates the following process:

- The Collection Service displays to the user the list of collections that he/she can remove.
- The user selects one of them and submits the delete request to the Collection Service.
- The Collection Service removes the specified collection from the set of existing collections.
- The Collection Service then notifies the Collaborative Working Service about the removal of the collection.

3.2.23 Adding a search and browse format

The user requests to edit the collection related information. The Mediator Service passes this request to the Collection Service that initiates the following process:

- The Collection Service shows the list of collections whose information can be edited by the user. The user selects one of these collections.
- The Collection Service shows the list of collection metadata. This list contains the set of schemas available for searching and browsing on the specified collection.
- The user selects a reference schema, defines the subschema that better meets his/her needs and then submits the add format request.
- The Collection Service checks the parameters of the request. If they are not correct it returns an error message to the user. Otherwise, it updates the collection metadata by adding the new search and browse format.
- The Collection Service then notifies the Collaborative Working Service about the availability of the new format.

3.2.24 Removing a search and browse format

The user requests to edit the collection related information. The Mediator Service passes this request to the Collection Service that initiates the following process:

- The Collection Service shows the list of collections whose information can be edited by the user. The user selects one of these collections.
- The Collection Service shows the list of collection metadata. This list contains the set of formats available for searching and browsing on the specified collection.
- The user selects a format and submits a remove format request.
- The Collection Service updates the collection metadata by removing the specified search and browse format.
- The Collection Service then notifies the Collaborative Working Service about the applied removal.

3.3 Editing collection metadata

The user requests to edit the collection related information. The Mediator Service passes this request to the Collection Service that initiates the following process:

- The Collection Service shows the list of collections whose information can be edited by the user. The user selects one of these collections.
- The Collection Service shows the list of collection metadata that can be edited.
- The user selects one or more metadata fields and submits an edit request.
- The Collection Service updates the collection metadata according to the user request.
- The Collection Service then notifies the Collaborative Working Service about the applied modification.

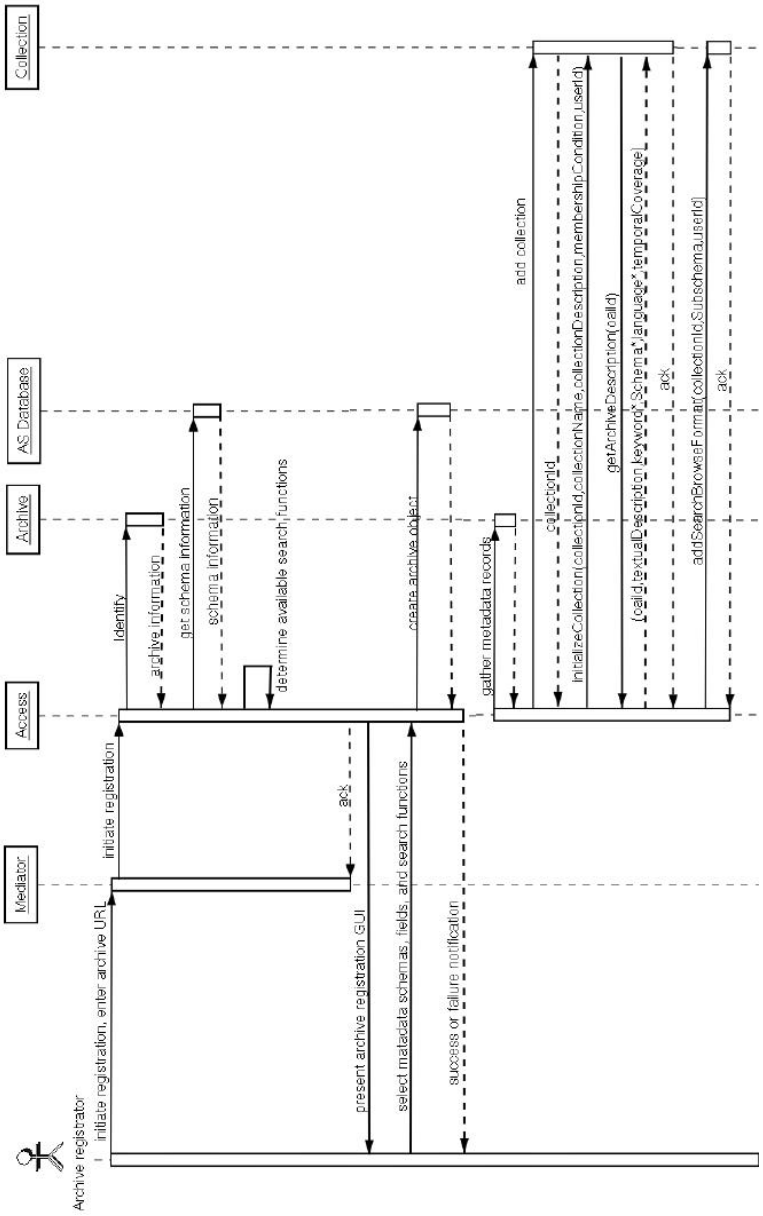

Figure 3.22: Register an archive - interaction diagram

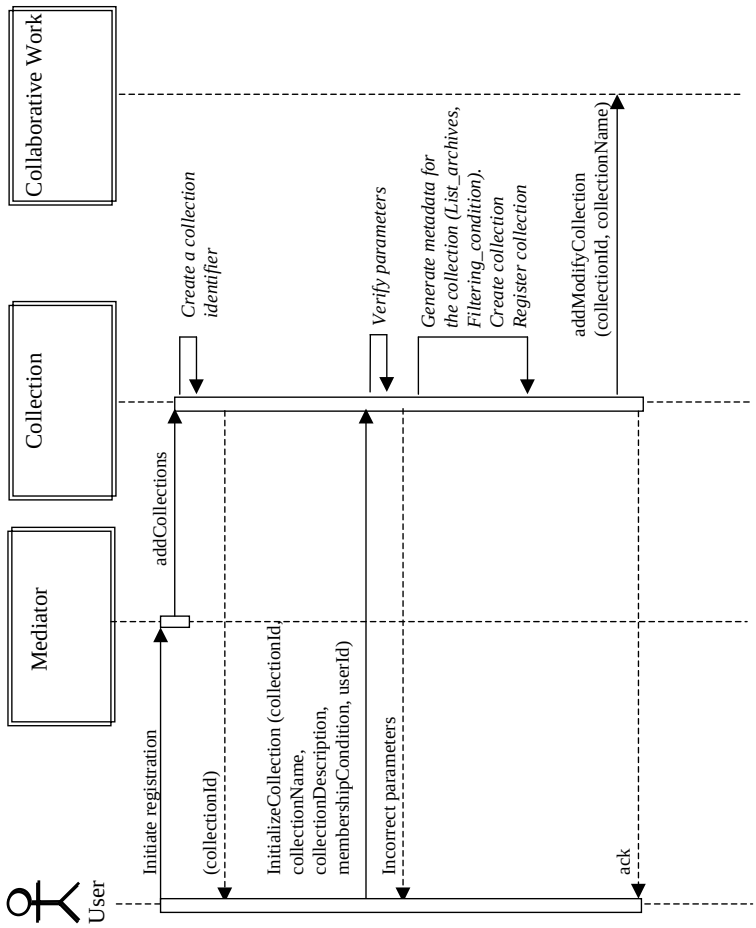

Figure 3.23: Create a collection (1) - interaction diagram

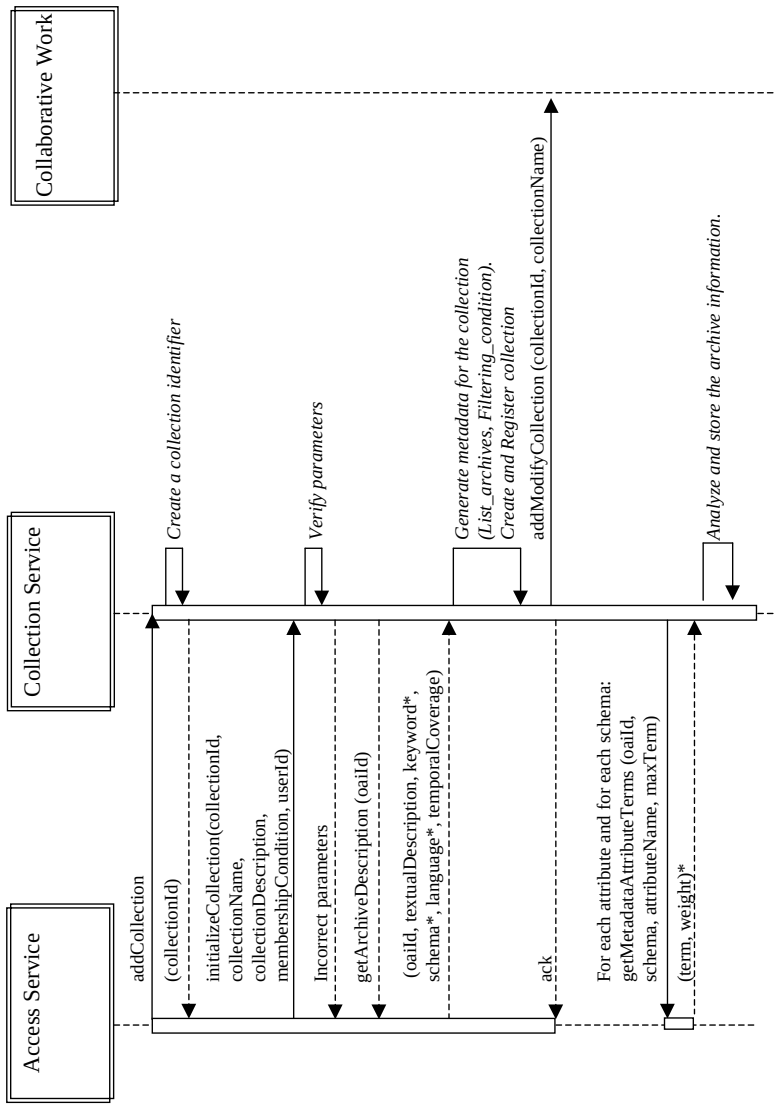

Figure 3.24: Create a collection (2) - interaction diagram

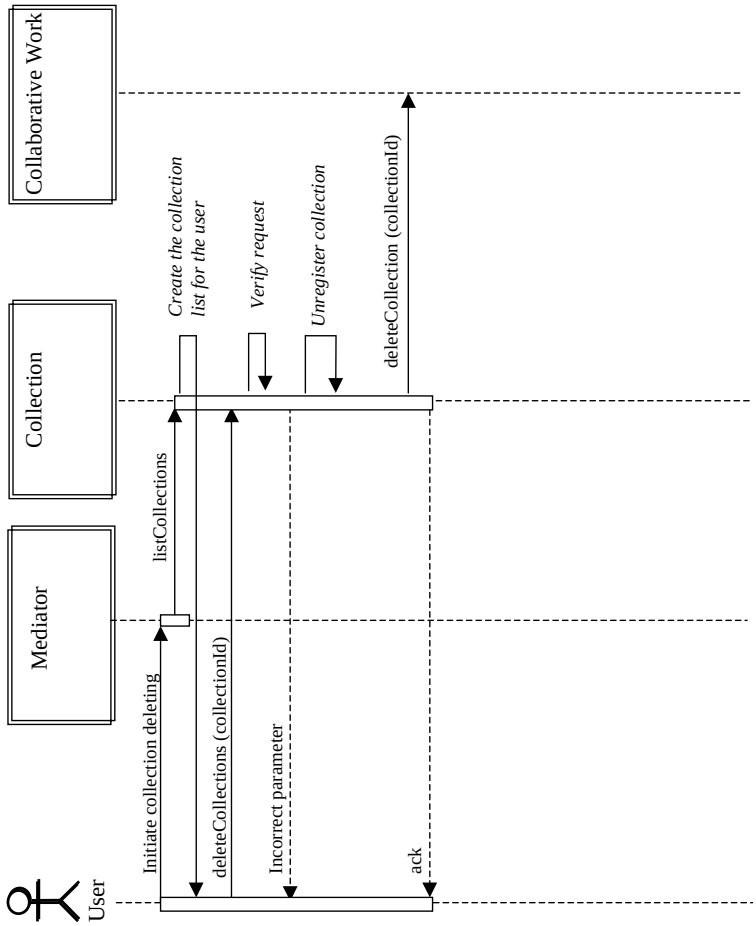

Figure 3.25: Delete a collection - interaction diagram

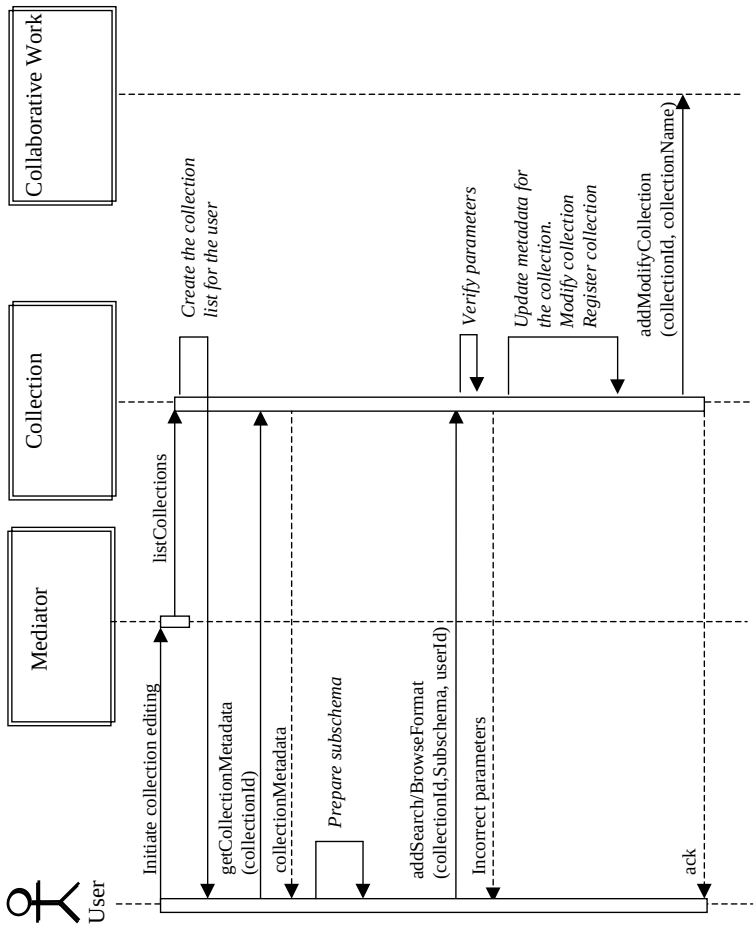

Figure 3.26: Add a search/browse format for a collection - interaction diagram

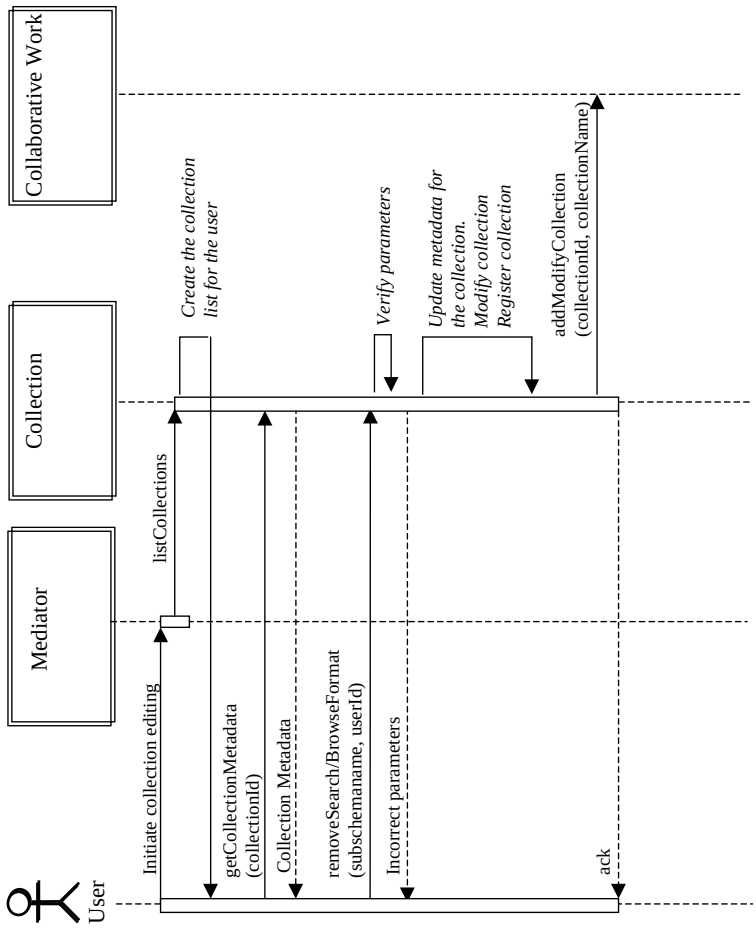

Figure 3.27: Remove a search/browse format from a collection - interaction diagram

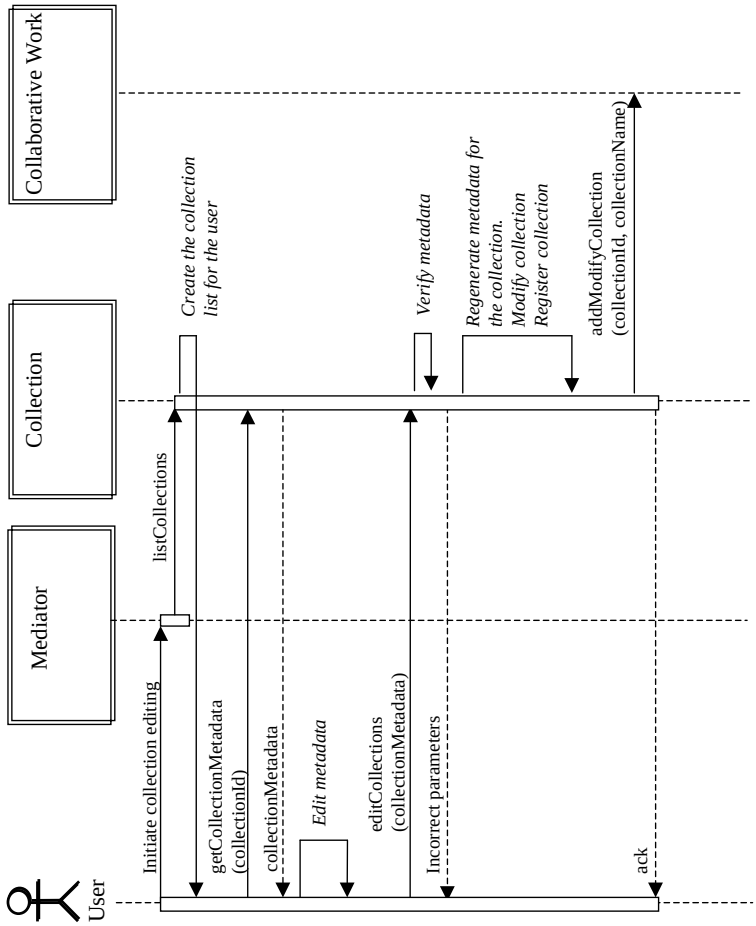

Figure 3.28: Edit collection metadata - interaction diagram

3.4 Class definitions in detail

In this section, we specify the relevant public classes.

For each class, only the publicly known attributes are listed. We assume that for each individual attribute, there is a get and a set method, for list type attributes additionally a add, select and delete method (for the handling of individual elements).

Class names begin with a capital letter, other variable and data type names with lower case letters. An asterisk after a data type indicates a list of this type (thus, Record* denotes a list of objects of type Record).

3.4.1 Access Service

The persistent data of this service are registered archives.

Archive

An instance of this class represents one single open archive.

- id
  Description: this is the unique ID of the archive
- textualDescription
  Description: a textual description of the archive, entered by the archive registrator
- schemas
  Description: a list of Schema objects, containing the metadata schemas that the archive exports
- url
  Description: the main URL of the archive
- mirros
  Description: a list of further URLs for the archive
- languages
  Description: a list of languages that the archive data covers, specified by the archive registrator
- temporalCoverage
  Description: a list of pairs (year,year) that specify the temporal coverage of the archive, the archive registrator enters this data during registration
- keywords
  Description: a list of keywords describing the archive content, entered by the archive registrator

AccessService

- id
  Description: the unique ID of the service
- archives
  Description: a list of Archive objects that the service manages
- (term,weight)* getIndexedTermsAndWeights(recordId*)
  Description: this method exports the indexed terms and their respective weights

- (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
- (oaiId,textualDescription,keyword*,schema*,language*,temporalCoverage) getArchiveDescription(oaiId)
  Description: this method returns a complete description of the archive specified by *oaiId*
- (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
- Schema* getSchemas(collectionId*)
  Description: this method exports a list of all schemas that the archives managed by this service supply
- 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*

Schema

An instance of this class describes one metadata schema.

- name
  Description: the unique name of the schema
- url
  Description: the URL of a DTD or namespace for the schema
- attributes
  Description: a list of the attributes of the schema, each attribute being a tuple (name,datatype)

3.4.2 Collection Service

The Collection Service is implemented by two classes: the CollectionService class and the Collection class.

CollectionService

- id
  Description: this is the unique identifier of the class
- collections
  Description: this is the set of registered collections

- collectionId initializeCollection(collectionId, collectionName, collectionDescription, membershipCondition, userId) **Description:** this method creates a collection if the membership condition is legal.
  Input. collectionId: the identifier of the new collection
  collectionName: the printable name of the collection
  collectionDescription: textual description of the collection
  membershipCondition: the condition to be verified by all the members of the collection
  userId: the identifier of the user which send the request
  Ouput. the identifier of the collection that has been initialised.

- void deleteCollection(collectionId, userId)
  Description: this method removes a collection from the set of existing collection if: a) the user is authorized to do it and b) the specified collection exists.
  Input. collectionId: the identifier of the collection
  userId: the identifier of the agent which send the request

- void addSearchBrowseFormat(collectionId, Subschema, userId)
  Description: this method add a new search/browse format if: a) the user is authorized to do it, b) the subschema is legal.
  Input. collectionId: the identifier of the collection
  Subschema: the specification of the subschema that is used for querying, browsing and displaying results
  userId: the identifier of the user that sends the request

- void removeSearchBrowseFormat(subschemaName, userdId)
  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.
  Input. subschemaName: the name of a search/browse format to be removed
  userId: the identifier of the user that sends the request

- (collectionId, collectionName)* listCollections
  Description: this method returns the list of existing collections
  Ouput. list of pair (collection identifier, collection name)

- (collectionId,collectionMetadata)* getCollectionMetadata(collectionId*)
  Description: For each specified collection identifier this method returns the corresponding descriptive metadata
  Input. collectionId*: a list of collection identifiers
  Ouput. a list of pairs. The first element of each pair is a collection identifier, the second is the corresponding collection metadata description.

Collection

- id
  Description: this is the unique Id of the class

- name
  Description: this is the printable name of the collection

- description
  Description: this is the a textual description of the collection

- membershipCondition
  Description: this is the condition that characterises the members of the collection. It is a condition on specific fields, like availability of metadata descriptions in a given format, or the identifier of the archive that maintains the documents, etc. All the documents that satisfy the membership condition up to an established threshold belong to the collections.

- archives
  Description: this is the list of archives that store the elements of the collection

- filteringCondition
  Description: this is the the condition that filters the members of the collections among those stored within the archives associated with the collections (see archive field above)

- Schemas
  Description: this is the set of Schema objects associated with the collection

- search/browseFormats
  Description: this is a set of Subschema objects that specifies the search and browse formats that are available on the collection

- ownerId
  Description:this is the identifier of the person that created the collection.

Subschema

- name
  Description: this is the unique name of the subschema

- referenceSchema
  Description: this the Schema object that describe the schema where the subschema is derived from

- url
  Description: the URL of a DTD or namespace for the subschema

- attributes
  Description: this is the set of attributes that characterises the subschema. Each attribute is a pair (name, datatype). The attributes must belong to the specified reference schema.

3.4.3 Search and Browse Service

The Search and Browse Service provides the search and browse functionality of the system. This service has no persistent data.

SearchAndBrowseService

This class implements the Search and Browse Service. It maintains a list of search and browse sessions.

- id
  Description: this is the unique ID of the service

- sessions
  Description: the list of search and browse sessions

- void initiateSearch(folderId,userId)
  Description: this method can be called to initiate a search and browse session for the given *folderId* and *userId*

SearchBrowseSession

One instance of this class corresponds to one user-service interaction sequence, i. e. this is the class that interacts with the user (through a separate user interface).

- id
  Description: the unique ID of the session
- expirationTime
  Description: the timestamp when this object will be deleted, i. e. when the session will expire
- userId
  Description: the ID of the user that initiated the search
- activeFolderId
  Description: the ID of the folder the search was initiated from
- queryHistory
  Description: the list of all queries formulated during this search session
- resultHistory
  Description: the list of result sets obtained during this search session, each result set being a set of Record objects
- relevantRecords
  Description: a list of those records marked as relevant by the user

3.4.4 Collaborative Work Service

The Collaborative Work Service stores the private, community and project folders of the registered Cyclades users along with their contents which may be metadata records, queries and other material (other material in project folders only). The service supports folder and record management, user registration, and collaboration between users by way of folder sharing in communities and projects, discussion forums and mutual awareness. The service has the classes CollaborativeWorkService, User, Folder, Record and Query. The persistent data of this service are users, folders, records, queries and collection names.

CollaborativeWorkService

- id
  Description: this is the unique ID of the service
- collectionNames
  Description: this is a list of pairs containing the ID and name of the currently existing collections.
- boolean startRegistration(emailAddress)
  Description: this method may be invoked in order to start the registration process for a user with a given e-mail address.
  Input: emailAddress: an e-mail address.
  Output: true, if no user has yet been registered under the e-mail address given, false otherwise.
- folderId* getFolders(userId)
  Description: this method may be invoked in order to get a list of IDs of folders to which a specific user has access.
  Input: userId: a user ID.
  Output: a list of folder IDs.

- name getName(folderId)
  Description: this method may be invoked in order to get the name of a folder.
  Input: folderId: a folder ID.
  Output: a string containing the folder name.

- description getDescription(folderId)
  Description: this method may be invoked in order to get the description of a folder.
  Input: folderId: a folder ID.
  Output: a string containing the folder description.

- recordId* getRecords(folderId, timestamp)
  Description: this method may be invoked in order to get the IDs of the records that have been saved into, or moved to, a folder since a certain time.
  Input: folderId: a folder ID.
  timestamp: a point in time coded as an ISO 8601 date/time in GMT.
  Output: a list of record IDs.

- label getClassifierLabel(recordId)
  Description: this method may be invoked in order to get the classifier label of a record.
  Input: recordId: a record ID.
  Output: a string containing the classifier label of a record; the string is empty if the record has no classifier label.

- userId* getMembers(folderId)
  Description: this method may be invoked in order to get the IDs of the users who have access to a folder.
  Input: folderId: a folder ID.
  Output: a list of user IDs.

- Query* getQueries(folderId)
  Description: this method may be invoked in order to get the queries that are contained in a folder.
  Input: folderId: a folder ID.
  Output: a list of queries.

- folderId getParent(folderId)
  Description: this method may be invoked in order to get the ID of the folder that contains a given folder.
  Input: folderId: a folder ID.
  Output: a folder ID or an empty string, if the given folder is not contained in another folder.

- folderId* getChildren(folderId)
  Description: this method may be invoked in order to get the IDs of the folders that are contained in a given folder.
  Input: folderId: a folder ID.
  Output: a list of folder IDs.

- collectionId* getCollections(folderId)
  Description: this method may be invoked in order to get the IDs of the collections that are associated to a folder.
  Input: folderId: a folder ID.
  Output: a list of collection IDs.

- communityId getCommunity(folderId)
  Description: this method may be invoked in order to get the ID of the community to which a folder belongs.
  Input: folderId: a folder ID.
  Output: a community ID or an empty string if the folder does not belong to a community.

- 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.
  Input: folderId: a folder ID.
  userId: a user ID.
  query: a query.

- 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.
  Input: folderId: a folder ID.
  userId: a user ID.
  records: a list of records.

- boolean saveRecommendedRecords(folderId, records)
  Description: this method may be invoked in order to save a list of recommended records in a folder.
  Input: folderId: a folder ID.
  records: a list of records.
  Output: false if recommendations for this folder are not welcome, true otherwise.

- boolean saveRecommendedUsers(folderId, userIds)
  Description: this method may be invoked in order to store a list of user recommendations in a folder.
  Input: folderId: a folder ID.
  userIds: a list of user IDs.
  Output: false if recommendations for this folder are not welcome, true otherwise.

- boolean saveRecommendedCollections(folderId, collectionIds)
  Description: this method may be invoked in order to store a list of collection recommendations in a folder.
  Input: folderId: a folder ID.
  collectionIds: a list of collection IDs.
  Output: false if recommendations for this folder are not welcome, true otherwise.

- boolean saveRecommendedCommunities(folderId, communityIds)
  Description: this method may be invoked in order to store a list of community recommendations in a folder.
  Input: folderId: a folder ID.
  communityIds: a list of community IDs.
  Output: false if recommendations for this folder are not welcome, true otherwise.

- void addModifyCollection(collectionId, name)
  Description: this method may be invoked in order to add a new collection to the list of collection names, or to modify the name of a collection already in the list.
  Input: collectionId: a collection ID.
  name: a string containing the name of the collection.

- void deleteCollection(collectionId)
  Description: this method may be invoked in order to remove a collection from the list of collection names.
  Input: collectionId: a collection ID.

Folder

- id
  Description: this is the unique ID of the folder.
- name
  Description: this is a string containing the name of the folder.
- description
  Description: this is a string containing the description of the folder.
- associatedCollections
  Description: this is a list of the IDs of the collections that are associated to a folder.
- recommendationsYesNo
  Description: this is a boolean set to true if recommendations for this folder are welcome, false otherwise.

Query

- id
  Description: this is the unique ID of the query.
- conditionsAndSourceSchema
  Description: this is a string containing the query conditions and the source schema for the query coded in XML.

Record

- id
  Description: this is the unique ID of the record.
- metadata
  Description: this is a string containing the metadata of the record coded in XML.
- classifierLabel
  Description: this is a string containing a label indicating the classification of the record by the Filtering and Recommendation Service. Only records which has been originally recommended by this service have a string value for this attribute, all other records' classifier label attribute is Null.

3.4.5 Rating Management Service

The Rating Management Service stores explicit user ratings of records and produces "projections" of these ratings to folders (all ratings of records contained in a particular folder), users (all ratings of a particular user), and records (all ratings that a particular record has received). Ratings are identified by the record that has been rated, the user that has rated, and the folder that contained the record when it was rated. The Rating Management Service has two classes, RatingManagementService and Rating. The persistent data of this service are the ratings.

RatingManagementService

- id
  Description: this is the unique id of the service.

- void saveRating(recordId, folderId, userId, ratingValue, timestamp)
  Description: this method may be invoked in order to store a rating in the service. The rating is specified by the ID of the record that has been rated, the ID of the folder that contained the record when it was rated, the ID of the user that rated, the rating value that was assigned to the record by the user, and the time when the rating happened.
  Input: recordId: a record ID.
  folderId: a folder ID.
  userId: a user ID.
  ratingValue: an integer representing the rating value.
  timestamp: a point in time coded as an ISO 8601 date/time in GMT.

- (recordId, userId, value)* getFolderRatings(folderId, timestamp)
  Description: this method may be invoked in order to get all ratings that the records contained in a given folder received since a given point in time. The ratings are specified by the ID of the record that was rated, the ID of the user who rated, and the rating value assigned to the record by the user.
  Input: folderId: a folder ID.
  timestamp: a point in time coded as an ISO 8601 date/time in GMT.
  Output: a list of triples containing a record ID, a user ID and a rating value each.

- (folderId, userId, value)* getRecordRatings(recordId, timestamp)
  Description: this method may be invoked in order to get all ratings that a given record received since a given point in time, regardless of the folders in which the record was contained when rated (note that the same record may be contained in several different folders even at the same time). The ratings are specified by the ID of the folder that contained the record when it was rated, the ID of the user who rated, and the rating value assigned to the record by the user.
  Input: recordId: a record ID.
  timestamp: a point in time coded as an ISO 8601 date/time in GMT.
  Output: a list of triples containing a folder ID, a user ID and a rating value each.

- (recordId, folderId, value)* getUserRatings(userId, timestamp)
  Description: this method may be invoked in order to get all ratings that a given user has assigned to arbitrary records since a given point in time. The ratings are specified by the ID of the record that was rated, the ID of the folder which contained the record when it was rated, and the rating value assigned to the record by the user.
  Input: userId: a user ID.
  timestamp: a point in time coded as an ISO 8601 date/time in GMT.
  Output: a list of triples containing a record ID, a folder ID and a rating value each.

Rating

- recordId
  Description: this is the ID of the record that was rated.

- folderId
  Description: this is the ID of the folder that contained the record when it was rated.

- userId
  Description: this is the ID of the user who rated.

- value
  Description: this is an integer representing the rating value that was assigned.

- timestamp
  Description: this is the time when the rating happened.

3.4.6 Filtering and Recommendation Service

The Filtering Service and Recommendation Service (FRS) has a class FilteringRecommendation-Service and a class FolderProfile. The persistent data of this service are the folder profiles.

FilteringRecommendationService

- id
  Description: this is the unique ID of the class

- folderProfileList
  Description: this is the list of all folderProfiles known to the Filtering and Recommendation Service

- 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 learnd from the folder.
  Input. query: the query according to the syntax specified by the access service
  maxRecordNo: maximal number of records to be retrieved
  folderId: the folder ID w.r.t. to filter
  Output. list of records

- Record* getNewRecords(collectionId,maxRecordNo,folderId)
  Description: this method may be invoked in order to get new records, retrieved with respect to the profile learnd from the folder.
  Input. collectionID: a collection ID
  maxRecordNo: maximal number of records to be retrieved
  folderId: the folder ID w.r.t. to filter
  Output. list of records

- void updateFolderProfile(folderId)
  Description: this method may be invoked in order to update the folder profile, i.e. to learn the folder profile.
  Input. folderId: the folder ID for which to learn the profile

- void setRecommendationYesNo(folderId,value)
  Description: this method may be invoked in order activate/disactivate the production of recommendations w.r.t. a folder
  Input. folderId: the folder ID
  value: true=recommend item to that folder, else do not recommend items

FolderProfile

- id
  Description: this is the unique ID of the class

- profile
  Description: this is the folder profile computed taking into account folder content and user ratings

- recommendedRecordsTimeStamp
  Description: this is a hash table, that for a give folder mantains a timestamp which is the most recent time where a folder F has been considered among the top k similar folders to this current one. The purpose is to avoid duplicated record recommendations

- recommendedCollections
  Description: list of recommended collections for avoiding duplicated recommendation
- recommendedUsers
  Description: list of recommended users for avoiding duplicated recommendation
- recommendedCommunities
  Description: list of recommended Communities for avoiding duplicated recommendation
- onDemandTimeStampList
  Description: this is a list of pairs (owner, onDemandTimeStamp). We consider a list, since in the case of a community folder more than one user may have access to this folder.
- profileUpdateTimeStamp
  Description: time and date of last profile update, related to the content of the folder.
- ratingUpdateTimeStamp
  Description: time and date of last rating update, related to the ratings of the folder.

3.4.7 Mediator Service

The Mediator Service (MS) has a class MediatorService and a class Service. The persistent data of this service are the various services in the System and the users that have registered the System.

MediatorService

- id
  Description: this is the unique ID of the service.
- (servId, version, address, quality)* getService(type)
  Description: this method is used in order to get a list of services of particular type.
  Input: type: 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).
- description getServiceDescription(servId)
  Description: this method is used in order to get the description of a service.
  Input: servId: a service ID.
  Output: the (short) description of the particular service.
- errorLog getErrorLog(servId)
  Description: this method is used in order to get the error log file of a service.
  Input: servId: a service ID.
  Output: the error log file of the particular service.
- void reportError(servId, errorLogs)
  Description: this method is used in order to report an error(s) for a service.
  Input: servId: a service ID.
  errorLogs: a list of error logs, to be added to the already existing error log file of the service.
- serviceId addService(version, address, type, description)
  Description: this method is used in order to add a service to the system.
  Input: version: the version of the service.
  address: the machine address.
  type: the type of the service.
  description: a short description of the service.
  Output: a service ID.

- void deleteService(servId)
  Description: this method is used in order to delete/remove a service from the system.
  Input: servId: a service ID.

- void updateService(servId, version, address, description)
  Description: this method is used in order to update the information (version, machine address, description) of a service.
  Input: servId: a service ID.
  version: the (new) version of the service.
  address: the (new) machine address.
  description: a (new) short description of the service.

- void resetErrorLog(servId)
  Description: this method is used in order to reset the error log file of a service.
  Input: servId: a service ID.

- void setCollectionAccesRight(userId, collectionId, ON/OFF)
  Description: this method is used in order to set a user's access rights for a collection.
  Input: userId: a user ID.
  collectionId: a collection ID.
  ON/OFF: a boolean indicating whether to enable or disable the access right.

- void addUser(userId, userName, password)
  Description: this method is used in order to add a, newly registered, user to the system.
  Input: userId: a user ID.
  userName: a user name.
  password: the password for that user.

- UserId* getUserIds()
  Description: this methods is used in order to obtain the ids of all users.
  Output: a list containing the IDs of all users.

  Perhaps the following method should be added so as the CWS can directly set the current folder of a user (comments accepted)

- void setUserCurrentFolder(userId, folderId) **Description:** this method is used in order to set the current folder for a user, as that user navigates through the various folders.
  Input: userId: a user ID.
  folderId: a folder ID.

Service

- servId
  Description: this is the unique ID of the service.

- version
  Description: this is the version of the service.

- address
  Description: this is the machine address of the service, so as others can communicate with it.

- quality
  Description: a number between 0-1, indicating the quality of the service.

- type
  Description: this is the type of the service.

- description
  Description: this is the (short) description of the service.
- errorLog
  Description: this is for descriping-holding the various errors that occur.

3.5 Query language

A query consists of a list of criteria, each criterion being a tuple of (name of a schema, attribute name, reference value, predicate, weight) combined by an implicit OR.

The schema name and attribute name specify which field of a record should be examined, the predicate and reference value specify how the field content should be interpreted, and the weight gives the relative importance of this criterion regarding the other criteria. The predicates allowed depend on the data type of the attribute (equality, phonetic similarity, less than, greater than, ...).

Example query:

(DC,creator,"Smith",eq,.7),(DC,contributor,"Smith",eq,.3)

Look for records that a person called "Smith" created or contributed to. Retrieve rather those records where "Smith" is the creator.

Chapter 4

System architecture

This chapter describes the architecture of the Cyclades system.

4.1 Services

The Cyclades system consists of the following services, as outlined in chapter 3:

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

The Access Service interacts with the underlying metadata archives, currently with archives adhering to the Open Archives specification.

The Collaborative Work Service provides a folder environment for managing records, queries, collections, documents, and annotations, supports collaboration between users by way of 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, formulating and reusing queries, and browsing schemas, attribute values, and metadata records.

The Collection Service manages collections, thus allowing a partinioning 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, recommendations, and personalized folder management.

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.

4.2 Replication and distribution

In the first version, i. e. the system as it will be implemented in this project, the infrastructure will provide for replication of the services, but not for distribution.

Each service registers itself with the Mediator. Whenever a service needs to communicate with another service, it asks the Mediator for a list of services of the appropriate type. In this version, when asked for, say, Search and Browse Services, the Mediator will return a list of Search and Browse Services, and the requesting service can be sure that each Search and Browse Service of this list will provide the same funtionality.

The registry is extendable, so that in future versions, also distribution may be supported.

4.3 User interfaces

Most of the services provide their own user interface, i. e. the Collaborative Work Service, the Search and Browse Service, the Access Service (for archive management), and the Collection Service (for collection management). The Mediator itself provides the registration and login interface, and a system administration interface (for assigning access rights etc.). Additionally, the Mediator 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.

4.4 Service autonomy

The Cyclades services can run on different machines and need only a HTTP connection to communicate and collaborate.

Still, they do rely on each other in some cases. In this section, we discuss which services are necessarily needed by which other services.

4.4.1 Access Service

Theoretically, the Access Service can run without the other services present. Via the user interface, it would still be possible to register and manage archives. Without a Collection Service, the registration of an archive would just not result in the creation of the corresponding default collection. Without a Search and Browse Service, respectively a Filtering and Recommendation Service, there would be no call to the Access Services search functionality, thus, to make the Access Service truely autonomous, it would need an extended user interface that would have to support querying, too.

4.4.2 Collaborative Work Service

As we have seen, the CWS is primarily responsible for supporting folder management, storing records, as well as providing community and project functionality. The folder management for the private folders of single users is basically independent of other Cyclades services. The record management is highly dependent on the Search and Browse service, the Access service, as well as the Collection service. As far as the functionality for community and project folders is concerned the Filtering and Recommendation service is particularly important for providing users of community folders with recommendations about records, other users, collections, and communities. Also, the Collection service is needed when a user wants to assign collections to folders.

On a whole, the CWS interacts with most other Cyclades services. Nevertheless, through the open architecture and the open communication protocols of Cyclades the CWS could run with reduced functionality without the other services (except for the Search and Browse as well as the Collection

service, without which no records can be accessed). Additionally, the CWS could be combined with other services that either substitute one of the existing services or provide new functionality.

4.4.3 Rating Management Service

The RMS is an internal service that provides functionality to the other services—particularly, to the Filtering and Recommendation service. The users enter the ratings in the Collaborative Work service; so, the RMS needs the input from the Collaborative Work service. On the other hand the RMS provides its internal services to the Filtering and Recommendation service.

So, the RMS needs the Collaborative Work service for input and needs the Filtering and Recommendation service in order to make sense. It could also receive input and provide service for other (future) Cyclades services.

4.4.4 Search and Browse Service

As the Search and Browse Service itself does not have any persistent data, it relies on the existence of an Access Service, at least.

In order to find this Access Service, and in order to be invoked in the first place, the Search and Browse Service also requires a Mediator Service. Without the Mediator, the Search and Browse Services might run in a modified version, with the address of the Access Service specified in a configuration file, and an appropriate extention of the user interface.

Without a Collection Service, the Search and Browse Service could not limit searches to one collection, but otherwise, it would function unchanged.

Without a Filtering and Recommendation Service, the Search and Browse Service could not provide personalized searching and browsing.

Finally, without a Collaborative Work Service, the Search and Browse Service would function like a Web search engine, without the possibility to store the query results persistently.

Thus, to sum it up, the Search and Browse Service requires only an Access Service and is able to function in a useful way even without the other services.

4.4.5 Collection Service

The Collection Service provides to the user of the Cyclased system a view of an information space dynamically organised into a set of virtual collections. It also provides, on demand, information about the defined collections that can be exploited by the other architectural components to implement higher quality services.

In order to implement its functionality, the Collection Service needs to know which are the characteristics of the underlying information space. To this aim it uses the information gatherer by the Access Service.

4.4.6 Filtering and Recommendation Service

As we have seen, the Filtering and Recommendation Service provides a set of functionalities which may be partitioned into two sets: (i) those supporting filtering of records on user demand; and (ii) those supporting recommendation of records, users, collections and communities. Filtering and recommendation is based both on the user's folder profile and on the ratings given by the users to the records. Therefore, a prerequisite for the FRS to work properly is that user's folders are given. The service may work (i) in case only ratings are given to records belonging to folders; or (ii) only the records belonging to folders are given; or (iii) both case above are considered. Consequently,

the FRS may be used by any other new service (or cluster of services) subscribing to the Cyclades environment, if it provides the necessary data through the methods that the FRS service uses in order work (see the interaction diagrams of cases filtering, recommendation and profile updating). As the code is open source, flexibility and customisation is guaranteed.

4.4.7 Mediator Service

The Mediator Service is the service of the Cyclades System, which integrates the other services (by enabling them to communicate with each other). Also it is the service that is responsible for "Service Registration" to the Cyclades System, meaning that new services can "join" the System by contacting the Mediator and executing, following the communication protocol, the appropriate method of the Mediator Service and specifying the parameters needed. The latter means that the Mediator Service can function as a "Library of Services", that is able to dynamically change (new services can be added, existing services can be removed). A "Library of Services" that also provides the means of communication between the various services, in the sense that the services do not have to know which other services are there in the library, but they can learn from the Mediator, when they need to do so, all the appropriate information in order to communicate with them (they have to follow the communication protocol).

So, concerning the service dependencies, it can be assumed that the Mediator Service can stand alone in the Cyclades System, because its functionality does not depend on other services' functionality. Even if a service is not available, the Mediator will be able to provide its services and "work". The only dependency is on the Collaborative Work Service (CWS), in the case of a user registration to the Cyclades System. When a user wants to register with the Cyclades System, the Mediator interacts with the Collaborative Work Service so as to obtain that user's id in the System (which is created by the CWS.)

4.5 Customization and extensibility

As a default, a Cyclades system consists of a set of the services described above. It can be used to work with any or all open archives available. By restricting it to certain archives, or by leaving out one or more of the services, the system can easily be adapted to more specialized communities where only a part of the data or the functionality is needed.

The minimal configuration is an archives search engine, consisting of one Search and Browse Service and one Access Service. If a Mediator Service and a Collaborative Work Service are added, the system could be called an archives record gathering and exchange system; its extended functionality would include persistent storage of search results, collaboration and user registration. In further steps, a Collection Service and/or a Filtering and recommendation Service could be added to the system for the support of user-defined collections, personalized filtering and recommendations.

The other case, when more functionality or different data is needed, can be dealt with by adding further services, e. g. another kind of Access Service to treat other kinds of archives, or e. g. a mapping service for word mapping between different languages. Such new services can be added easily, they just have to register with the Mediator and use the public methods of the other services.

4.6 Functionality and efficiency tests

This section specifies the (i) functionality and (ii) efficiency tests for the CYCLADES system validation to be carried out in the framework of WP5.

The functionality tests will consists in a obvious set of tests aiming to verify whether all use cases described in Chapter 1 maybe successfully be completed. We omit the detailed list, as it is straightforwardly determined "going" through the use cases.

The aim of efficiency testing is to quantify the component and overall system performance. In this respect, we will

1. measure elapsed average time between request and answer during the functionality tests;
2. evaluate system scalability to large volumes of data;
3. evaluate system stability under repetition in different time moments;

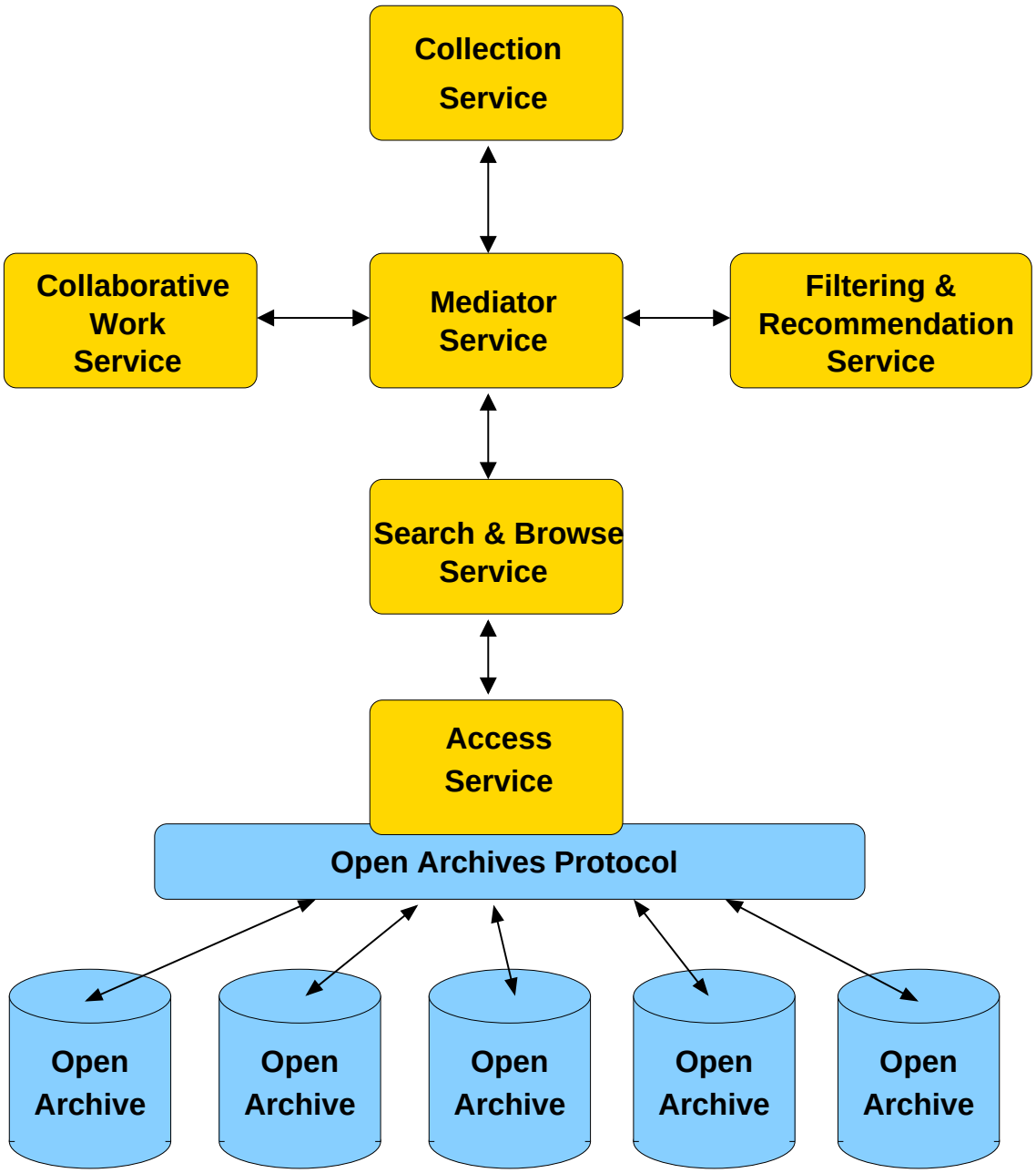

Figure 4.1: Cyclades services

Chapter 5

Communication protocol

This chapter defines the communication protocol to be used within the Cyclades system.

5.1 Remote procedure calls via XML-RPC

The services of the Cyclades system will communicate via HTTP, using XML-RPC [1].

An XML-RPC message is an HTTP-POST request. The body of the request is in XML.

A procedure executes on the server and the value it returns is also formatted in XML. Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

5.1.1 Method call

Header requirements

- The format of the URI in the first line of the header is not specified. For example, it could be empty, a single slash, if the server is only handling XML-RPC calls. However, if the server is handling a mix of incoming HTTP requests, we allow the URI to help route the request to the code that handles XML-RPC requests. (In the example, the URI is /RPC2, telling the server to route the request to the "RPC2" responder.)
- A User-Agent and Host must be specified.
- The Content-Type is text/xml.
- The Content-Length must be specified and must be correct.

Body

The body contains XML, a single <methodCall> structure.

The <methodCall> must contain a <methodName> sub-item, a string, containing the name of the method to be called.

If the procedure call has parameters, the <methodCall> must contain a <params> sub-item. The <params> sub-item can contain any number of <param>s, each of which has a <value>.

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

Example

Get the IDs of the children of the folder with the ID 42:

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: ...


<?xml version="1.0"?>
<methodCall>
  <methodName>FilteringRecommendationService.getChildren</methodName>
  <params>
    <param>
      <value><i4>42</i4></value>
    </param>
  </params>
</methodCall>
```

5.1.2 Method response

Header

- Unless there's a lower-level error, always return 200 OK.
- The Content-Type is text/xml. Content-Length must be present and correct.

Body

The body of the response is a single XML structure, a <methodResponse>, which can contain a single <params> which contains a single <param> which contains a single <value>.

The <methodResponse> could also contain a <fault> which contains a <value> which is a <struct> containing two elements, one named <faultCode>, an <int> and one named <faultString>, a <string>.

A <methodResponse> can not contain both a <fault> and a <params>.

Example

Return the IDs requested in the method call example (assuming that the children of folder 42 have the IDs 57 and 110):

```
HTTP/1.1 200 OK
Connection: close
Content-Length: ...
Content-Type: text/xml
Date: Tue, 31 Jul 2001 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT


<?xml version="1.0"?>
<methodResponse>
```

```
   <params>
      <param>
         <value>
           <array>
             <data>
               <value><i4>57</i4></value>
               <value><i4>110</i4></value>
             </data>
           </array>
        </value>
      </param>
   </params>
</methodResponse>
```

5.2 Other communication protocols

At W3C, there is yet work in progress on XML-based communication protocols. Maybe in the future, it will be necessary or convenient to change to another protocol (e. g. in order to integrate other services). Therefore, each Cyclades service is going to have a separate communication layer (implemented as a separate module) which will hide the actual protocol from the service. In this way, the Cyclades services will be easily adapted to other communication protocols.

List of Figures

2.1 Register as a new user - activity diagram 14
2.2 Create a folder - activity diagram 16
2.3 Search and browse - activity diagram 23
2.4 Rate a record - activity diagram 27
2.5 Create a collection - activity diagram 31
2.6 Add a search/browse format for a collection - activity diagram 32
2.7 Register an archive - activity diagram 34

3.1 Registration (Mediator perspective) - interaction diagram 37
3.2 Login - interaction diagram 38
3.3 Registration (CWS perspective) - interaction diagram 39
3.4 On-demand folder profile update (CWS perspective) - interaction diagram 40
3.5 Activate recommendations - interaction diagram 41
3.6 Deactivate recommendations - interaction diagram 42
3.7 Invitation - interaction diagram 43
3.8 Rate records - interaction diagram 44
3.9 On-demand folder profile modification - interaction diagram 50
3.10 Scheduled folder profile modification - interaction diagram 51
3.11 Search and browse, select collections and schemas - interaction diagram 52
3.12 Search and browse, edit query - interaction diagram 53
3.13 Search and browse, submit query without personalization - interaction diagram . . 54
3.14 Search and browse, submit query with personalization - interaction diagram 55
3.15 Personalized on-demand searching (SBS perspective) - interaction diagram 56
3.16 Personalized ad-hoc searching (FRS perspective) - interaction diagram 57
3.17 Personalized on-demand searching (FRS perspective) - interaction diagram 58
3.18 Recommend records - interaction diagram 59
3.19 Recommend collections - interaction diagram 60
3.20 Recommend users - interaction diagram 61
3.21 Recommend communities - interaction diagram 62
3.22 Register an archive - interaction diagram 66
3.23 Create a collection (1) - interaction diagram 67
3.24 Create a collection (2) - interaction diagram 68

3.25 Delete a collection - interaction diagram 69
3.26 Add a search/browse format for a collection - interaction diagram 70
3.27 Remove a search/browse format from a collection - interaction diagram 71
3.28 Edit collection metadata - interaction diagram 72

4.1 Cyclades services 92